Юлий Кетков
Александр Кетков

САМОУЧИТЕЛЬ
Практика программирования: Бейсик, Си, Паскаль

Санкт-Петербург

Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Содержится более 130 готовых к исполнению программ, большинство из которых представлено на трех алгоритмических языках — Бейсике, Си и Паскале. Все разделы предваряются описанием соответствующих конструкций каждого алгоритмического языка. При этом особое внимание обращается на общность языковых средств рассматриваемых систем программирования — QBasic, Turbo C (Borland C++) и Turbo Pascal. Текстам программ предшествуют советы по их разработке с учетом специфики того или иного алгоритмического языка и описание наиболее характерных особенностей.

Для учащихся старших классов и студентов вузов,
а также для преподавателей информатики

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Ольга Михайлова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Кетков Ю. Л., Кетков А. Ю.
Практика программирования: Бейсик, Си, Паскаль. Самоучитель. — СПб.: БХВ-Петербург, 2001. — 480 с.: ил.

ISBN 5-94157-104-6

© Ю. Л. Кетков, А. Ю. Кетков, 2001
© Оформление, издательство "БХВ-Петербург", 2001

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.07.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Предисловие **1**

Почему была написана эта книга? 1
Для кого написана эта книга? 1
Что такое "хорошая" программа? 2

Благодарности **4**

Глава 1. Три богатыря на рубеже столетий **5**

Глава 2. Работа с числовыми данными **13**

Внешнее и внутреннее представление числовых данных 13
Ввод числовой информации 16
Вывод числовых результатов 17
Задачи, советы и ответы 18

Глава 3. Обработка текстовой информации **117**

Символьные данные и их внутреннее представление 117
Ввод и вывод текстовой информации 119
Обработка фрагментов строк 122
Сравнение и сортировка текстовых данных 124
Управление цветом в текстовом режиме 126
Задачи, советы и ответы 127

Глава 4. Работа с массивами **169**

Объявление массивов 169
Инициализация массивов 171
Статические и динамические массивы 172

Массивы в качестве параметров процедур и функций 175
Сортировка больших массивов 181
Поиск 192
Задачи, советы и ответы 198

Глава 5. Рекурсивные функции и процедуры 253

Задачи, советы и ответы 254

Глава 6. Подпрограммы (процедуры) и функции 263

Оформление и вызов программных единиц в системе QBasic 264
Оформление и вызов программных единиц в системе Turbo C 267
Оформление и вызов программных единиц в системе Turbo Pascal 268
Оформление модулей на Паскале 269
Параметры подпрограмм, локальные и глобальные данные 271
Дерево решений 273

Глава 7. Работа с дисковыми файлами 285

Основные типы файлов в системе QBasic 286
Основные типы файлов в Паскале 292
Основные типы файлов в Си 298
Задачи, советы и ответы 303

Глава 8. Машинная графика 323

О мониторах и графических системах 323
О системах координат и текущей точке 324
О видеопамяти 324
Как формируется RGB-цвет пикселов 325
Краткий обзор графических возможностей систем программирования 326
Инициализация графического режима 327
Определение области графического вывода и выбор системы координат 329
Управление цветом 333
Работа с отдельными точками и растровыми изображениями 336
Отрезки прямых и прямоугольники 338
Окружности, эллипсы и дуги 342
Закрашивание и заполнение замкнутых областей 344
Заливка площадных фигур "прозрачными" шаблонами 357
Текстовые сообщения в графическом режиме 359
Задачи, советы и ответы 364

Глава 9. Работа с календарными датами 387

Немного истории 387
Вычисление юлианских дат 390
Задачи, советы и ответы 392

Глава 10. Использование системных функций 427

Управление мышью 436
Красивые окна в текстовом режиме 441

Приложение 1. Указатель программ 457

Приложение 2. Список литературы 463

Приложение 3. Описание дискеты 465

Предисловие

Почему была написана эта книга?

Сегодня грех жаловаться на недостаток литературы по компьютерной тематике. Однако полки в специализированных отделах книжных магазинов заполнены, в основном, многочисленными руководствами, обещающими в немыслимо короткие сроки обучить пользователя навыкам работы с наиболее популярными программными продуктами. На фоне этого довольно поверхностного изобилия не так часто встречаются хорошие книги, посвященные глубокому изучению алгоритмических языков и методам их использования для решения различных задач.

Столь же безрадостная участь постигла и серию книг по практике программирования. Во-первых, таких книг просто мало. Во-вторых, набор рассмотренных в них программ обычно относится к двум противоположным полюсам. Это либо тексты программ, занимающих 5—10 строк и демонстрирующих самые поверхностные возможности того или иного алгоритмического языка, либо задачи повышенной сложности, входившие в программы международных олимпиад по информатике, где наиболее важным моментом является выбор оптимального алгоритма, а вопросы конструирования, оформления и тестирования программ отходят на второй план или просто не рассматриваются. Наконец, нам кажется абсолютно непродуктивной идея многочисленных курсов по информатике, базирующихся на последовательном изучении двух-трех универсальных языков программирования. Такие алгоритмические языки имеют слишком много общего и просто нерационально отводить дефицитные учебные часы на изучение каждого из них с самого начала по стандартной схеме. Параллельное изучение эквивалентных конструкций в разных алгоритмических языках позволяет выработать навыки автоматического преобразования программ, что может оказаться полезным в практической работе.

Для кого написана эта книга?

Для тех, кто начинает изучение основ информатики и пытается освоить технику программирования на примерах сравнительно несложных задач. Для их понимания вполне достаточно элементов математики, изучаемых в средней школе.

Для тех, кто хочет научиться отличать хорошую программу от плохой и не собирается ограничивать свои познания техникой программирования. Последнее подразумевает знание допустимых типов данных и их внутреннего представления в памяти ЭВМ, умение использовать конструкции алгоритмического языка, функциональные возможности системы программирования и операционной системы.

Для тех, кто владеет основами программирования на одном из алгоритмических языков и хочет познакомиться со спецификой других достаточно распространенных алгоритмических языков. Современное общее образование вряд ли может считаться полноценным без изучения хотя бы одного иностранного языка. Точно так же профессиональное образование программиста не может ограничиваться рамками единственного алгоритмического языка. Мы постарались донести до читателей идею о том, что все достаточно универсальные алгоритмические языки очень похожи друг на друга и хорошее знание одного из них позволяет сравнительно просто разобраться с изобразительными средствами другого.

Для начинающих преподавателей информатики, которые хотели бы быстро пополнить арсенал своих учебных программ. Мы были бы крайне признательны вам за любые предложения по совершенствованию предлагаемых программ и расширение состава полезных задач.

Что такое "хорошая" программа?

Если отвлечься от конкретного содержания той или иной задачи, то основные этапы ее решения с помощью компьютера, преобразующего исходные данные в выходные, приведены в таблице ниже.

Номер этапа	Содержание	Исполнитель
1	Формулировка задачи	Человек
2	Выбор алгоритма	Человек
3	Составление исходной программы на алгоритмическом языке	Человек
4	Перевод исходной программы в коды машинных команд	Компьютер
5	Исполнение машинной программы	Компьютер

Эта схема достаточно условна. В ней скрыты довольно важные моменты, связанные с использованием готовых к употреблению библиотечных программ, с устранением синтаксических и алгоритмических ошибок в тексте исходной программы. В некоторых системах программирования (к ним, в частности, относится и QBasic) этапы 4 и 5 совмещены.

Каждому из этапов присущи свои особенности. Формулировка задачи должна исключать какую-либо неопределенность в задании исходных данных и устанавливать область их допустимых значений. Состав исходной информации должен быть достаточен для решения поставленной задачи. Так, например, нельзя построить треугольник, зная только два его параметра. Задание трех его углов тоже не позволяет найти однозначное решение. Алгоритм, как правило, должен приводить к решению задачи за конечное чис-

ло шагов или предусматривать прекращение бесконечных циклов при выполнении определенных условий. Программа на алгоритмическом языке должна иметь четко выраженную структуру и быть понятной не только ее автору. В случае необходимости она должна допускать участие человека в процессе принятия решений. Машинная программа должна располагать удобным интерфейсом и предоставлять в распоряжение пользователя интуитивно понятные средства ввода, управления вычислительным процессом, визуализации промежуточных и окончательных результатов. Это — далеко не полный перечень требований, выработанных многолетней практикой. Но остановимся чуть подробнее на деталях, связанных с организацией исходной программы на алгоритмическом языке.

Во-первых, программа должна выполнять свое главное функциональное назначение — правильно решать поставленную задачу при любом допустимом наборе исходных данных. Без этого любая программа теряет свой смысл.

Во-вторых, программа должна быть, по возможности, эффективной и не тратить на решение задачи лишнее время и ресурсы компьютера. Это особенно важно, когда предполагается многократное использование программы или ее включение в состав более сложного программного комплекса. Конечно, эффективность программы в первую очередь зависит от выбранного алгоритма. Но и реализация последнего может внести свою лепту. Программа может быть идеальной с точки зрения использования конструкций алгоритмического языка, но далеко не самой эффективной из-за неудачного алгоритма. Представим себе, что потребовалось сложить натуральные числа от k1 до k2. Такая процедура реализуется простым циклом, например, на Паскале:

```
s:=0;
for j:=k1 to k2 do s:=s+j;
```

Однако сумму членов арифметической прогрессии можно найти вспомнив или элементарно получив нужную формулу. Выпишем друг под другом ее элементы в прямом и обратном порядках:

k1	k1+1	k1+2	.....	k1+(k2−k1)
k1+(k2−k1)	k2−1	k2−2	.....	k1

Сумма каждой пары равна (k1+k2), и таких пар (k2−k1+1). Поэтому искомая сумма может быть найдена по формуле

```
s:=(k1+k2)*(k2-k1+1)/2;
```

И для ее нахождения потребуется всего пять операций, независимо от числа слагаемых. Из этого примера можно сделать довольно тривиальный вывод — знание техники программирования является необходимым, но не всегда достаточным фактором для эффективного решения задачи.

В-третьих, программа не должна быть очень замысловатой по своей реализации и не должна допускать модификацию или расширение возможностей другими программистами. Поэтому такие моменты, как простота и наличие полноценного комментария способствуют продлению жизненного цикла программы. Очень вредит простоте программы неумеренное использование оператора безусловного перехода. Приводимый ниже пример программы-

спагетти на языке Бейсик предназначен для поиска максимального среди значений трех переменных — max (a,b,c).

```
10 INPUT A,B,C
20 IF A>B THEN GOTO 80
30 IF B>C THEN GOTO 60
40 PRINT "Наибольшее число = "; C
50 GOTO 100
60 PRINT "Наибольшее число = "; B
70 GOTO 100
80 IF A<C THEN GOTO 40
90 PRINT "Наибольшее число = "; A
100 END
```

Представляете, как разрастется подобный монстр, если количество переменных, среди которых ищется максимум, достигнет 10. Насколько изящнее выглядит следующая программа:

```
10 INPUT A,B,C
20 MAX=A
30 IF MAX<B THEN MAX=B
40 IF MAX<C THEN MAX=C
50 PRINT "Наибольшее число = ";MAX
```

Эта программа значительно короче, и в ней нет ни одного оператора перехода. Кроме того, она допускает простое обобщение на поиск максимума среди элементов массива:

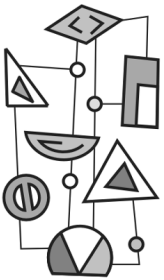
```
MAX=A(1)
FOR K=2 TO N
  IF MAX<A(K) THEN MAX=A(K)
NEXT K
```

Красивую программу можно сравнить со скульптурой, гениальный творец которой отсек от камня все лишнее.

Благодарности

Авторы выражают свою искреннюю признательность всем сотрудникам издательства "БХВ-Петербург", принимавшим участие в подготовке рукописи к печати, за оперативный выход книги в свет. Отдельно хотелось бы поблагодарить менеджера Адаменко Анатолия Николаевича, усилиями которого эта книга нашла своего читателя, и редактора Михайлову Ольгу Викторовну, которая помогла более точно донести до читателя идею книги, за ее внимательное, добросовестное отношение к работе. Большое спасибо.

Глава 1



Три богатыря на рубеже столетий

История алгоритмических языков, насчитывающая немногим менее полувека, успела зафиксировать создание и разрушение своеобразной Вавилонской башни, в стенах которой было погребено более 3000 оригинальных и не очень оригинальных версий универсальных и специализированных языков программирования.

Почти одновременно с появлением первых ЭВМ системные программисты стремились переложить на плечи ЭВМ наиболее рутинную работу, сопровождавшуюся многочисленными ошибками и описками. Первые элементы автоматизации процесса написания программ были связаны с заменой числовых кодов машинных операций их мнемоническими символьными обозначениями. Например, команда сложения содержимого двух ячеек памяти вместо сугубо числового кода `01 0100 0101 0102` превращалась в более осмысленное действие типа `ADD 0100,0101,0102`. Почти сразу же стало ясно, что использование естественной числовой нумерации ячеек памяти становится неразумной преградой между обозначениями переменных решаемой задачи и их эквивалентами в виде числовых адресов. Почему бы не возложить на специальную программу чисто механическую работу по замене символьных обозначений исходных и промежуточных данных задачи на их машинные адреса? И тогда очередной пункт алгоритма, выражавшийся простой формулой z = x + y, превращался в достаточно наглядную и близкую по смыслу команду `ADD X,Y,Z`. На первом этапе развитие этих идей сдерживало отсутствие устройств ввода/вывода, которые могли бы обрабатывать алфавитно-цифровую информацию. Как только аппаратные средства позволили преодолеть это препятствие, неотъемлемой частью программного обеспечения ЭВМ стали системы, получившие название Автокодов или Ассемблеров.

Следующий бастион, который был взят сторонниками автоматизации тяжелого ручного труда по составлению программ, — укрупнение и стандартизация операций, встречавшихся при решении различных задач. Операции, входившие в состав машинных команд, представляли собой слишком микроскопические действия, на которые приходилось разлагать более понятные человеку процедуры. Так появились алгоритмические языки, каждая строка которых могла быть автоматически трансформирована в цепочку эквивалентных машинных команд. И этот труд по сию пору с честью выполняют

многочисленные "переводчики" (трансляторы, компиляторы) и "исполнители" (интерпретаторы).

К числу первых алгоритмических языков, получивших достаточно широкое распространение, относятся Фортран (FORTRAN — от FORmula TRANslation, "трансляция формул") и Алгол (ALGOL — от ALGOrthmic Language, "алгоритмический язык"). Первый из них родился в недрах фирмы IBM в 1954 г. и активно поддерживался этим наиболее могущественным концерном по производству средств вычислительной техники. В нашей стране он стал широко известен в связи с переходом на выпуск ЭВМ Единой Системы (ЕС ЭВМ), программно совместимых с ранее появившимися моделями IBM/360, IBM/370. История развития Фортрана насчитывает довольно много версий и диалектов, среди которых наиболее известны Fortran-II, Fortran-IV, Fortran-77 и Fortran-90.

Алгол-60 появился как противовес Фортрану в результате объединенных усилий европейских ученых, представлявших, по большей части, академическую и вузовскую науку. Первые сведения о нем были опубликованы в 1958 г., но как стандарт языка он был утвержден в 1960 г. Алгол получил довольно широкое признание в нашей стране. Для него отечественными учеными были разработаны несколько систем программирования — ТА-1М (Вычислительный центр фирмы С. П. Королева), ТА-2М (Институт прикладной математики АН СССР), Альфа (Вычислительный центр СО АН СССР). Однако следующий стандарт языка Алгол-68 практически остался академическим проектом и большого распространения не получил, т. к. в срочном порядке был вытеснен Фортраном.

Несмотря на совершенно разную структуру и наличие несоизмеримых по финансовой мощи спонсоров, оба языка оказали существенное влияние на последующее развитие систем программирования. Так, например, язык Бейсик многое заимствовал из Фортрана, тогда как Паскаль продолжил линию алголоподобных языков. Несколько особняком стоит язык Си, впитавший в себя элементы как низкоуровневого программирования (наследие от языков типа Ассемблер), так и типы данных и синтаксические конструкции алгоритмических языков высокого уровня. Он не только объединил эти две кажущиеся противоположными компоненты, но и внес значительный вклад в развитие объектно-ориентированного программирования.

Далеко немногие алгоритмические языки выдержали испытание временем. Однако языки Бейсик, Си и Паскаль, включенные в нашу книгу, сродни трем былинным богатырям. Их возраст либо приближается к заветному сказочному рубежу в 33 года, либо уже превысил его, что мы и постарались отметить в названии первой главы. На сегодняшний день они являются наиболее популярными как у начинающих, так и у профессиональных программистов.

Самый почтенный среди них — Бейсик, днем рождения которого считается 1 мая 1964 г. В этот день в Дартмутском колледже (США) проявил первые

признаки жизни интерпретатор, созданный студенческим коллективом во главе с профессорами Джоном Кемени (J. G. Kemeny) и Томасом Куртцем (T. E. Kurtz). Своим названием BASIC обязан сокращению фразы Beginner's All-purpose Symbolic Instruction Code, которая дословно переводится как "многоцелевой код (язык) символических инструкций для начинающих".

Бейсик открыл эру диалогового программирования. До него культивировался пакетный режим, при котором бумажные или магнитные носители с программами сдавались дежурному оператору и упорядочивались в соответствии с приоритетами их владельцев. Составленный таким образом пакет программ поступал в ЭВМ на последовательную обработку. При этом достигалась максимальная загрузка оборудования, но каждая программа выполнялась либо до первой автоматически обнаруженной ошибки, либо до истечения лимита заказанного времени. Информация о результатах прохождения программ выдавалась их авторам 2—3 раза в сутки. Поэтому календарные сроки создания программ затягивались на многие месяцы.

Системы коллективного доступа, работавшие в диалоговом режиме, обеспечивали одновременное обслуживание нескольких пользователей, запускавших свои программы с электромеханических или электронных терминалов. Оперативно получив сообщение об очередной ошибке, пользователь имел возможность тут же исправить текст исходной программы и снова выполнить ее.

Диалоговый режим, естественно, был связан с дополнительными накладными расходами на многотерминальное обслуживание. Загрузка ЭВМ при этом снижалась, но оперативность в отладке программ приводила к существенному сокращению календарных сроков их разработки.

Бейсик был одним из первых алгоритмических языков, в составе которого изначально присутствовали операторы общения пользователя с пошагово выполняющейся программой. Одновременно с текстом сообщения об ошибке Бейсик-система сообщала номер строки программы, нарушившей синтаксис языка или приведшей к аварийной ситуации. Первые Бейсик-системы, совмещавшие в себе возможности ввода, редактирования, исполнения и отладки программ, послужили прототипами современных интегрированных сред.

Вторая особенность, привлекающая массового потребителя к Бейсику, кроется в простоте начального освоения и краткости его изобразительных средств. Попробуйте найти хотя бы еще один язык, на котором программа, отвечающая на вопрос, чему равно дважды два, состоит всего из четырех символов:

```
?2*2
```

Аналогичная программа на Паскале содержит, минимум, три строки, а ее длина превышает 20 символов:

```
begin
```

```
(writeln 2 *2);
end.
```

Примерно вдвое большего по числу символов требует ее аналог на Си:

```
#include <stdio.h>
main()
{ printf ("%d",2*2); }
```

Становлению и распространению Бейсика в нашей стране способствовали первые в мире пошаговые компиляторы с этого языка, созданные в 1969 г. для отечественных ЭВМ типа М-20 в Горьковском университете под руководством одного из авторов этой книги.

Второе поколение Бейсик-систем ведет свой отсчет от появления первых ПК на базе 8-разрядных микропроцессоров Intel-8080 и Z-80, для которых в середине 70-х годов был разработан компактный интерпретатор BASIC-80. Именно с него началась карьера самого молодого американского миллиардера Билла Гейтса, основавшего корпорацию Microsoft.

Появление 16-разрядных IBM-совместимых ПК ознаменовалось конкурентной борьбой между компаниями Borland International и Microsoft Corp. Первая из них выпустила на рынок удобную интегрированную среду с компилятором Turbo BASIC, которая быстро привлекла на свою сторону многочисленных любителей Бейсика. Однако более мощная компания, постоянный президент которой не упускает случая прибавить к своей профессии приставку "программист Бейсика", не могла смириться с таким положением. На смену тихоходному интерпретатору GW-BASIC пришла целая серия скоростных систем Quick BASIC, в составе которых наряду с интегрированной средой поставлялись автономные компиляторы и достаточно мощные библиотеки программ. Соревнование Бейсик-систем третьего поколения закончилось поражением фирмы Borland, прекратившей сопровождение своей разработки и передавшей права на Turbo BASIC одному из авторов, вышедшему из состава компании. Одна из последующих его разработок известна под названием Power Basic (Мощный Бейсик).

А фирма Microsoft совершенствовала свое любимое детище и выпустила в конце 80-х годов мощные системы для профессиональных разработок (Professional Development System) BASIC PDS-6 и PDS-7.

Авторы языка Кемени и Куртц спустя 25 лет попытались разработать новую версию под названием True Basic (Истинный Бейсик), однако из-за отсутствия серьезной поддержки со стороны крупных производителей программного обеспечения эта попытка, кроме издания книги и выпуска пробной версии системы, продолжения не получила.

Наконец, четвертое поколение Бейсика мы связываем с появлением в 1991 г. системы визуального программирования Visual Basic и ее последующим совершенствованием сотрудниками Microsoft. На момент выпуска нашей кни-

ги в эксплуатации находится версия VB-6.0. До версии 3.0 включительно Visual Basic был построен по схеме интерпретатора, однако последние его версии позволяют получать готовые к исполнению модули. Кроме полноценной системы программирования, фирма Microsoft с 1993 г. выпускает версии VBA (Visual Basic for Application — система для разработки программируемых процедур в пакете MS Office) и VBScript (ограниченная версия для разработки Web-приложений).

Алгоритмический язык Паскаль был придуман в 1968 г. профессором Института информатики при Швейцарской высшей технической школе Никлаусом Виртом. В 1970 г. под его руководством был разработан первый компилятор, а в следующем году появилась и первая публикация. Свое название язык получил в честь известного французского математика Блеза Паскаля, который в 19-летнем возрасте изобрел первую суммирующую машину.

Новый язык, являясь продолжателем традиций алгоритмического языка Алгол-60, был ориентирован, главным образом, на обучение курсу систематического программирования. В Паскале были представлены наиболее распространенные типы данных и средства для создания пользовательских структур. К числу элементов, способствующих созданию надежных программ, относились блочные конструкции и требование обязательного описания (объявления) всех объектов — типов данных, констант, переменных, меток, функций и процедур.

Заметную роль в разработке стандарта языка Паскаль и совершенствовании его средств ввода/вывода сыграла рабочая группа Британского института стандартов во главе с А. Эддиманом. Британский стандарт был принят в 1982 г., а несколько позднее его утвердила международная организация ISO. Однако к этому времени Н. Вирт, недовольный предложениями рабочей группы, отказался от сотрудничества по совершенствованию Паскаля и переключился на новый проект Модула.

Паскаль довольно долго оставался средством для изучения программирования в университетах, т. к. ни одна серьезная компьютерная фирма его не поддерживала. Перелом в отношении к этому языку наметился в 1984 г., когда молодой француз Филипп Кан привез в США необычайно скоростной компилятор Turbo Pascal для IBM-совместимых ПК и начал торговать им по смехотворно низкой цене 49 долларов 95 центов (для сравнения напомним, что первые версии Бейсик-интепретатора распространялись по цене порядка 500 долларов). Удачная реклама и бросовая цена позволили Кану продать за первый месяц более 3000 копий системы и заложить основы фирмы Borland International. Последующие восемь лет Turbo Pascal оставался наиболее опекаемым продуктом фирмы, которая сумела выпустить девять различных версий. Самая последняя из них (7.0, 1992 г.) включает две системы — Turbo Pascal, функционирующую под управлением MS-DOS, и расширенную версию Borland Pascal, работающую в среде Windows. Несмотря на то, что в январе 1995 г. Ф. Кан покинул пост президента и ушел из ком-

пании, фирма Borland на базе языка Object Pascal выпустила одну из наиболее популярных сред визуального программирования — Delphi и продолжает поддерживать ее версии в современных операционных системах. На сегодня наибольшей популярностью пользуется версия Delphi 5.0.

Язык Си был придуман в 1972 г. сотрудником Bell Laboratories (отделение известной телефонной компании AT&T) Деннисом Ритчи, одним из первых пользователей операционной системы Unix. Задумывался он не как универсальный алгоритмический язык, а, скорее, как инструмент для развития операционной системы и создания новых обслуживающих программ (утилит). Такой подход характерен для большинства системных программистов, разрабатывающих сложные проекты и придумывающих для облегчения своего труда различные сервисные процедуры, макрокоманды и т. п. По завершении разработки, как правило, эти инструментальные наборы предаются забвению или, в лучшем случае, остаются в личных архивах авторов. Язык Си эта участь миновала. Вполне возможно, что его становлению способствовало последующее всемирное признание операционной системы Unix.

Как алгоритмический язык сравнительно низкого уровня, т. е. достаточно близкий к Ассемблеру, Си имел предшественников в лице аналогичных инструментальных средств — языков CPL, BCPL (Basic Combined Programming Language — базовый комбинированный язык программирования) и B. Два первых разрабатывались в конце 70-х годов в Кембриджском университете в качестве машинно-независимых языков для создания трансляторов. Последний был придуман Кеном Томпсоном — сотрудником Bell Laboratories и автором операционной системы Unix. В отличие от своих предшественников Д. Ритчи наряду с машинно-ориентированными типами данных (байт, слово) ввел в состав Си объекты и операторы, присущие универсальным языкам (числовые и символьные переменные, структурные блоки), сохранив при этом элементы, характерные для макроассемблера MACRO-11 (логические операции над битами, сдвиги, работа с адресами и регистрами).

Первым программным продуктом, написанным почти полностью на Си, был компилятор с языка Си в код машинных команд компьютера PDP-11/20 (прототип мини-ЭВМ СМ-4). В 1973 г. Д. Ритчи и К. Томпсон переписали на Си большую часть операционной системы Unix. Из 13 000 машинных команд для PDP-7, на которой появилась первая версия Unix, только 800 пришлось вручную перевести в ассемблер PDP-11. В процессе перевода Unix из однопользовательской операционной системы, ориентированной на работу в конкретной ЭВМ, превратилась в мобильную операционную систему коллективного пользования. Успех этой операции в значительной мере предопределил популярность новой операционной системы и ее базового инструмента — языка Си. В 1976 г. Д. Ритчи и К. Томпсон перенесли Unix с ЭВМ фирмы DEC на компьютеры другой архитектуры (Interdata 8/32), практически ничего не изменив в ядре операционной системы,

написанном на Си. Точно таким же образом система Unix распространялась на десятки машин различных типов.

В 1978 г. появилась первая книга, посвященная описанию Си и технике программирования на этом языке, которая с большим запозданием была переведена на русский язык (Б. Керниган, Д. Ритчи, А. Фьюэр. "Язык программирования Си. Задачи на языке Си". — М.: Финансы и статистика, 1985). От фамилий двух первых авторов произошло сокращенное обозначение первого, никем не утверждавшегося, но принятого всеми программистами стандарта языка Си — K&R.

Дальнейшая работа по совершенствованию языка Си и принятию в 1987 г. первого настоящего стандарта ANSI C была выполнена на общественных началах рабочей группой при Американском национальном институте стандартов. Возглавлял эту работу Лэрри Рослер — сотрудник Bell Labs. Наиболее серьезный вклад в развитие языка Си за последние годы внес еще один представитель той же лаборатории Бьерн Страуструп, который ввел в обращение новые объекты — классы, объединяющие данные, и обрабатывающие их функции. С 1983 г. за расширенной версией языка Си с классами закрепилось название C++.

Первые версии Си подвергались серьезной критике за отсутствие достаточной строгости, приводившей к многочисленным ошибкам из-за работы с неинициализированными переменными, отсутствия контроля за выходом индексов у элементов массивов из установленных пределов, несоответствия типов формальных и фактических параметров функций и т. п. Перед системными программистами Bell Labs эти проблемы остро не стояли, т. к. они пользовались специальной программой Lint, которая проводила тщательный анализ программ, написанных на Си, перед их трансляцией и выполнением.

Для рядовых пользователей ситуация изменилась с появлением интегрированных сред, из которых наибольшую популярность приобрели Турбо-системы фирмы Borland. Первая версия Turbo C, работавшая в среде MS-DOS, была выпущена в 1987 г. Совсем недавно фирма Borland выпустила на рынок версию 5.0, предназначенную для работы под управлением Windows. На базе этого компилятора компания Borland (новое название — Inprise Corp.) разработала серию систем визуального программирования — Borland C++ Builder.

Известны и другие реализации языка Си на IBM-совместимых ПК — Microsoft C, Lattice C, Zortech C, Symantec C. В нашей стране продукция фирмы Borland получила наибольшее распространение, хотя за последнее время намечается тенденция к более широкому использованию среды Visual C++, разработанной фирмой Microsoft. В основном, это диктуется требованиями зарубежных фирм, размещающих свои заказы у отечественных производителей программных продуктов.

Современные версии трех наиболее устоявшихся алгоритмических языков обладают примерно равными функциональными возможностями. Подтверждению именно этого фундаментального свойства универсальных систем программирования посвящена наша книга. Завершая краткий исторический экскурс, остановимся еще раз на мотивах, объясняющих наш выбор языков и систем программирования.

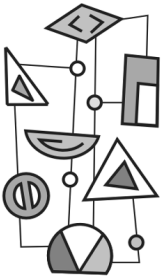
Бейсик — один из наиболее распространенных в мире алгоритмических языков, используемых, как правило, непрофессиональными программистами. По данным [3] из пяти миллионов программирующих пользователей ЭВМ Бейсиком пользуется не менее двух миллионов человек. Большая часть учащихся средних школ начинает освоение информатики с изучения Бейсика, т. к. он доступен на самой допотопной вычислительной технике, которой, к сожалению, еще до сих пор укомплектовано большинство школ России. Несмотря на то, что Бейсик является старожилом по сравнению с Си и Паскалем, в последние годы он переживает вторую молодость в связи с появлением модных систем визуального программирования и массовым использованием Microsoft Office.

Паскаль — алгоритмический язык, созданный специально для изучения программирования и широко используемый в высших учебных заведениях. Вслед за Бейсиком он тоже вступил на тропу визуального программирования, и система Delphi считается одним из лучших инструментальных средств для создания приложений, функционирующих под управлением Windows 95/98/NT.

Си — язык профессионалов, на котором сегодня написано большинство программных систем. Большая часть высших учебных заведений, особенно технического профиля, использует Си в качестве основного языка преподавания программирования. Такие среды визуального программирования, как Borland C++ Builder и Microsoft Visual C++ являются несомненными лидерами в разработках современного программного обеспечения.

В связи с тем, что книга ориентирована на начальное знакомство с перечисленными алгоритмическими языками, в качестве сред программирования нами выбраны самые непритязательные по требованиям к оборудованию системы — QBasic, Turbo C 2.0 или Borland C++ 3.1 и Turbo Pascal 7.0. Все они могут функционировать на самых примитивных IBM-совместимых ПК с оперативной памятью от 1 Мб, занимая на винчестере от 1 до 25 Мб. Кроме того, система QBasic является составной частью операционной системы MS-DOS и представлена там всего двумя небольшими файлами — интерпретатором qbasic.exe и файлом помощи qbasic.hlp.

Глава 2



Работа с числовыми данными

Современные представления о числовых данных базируются на так называемых позиционных системах счисления. Для этих систем характерно основание системы p, степень которого определяет вес цифры a (a = 0, 1, 2, ..., p-1) в зависимости от занимаемой позиции:

$a_k a_{k-1} \ldots a_1 a_0,\ b_{-1} b_{-2} \ldots b_{-m} = a_k * p^k + a_{k-1} * p^{k-1} + \ldots + a_1 * p^1 + a_0 * p^0 + b_{-1} * p^{-1} +$
$+ b_{-2} * p^{-2} + \ldots + b_{-m} * p^{-m}$

Главенствующую роль в человеческом общении играет десятичная система счисления (p = 10; a = 0, 1, ..., 9). Однако ЭВМ используют более рациональную двоичную систему (p = 2; a = 0,1). Единственным ее недостатком является большая длина чисел. Количество разрядов (цифр) в двоичном представлении числа примерно в три раза превышает количество десятичных цифр. Для преодоления этого неудобства программисты прибегают к более компактной записи двоичных кодов в виде восьмеричных или шестнадцатеричных чисел. При этом три или четыре смежные двоичные цифры заменяют одной восьмеричной или шестнадцатеричной цифрой. Например:

$192_{10} = 11000000_2$,

$11000000_2 = 300_8$,

$11000000_2 = C0_{16}$.

Внешнее и внутреннее представление числовых данных

Под внешним представлением числовой информации подразумеваются способы записи данных, используемые в текстах программ, при наборе чисел, вводимых в ЭВМ по запросу программы, при отображении результатов на экране дисплея или на принтере. Кроме естественного представления числовых констант в виде целого или вещественного числа, языки программирования допускают различные добавки в начале ("префиксы") или конце ("суффиксы") числа, определяющие способы преобразования и хранения данных в памяти компьютера.

Во входном языке системы QBasic такого рода добавки представлены одним из символов `%`, `!`, `&` или `#`, приписываемым вслед за числом, и одной из двухсимвольных комбинаций `&B`, `&O` или `&H`, располагаемой перед числом:

- `5%` — целое число;
- `5&` — целое число с удвоенной точностью;
- `5` или `5!` — вещественное число;
- `5#` — вещественное число с удвоенной точностью;
- `&B0011100100110111` — двоичное число;
- `&O34467` — восьмеричное число;
- `&H3937` — шестнадцатеричное число.

В Си к аналогичным суффиксам относятся указания об удвоенной длине целых чисел (буквы `L` или `l`), указания о вещественном формате числа, не содержащего в своей записи десятичной точки или десятичного порядка (буква `F` или `f`), указания об использовании беззнакового представления целых чисел (буква `U` или `u`). Префиксы в Си используются для записи восьмеричных (число начинается с `0`) или шестнадцатеричных (числу предшествует одна из комбинаций `0x` или `0X`) констант:

- `5` — короткое целое число со знаком;
- `5U` — короткое целое число без знака;
- `5L` — длинное целое число со знаком;
- `5LU` или `5UL` — длинное целое число без знака;
- `05` — восьмеричное число;
- `0x5` или `0X5` — шестнадцатеричное число;
- `5f` или `5F` — вещественное число со знаком.

В Паскале используется единственный префикс — символ `$`, предшествующий шестнадцатеричному числу:

`$0A`, `$F0A5`, `$FF00140D`.

Наличие в естественной записи числа точки (`3.1415`) или указателя десятичного порядка (`314.159265e-02`) означает, что соответствующее значение представлено в ЭВМ в виде вещественного числа с плавающей запятой.

Машинные форматы представления чисел мы будем называть внутренними, и из приведенных ранее примеров следует, что машинные числа бывают целыми и вещественными. В свою очередь, каждый из этих типов данных допускает несколько представлений, отличающихся диапазоном допустимых чисел. Остановимся более подробно на машинных форматах числовых данных, соответствующих их аналогам в алгоритмических языках.

Самые короткие числа со знаком представлены в памяти ЭВМ одним байтом, в котором может разместиться любое число из диапазона от —128 до

+127. В Си для описания данных такого типа используется спецификатор `char`, а в Паскале — `shortint`.

В одном же байте может быть расположено и самое короткое целое число без знака. В Си для описания таких данных служит спецификатор `unsigned char`, в Паскале — `byte`. Диапазон допустимых данных при этом смещается вправо и равен [0, 255]. QBasic не располагает языковыми средствами для работы с однобайтовыми целыми числами.

Вторая категория целых чисел представлена двухбайтовыми данными. В варианте со знаком они предлагают диапазон от —32 768 до +32 767, в варианте без знака — от 0 до 65 535.

QBasic обеспечивает работу с двухбайтовыми числами со знаком для переменных, описанных как `AS INTEGER` или устаревшее `DEFINT`, и переменных, чьи имена заканчиваются символом `%`. Двухбайтовые целые без знака в QBasic записываются в виде двоичных, восьмеричных или шестнадцатеричных констант. Однако арифметические операции над такими данными иногда выполняются некорректно.

Си использует для описания двухбайтовых целочисленных данных спецификаторы `int` и `unsigned int`. В Паскале для этой же цели служат спецификаторы `integer` и `word`. Оба языка корректно выполняют арифметические операции и с беззнаковыми данными при условии, что результат не выходит за пределы разрешенного диапазона.

Третья категория целых чисел в IBM PC представлена четырехбайтовыми данными. В варианте со знаком они перекрывают диапазон от —2 147 483 648 до +2 147 483 647, в варианте без знака — от 0 до 4 294 967 295.

В QBasic допустимы только данные со знаком и имена переменных такого типа описываются как `AS LONG` или `DEFLNG`. К ним же относятся и переменные, чьи имена заканчиваются символом `&`.

Для описания четырехбайтовых данных целого типа в Си используются спецификаторы `long` (эквивалент `long int`) и `unsigned long`. В Паскале работа с длинными целыми без знака не предусмотрена. Там можно оперировать с четырехбайтовыми данными только типа `longint`.

Следует помнить, что для хранения любых целых чисел со знаком в IBM PC используется дополнительный код, что сказывается на представлении отрицательных чисел:

```
+5          0 0000101          0 000000000000101
-5          1 1111011          1 111111111111011
```

Наиболее часто применяемые типы вещественных чисел представлены короткими (4 байта) и длинными (8 байт) данными.

В QBasic им соответствуют описания `AS SINGLE` (устаревшее — `DEFSNG`) и `AS DOUBLE` (устаревшее — `DEFDBL`). Си использует для этой же цели спецификаторы `float` и `double`, Паскаль — `single` и `double`.

Короткий вещественный формат по модулю обеспечивает представление чисел в диапазоне от 10^{-38} до 10^{+38} примерно с 7—8 значащими цифрами. Для 8-байтового формата диапазон существенно расширяется — от 10^{-308} до 10^{+308}, а количество значащих цифр увеличивается до 15—16.

Сопроцессор IBM PC предлагает еще два формата данных, занимающих соответственно 8 и 10 байт. Первый из них допускает работу с целыми числами из диапазона от -2^{63} до 2^{63-1}. Второй формат перекрывает диапазон данных (по модулю) от 10^{-4932} до 10^{+4932}, сохраняя 19—20 значащих цифр. Расширенный формат вещественных данных можно использовать в программах на Си (`long double`) и на Паскале (`extended`). А формат сверхдлинных целых чисел используется только в Паскале, но там они почему-то отнесены к вещественным данным типа `comp`.

В Паскале по традиции сохранился еще один тип вещественных данных — `real`, занимающий в памяти 6 байт. Его диапазон совпадает с типом `single`, однако количество значащих цифр несколько больше — 10—12. Формат `real` не поддерживается аппаратными средствами IBM PC, поэтому операции над такими данными выполняются с помощью подпрограмм, что существенно увеличивает время решения задачи.

В машинном представлении вещественных данных разного типа на IBM PC не выдержана какая-то общая идеология. Объясняется это, по всей вероятности, разными наслоениями на прежние аппаратные решения, которые принимались при разработке процессоров в разных отделениях фирмы Intel. Поэтому здесь имеют место такие нюансы, как сохранение или не сохранение старшего бита мантиссы, представление мантиссы в виде чисто дробного ($0{,}5 \le m < 1$) или смешанного ($1 \le m < 2$) числа и т. п. Прикладных программистов эти детали мало интересуют, однако при создании специальных системных компонент с точным представлением данных приходится считаться.

Ввод числовой информации

Каких-либо особых проблем с вводом числовой информации в программах не возникает. Но на некоторые детали в том или ином алгоритмическом языке надо обратить внимание.

В QBasic можно натолкнуться на неприятности при вводе нескольких числовых значений в одном операторе. Например:

```
INPUT A,B,C
```

Числовые данные, набираемые пользователем, должны быть разделены пробелом. Если в наборе второго или третьего значения вы допустите ошибку,

то последует сообщение `Redo from start`, которое заставит вас повторить ввод всех трех чисел с самого начала. Некоторые программисты даже предпочитают вводить одним оператором только одно значение. Несоответствие между типом переменной и вводимым значением приводит к ошибке при попытке ввести вещественное значение в целочисленную переменную или при наборе недопустимого символа в числе.

В Си основные неприятности форматного ввода (функция `scanf`) связаны с попыткой указать в списке ввода не адрес переменной, а ее имя:

```
scanf("%d",x);  //правильно было бы scanf("%d",&x);
```

Компилятор TC/BC такую ошибку, к сожалению, не замечает и преобразует имя `x` в какой-то фантастический адрес. Последующую работу программы в этом случае предсказать трудно. Не обращает внимания компилятор Си и на несоответствие между спецификатором формата и типом переменной из списка ввода. Всего этого можно избежать, используя потоковый ввод:

```
cin >> x;
```

Самый тщательный контроль за соответствием между типами вводимых значений и типами соответствующих переменных в списке параметров процедур `read` и `readln` обеспечивает Паскаль. Однако здесь при попытке ввести непредусмотренный символ в числе вслед за сообщением об ошибке задача снимается, если программист не смог предвидеть заранее обход системной реакции на особые ситуации.

Вывод числовых результатов

Наиболее приятный вид имеет числовая информация, организованная по табличному типу — в виде колонок фиксированной ширины, в которых одноименные числовые разряды располагаются друг под другом (единицы — под единицами, десятки — под десятками, сотни — под сотнями и т. д.). При этом, в частности, более рационально используется площадь экрана, что достигается за счет управления форматами выводимых данных.

В QBasic для этой цели служит оператор `PRINT USING`, совмещающий в себе и строку с описанием формата и список выводимых значений:

```
PRINT USING "A=##.### B=### C=#.####^^^^"; A,B,C
```

Аналогичные средства имеются и в Си:

```
printf("A=%6.3f B=%3d C=%10.4e",A,B,C);
```

При выводе в поток, когда к Си-программе подключаются заголовочные файлы `iostream.h` и `iomanip.h`, тоже существует возможность управлять форматом выводимых данных:

```
cout << "A=" << setw(6) << setprecision(5) << A;
```

Несколько скромнее выглядит управление форматом в Паскале:

```
write('A=',A:6:3,"B=',B:3,'C=',C:10);
```

Среди форматных спецификаторов в Си есть дополнительные возможности, отсутствующие в QBasic и Паскале. Например:

```
printf("%4x %6o",x,y);
```

Приведенные здесь спецификаторы позволяют вывести значения целочисленных переменных `x` и `y` соответственно в виде шестнадцатеричного числа с четырьмя цифрами и восьмеричного числа с шестью цифрами.

В Си можно прижимать выводимое число к левой границе отведенного поля (по умолчанию действует правый прижим), печатать знак "+" у положительных чисел, подавлять или разрешать вывод незначащих нулей и др. Подробную информацию о структуре форматного описателя можно найти в файлах помощи.

Задачи, советы и ответы

Задание 2.01. Ввод и вывод целочисленных данных

Ввести 20 целых чисел. Вывести их компактно (в одну или несколько строк), размещая перед каждым числом его порядковый номер. После нажатия какой-либо клавиши вывести числа столбиком, располагая одноименные разряды друг под другом, подвести под столбиком черту и напечатать сумму введенных чисел.

Совет 1

Предположим, что вы будете работать с двухбайтовыми целыми числами. Тогда для размещения самого длинного числа потребуется не менее 7 позиций (число —32 768 занимает 6 позиций плюс пробел между числами). Если перед числом необходимо вывести его номер, то дополнительно потребуются еще, как минимум, 3 позиции (2 под номер из диапазона [1,20] и разделитель между номером и числом, например в виде двоеточия). Таким образом, для компактного вывода на каждое число вместе с его номером можно отвести 10 позиций, что хорошо согласуется с длиной строки на дисплее. Если бы оказалось, что длина строки не кратна ширине колонки, то часть последнего числа была бы автоматически перенесена на следующую строку.

Совет 2

Как организовать ожидание в программе до нажатия какой-либо клавиши? В QBasic для этой цели подойдет системная переменная `INKEY$`. Достаточно присвоить ее значение какой-либо символьной переменной, например `A$`, и организовать бесконечный цикл до тех пор, пока длина значения `A$` перестанет отличаться от нуля:

```
A$=""
M10 : A$=INKEY$ : IF A$="" THEN GOTO M10
```

Можно воспользоваться и другим приемом — включить в программу оператор ввода в какую-либо переменную символьного типа. Такая переменная предпочтительнее числовой, т. к. в нее можно ввести пустое значение, нажав только клавишу <Enter>. Кроме того, набор любого отображаемого символа не приведет к ошибке.

В Си временный приостанов до нажатия какой-либо клавиши организуют с помощью функции `getch`.

В Паскале можно организовать бесконечный цикл, аналогичный приведенному выше варианту для QBasic, с помощью логической функции `KeyPressed`:

```
while not KeyPressed;
```

Аналогом ввода пустого значения `INPUT A$` в QBasic в Паскале является использование процедуры `read/readln` без параметров. В этом случае для проталкивания программы потребуется нажать клавишу <Enter>.

Программа 2_01.bas

```
REM Ввод, вывод и суммирование 20 целых чисел
DIM A(20)
F$="##:###### "
PRINT "Введите 20 чисел, по одному в строке"
FOR I=0 TO 19: INPUT A(I): S=S+A(I): NEXT I
FOR I=0 TO 19: PRINT USING F$;I+1;A(I); : NEXT I
INPUT A$
FOR I=0 TO 19: PRINT USING F$;I+1;A(I): NEXT I
PRINT "----------"
PRINT USING "   ######"; S
END
```

Программа 2_01.c

```
/* Ввод, вывод и суммирование 20 целых чисел */
#include <stdio.h>
#include <conio.h>
  main()
{
  int j,s=0,a[20];
  clrscr();
  printf("Введите 20 чисел, по одному в строке\n");
  for(j=0; j<20; j++)
    { scanf("%d",&a[j]); s+=a[j];  }
```

```
  for(j=0;j<20; j++)
    printf("%2d:%-6d ",j+1,a[j]);
  for(j=0;j<20; j++)
    printf("\n%2d:%6d ",j+1,a[j]);
  printf("\n   ------\n%9d",s);
  getch();
}
```

Программа 2_01.pas

```
program io_numbers;
{ Ввод, вывод и суммирование 20 целых чисел }
uses Crt;
var
  j,s:integer;
  a:array [1..20] of integer;
begin
  s:=0;
  clrscr;
  writeln('Введите 20 чисел, по одному в строке');
  for j:=1 to 20 do
    begin readln(a[j]);  s:=s+a[j]; end;
  for j:=1 to 20 do write(j:2,':',a[j]:6,' ');
  writeln('Нажмите любую клавишу';
  readkey; clrscr;
  for j:=1 to 20 do writeln(j:2,':',a[j]:6,' ');
  writeln('   ------');
  writeln(s:9);
  readln;
end.
```

Задание 2.02. Ввод и вывод вещественных данных

Самостоятельно выполнить задание 2.01 в предположении, что вводимые числа вещественные и имеют две значащие цифры в дробной части.

Задание 2.03. Преобразование десятичного числа в системы с основанием 2, 8, 16

Ввести длинное неотрицательное число. Вывести его в двоичном, восьмеричном и шестнадцатеричном представлении.

Совет 1 (QBasic)

Предлагается воспользоваться стандартными функциями `OCT$` и `HEX$`, преобразующими числовой аргумент в строку с его представлением в соответствующей системе счисления. Для двоичного представления можно распечатать восьмеричные цифры их трехразрядными двоичными эквивалентами.

Совет 2 (Си)

Предлагается воспользоваться библиотечной функцией `ltoa`, преобразующей свой первый аргумент из машинного представления числа в строку, заданную вторым аргументом. Третий аргумент этой функции определяет основание системы счисления, в которую конвертируется исходное число. Вообще говоря, восьмеричный и шестнадцатеричный эквиваленты числа проще вывести, используя спецификатор формата `%o` и `%x`. Для вывода двоичного представления числа можно было бы организовать цикл со сдвигом числа на один разряд влево (`N = N << 1;`), пробой старшей цифры (`N & 0x8000`) и выводом соответствующего символа в зависимости от результата сравнения.

Совет 3 (Паскаль)

В этом языке отсутствуют какие-либо системные функции или процедуры, отмеченные выше. Поэтому единственным средством будет лобовое преобразование путем последовательного деления исходного числа на основание соответствующей системы и запоминание получающихся остатков в некотором массиве. Так как длинное число занимает в памяти ЭВМ 4 байта, максимальный размер массива для хранения цифр числа в соответствующем представлении не должен превышать 32 элемента. Небольшие проблемы могут возникнуть при выводе шестнадцатеричных цифр от `A` до `F`. Из них придется вычитать 10 и добавлять полученную разницу к коду буквы `A`.

Программа 2_03.bas

```
REM Перевод числа в системы с основаниями 2, 8 и 16
CLS
INPUT "Введите положительное число : ",N&
A$=OCT$(N&)
PRINT "В двоичном представлении ";N&;"= ";
FOR k=1 TO LEN(A$)
  B$=MID$(A$,k,1) : ' Выделение очередной восьмеричной цифры
  SELECT CASE B$
    CASE "0": IF k=1 THEN PRINT "";  ELSE PRINT "000";
    CASE "1": IF k=1 THEN PRINT "1";  ELSE PRINT "001";
    CASE "2": IF k=1 THEN PRINT "10";  ELSE PRINT "010";
    CASE "3": IF k=1 THEN PRINT "11";  ELSE PRINT "011";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
```

```
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
  END SELECT
NEXT k
PRINT
PRINT "В восьмеричном представлении ";N&;"= ";OCT$(N&)
PRINT "В шестнадцатеричном представлении ";N&;"=";HEX$(N&)
END
```

Программа 2_03a.bas

```
REM Перевод числа в системы с основаниями 2, 8 и 16
CLS
INPUT "Введите положительное число : ",n&
a$=OCT$(n&) : ' Перевод в восьмеричную систему
IF n&=0 THEN
  PRINT "Это число в любой системе равно 0"
  STOP
END IF
PRINT "В двоичном представлении ";n&;"= ";
  B$=LEFT$(a$,1): : ' Выделение очередной восьмеричной цифры
  SELECT CASE B$
    CASE "0": PRINT "";
    CASE "1": PRINT "1";
    CASE "2": PRINT "10";
    CASE "3": PRINT "11";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
END SELECT
FOR K=2 TO LEN(a$)
  B$=MID$(a$,K,1)
  SELECT CASE B$
    CASE "0": PRINT "000";
    CASE "1": PRINT "001";
    CASE "2": PRINT "010";
    CASE "3": PRINT "011";
    CASE "4": PRINT "100";
    CASE "5": PRINT "101";
```

```
    CASE "6": PRINT "111";
    CASE "7": PRINT "111";
  END SELECT
NEXT K
PRINT
PRINT "В восьмеричном представлении ";n&;"= ";OCT$(n&)
PRINT "В шестнадцатеричном представлении ";n&;"= ";HEX$(n&)
END
```

Программа 2_03.c

```
/* Перевод числа в системы счисления с основаниями 2, 8 и 16 */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
   main()
{
  long N;
  char a[33];
  clrscr();
  printf("Введите положительное число : ");
  scanf("%ld",&N);
  if(N==0)
    {
       printf("\nТакое число в любой системе = 0");
       exit(1);
    }
  ltoa(N,a,2);  /* перевод в двоичную систему */
  printf("\nВ двоичном представлении %ld = %s",N,a);
  ltoa(N,a,8);  /* перевод в восьмеричную систему */
  printf("\nВ восьмеричном представлении %ld = %s",N,a);
  ltoa(N,a,16); /* перевод в шестнадцатеричную систему */
  printf("\nВ шестнадцатеричном представлении %ld = %s",N,a);
  getch();
}
```

Программа 2_03.pas

```
program _2_8_16;
{ Перевод числа в системы с основаниями 2, 8 и 16 }
```

```
uses crt;
var
  N1,N:longint;
  a:array [0..31] of byte;
  j,k:byte;
  s:char;
begin
  clrscr;
  write('Введите положительное число : ');
  readln(N);
  if N=0 then
     begin
       writeln('Такое число в любой системе = 0');
       exit;
     end;
  N1:=N;
  for j:=0 to 31 do a[j]:=0;
  while N1<>0 do
    begin
      a[j]:=N1 mod 2; {цикл выделения двоичных цифр}
      dec(j);
      N1:=N1 div 2;
    end;
  write('В двоичном представлении ',N,'=');
  for k:=j+1 to 31 do write(a[k]:1);
  writeln;
  N1:=N;
  for j:=0 to 10 do a[j]:=0;
  while N1<>0 do
    begin
      a[j]:=N1 mod 8; {цикл выделения восьмеричных цифр}
      dec(j);
      N1:=N1 div 8;
    end;
  write('В восьмеричном представлении ',N,'=');
  for k:=j+1 to 10 do write(a[k]:1);
  writeln;
  N1:=N;
  for j:=0 to 7 do a[j]:=0;
```

```
  while N1<>0 do
    begin
      a[j]:=N1 mod 16;
      dec(j);
      N1:=N1 div 16; {цикл выделения шестнадцатеричных цифр}
    end;
  write('В шестнадцатеричном представлении ',N,'=');
  for k:=j+1 to 7 do
    begin
      if a[k]<10 then s:=chr(ord('0')+a[k])
      else s:=chr(ord('A')+a[k]-10);
      write(s);
    end;
  readln;
end.
```

Задание 2.04. Преобразование десятичного числа в систему с основанием `r`

Составить функцию `num_to_str(num, r)`, возвращающую в качестве своего значения строку с представлением натурального числа `num` в системе счисления с основанием `r`. Предполагается, что число `num` в памяти компьютера представлено 4-байтовым целым, а основание `r` принадлежит диапазону [2, 16]. Для обозначения цифр, превосходящих 9, рекомендуется воспользоваться латинскими буквами `A`, `B`, ... , `F`.

Совет 1 (общий)

Так как основание системы может быть любым, необязательно совпадающим с наиболее распространенными вариантами (`r = 2,8,16`), то наиболее универсальным способом перевода остается последовательное деление исходного числа на основание с запоминанием целочисленных частных и остатков. Небольшое осложнение будет вызвано преобразованием двухзначных остатков (при `r > 10`) в односимвольную "цифру". Однако подход к решению этого вопроса уже был продемонстрирован в предыдущей программе на Паскале. Очередной остаток `q` надо сравнить с 10 и, в зависимости от исхода сравнения, либо преобразовывать в символ полученную цифру, либо добавлять разницу `q-10` к коду буквы `A`. Вторая тонкость заключается в том, что предлагаемый способ перевода позволяет получать цифры разложения справа налево, начиная с младшей. Поэтому перед выдачей результата полученную строку придется перевернуть.

Совет 2 (QBasic)

Вы можете выбрать один из двух способов описания типов данных — более современный, хотя и несколько более длинный, с использованием переменных,

описанных как `AS LONG`, `AS STRING`, `AS INTEGER`, или более старомодный с добавками `%`, `&`, `$` в конце имени переменной. Возможность "складывать" символьные значения в любом порядке позволяет избежать инвертирования результирующей строки. Для этого вместо оператора `A$ = A$ + CIF$` удобнее воспользоваться левосторонней конкатенацией `A$ = CIF$+A$`.

Совет 3 (Си)

Выбор длины строки, в которой будет накапливаться промежуточный или окончательный результат, определяется самым длинным разложением при `r = 2` (т. е. 32 разряда плюс еще один байт под признак конца строки). Заводить две строки под правое и левое представление, наверное, не стоит. Проще произвести перестановку симметричных байтов (первый с последним, второй с предпоследним и т. д.) в одной строке. Для выполнения этой операции придется ввести счетчик количества цифр или прибегнуть к библиотечной функции `strlen` из раздела `string.h`.

Программа 2_04.bas

```
REM Перевод числа в систему с основанием r
DECLARE FUNCTION NToStr$(NUM&,R%)
CLS
INPUT "Введите натуральное число : ",N&
PRINT "Его представление в разных системах счисления таково : "
FOR J%=2 TO 16
  PRINT "по основанию ";J%,N&;"= ";NToStr$(N&,J%)
NEXT J%
END

FUNCTION NToStr$(NUM&,R%)
  A$="":   M&=NUM&
  DO
    CIF=M& MOD R% : ' Выделение очередной цифры
    IF CIF<10 THEN
       A$=CHR$(ASC("0")+CIF)+A$ : ' Замена цифры кодом ASCII
    ELSE
       A$=CHR$(ASC("A")+CIF-10)+A$
    END IF
    M&=(M&-CIF)/R% : ' Исключение обработанной цифры
  LOOP UNTIL M&=0
  NToStr$=A$
END FUNCTION
```

Программа 2_04.c

```
/* Перевод числа в систему с основанием r */
#include <stdio.h>
#include <conio.h>
char *num_to_str(long num,char r);

void main(void)
{
  long N;
  char j;
  printf("\nВведите натуральное число ");
  scanf("%ld",&N);
  printf("Его представление в разных системах счисления таково :\n");
  for(j=2; j<17; j++)
    printf("\nпо основанию %2d = %s",j,num_to_str(N,j));
  getch();
}

char *num_to_str(long num, char r)
{
  static char a[33];
  char cif, k, count=0;
  do
    {
      cif=num % r; /* Выделение очередной цифры */
      if(cif<10) a[count++]='0'+cif;
/* Замена цифры кодом ASCII, если цифра меньше 10 */
      else a[count++]='A'+cif-10;
/* Замена цифры кодом ASCII, если цифра больше 9 */
      num=num/r; /* Исключение обработанной цифры */
    }
/* Цикл изменения порядка цифр в массиве a */
  while (num != 0);
  a[count--]=0x0;
  for(k=0; k<=count/2; k++)
     { cif=a[count-k];  a[count-k]=a[k]; a[k]=cif; }
  return a;
}
```

Программа 2_04.pas

```
program num_to_pos;
{ Перевод числа в систему с основанием r }
var
  N:longint;
  j:byte;
function num_to_str(num:longint;r:byte):string;
var
  a:string[33];
  cif,k:byte;
begin
  a:='';
  repeat
    cif:=num mod r; { Выделение очередной цифры }
    if (cif<10) then a:=chr(48+cif)+a
{ Замена цифры кодом ASCII, если цифра меньше 10 }
    else   a:=chr(65+cif-10)+a;
{ Замена цифры кодом ASCII, если цифра больше 9 }
    num:=num div r; { Исключение обработанной цифры }
  until (num=0);
  num_to_str:=a;
end;
begin
  write('Введите натуральное число - ');
  readln(N);
  writeln('Его представление в разных системах счисления таково :');
  for j:=2 to 16 do
    writeln('по основанию ',j:2,' = ',num_to_str(N,j));
  readln;
end.
```

Задание 2.05. Симметричное разложение с наименьшим основанием

Дано натуральное число `N` (`N < 30000`). Найти систему счисления с наименьшим основанием `Pmin`, в которой `N` имеет симметричное представление. Например,

для `N=9` `Pmin=2` `(910 = 10012)`,

для `N=1000` `Pmin=9` `(100010 = 13319)`

Программа должна запрашивать число `N`, выдавать `Pmin` и значения всех цифр в представлении числа `N` в этой системе (цифры можно выводить в виде обычных десятичных чисел).

Совет 1 (общий)

Очевидно, что самой расточительной по количеству цифр является двоичная система счисления. Однако при заданном ограничении на диапазон исходных чисел для записи самого большого числа `N` потребуется не более 15 двоичных разрядов. Поэтому для хранения цифр, получаемых при переводе в ту или иную систему счисления, вполне можно обойтись массивом из 15 элементов.

Второй очевидный факт заключается в том, что всегда найдется такая система счисления, в которой разложение любого числа `N` будет симметричным. Такой системой, в частности, является система с основанием `N-1`, т. к. в ней число `N` выглядит как `11`. Поэтому остается только организовать цикл по основаниям систем от `2` до `N-1` и, как только будет найдено симметричное разложение, прекратить перебор.

Совет 2 (общий)

Разумно организовать в программе две внешние функции — `perevod` (перевод исходного числа в систему с заданным основанием) и `proba` (проверка симметрии полученных цифр). Первая функция может быть построена по аналогии с функцией `num_to_str`, с той лишь разницей, что цифры, получаемые в процессе перевода, надо запоминать в массиве.

Совет 3 (QBasic)

Обратите внимание на схему вывода разложения, которое может не убраться в одной строке. Поэтому принято решение выводить в каждой строке по одному элементу разложения. Количества строк на экране вполне достаточно для размещения самого длинного представления в двоичной системе.

Программа 2_05.bas

```
REM Поиск симметричного разложения числа
DECLARE FUNCTION perevod!(n%,p%,a%())
DECLARE FUNCTION proba!(a%(),k%)
DIM a%(16)
CLS
INPUT "Введите число из диапазона [0,30000] : ",n%
FOR p%=2 TO n%-1
  k%=perevod(n%,p%,a%()) : ' Перевод в p-ричную систему
  IF proba(a%(),k%) <> 0 THEN
    PRINT "Симметричное разложение с минимальным основанием : "
    FOR i = 0 TO k% - 1
      PRINT TAB(0);a%(i);"*";p%; "^";k%-i;"+";
```

```
    NEXT i
    PRINT : PRINT a%(0);"*";p%;"^";0
    END
  END IF
NEXT p%
END

FUNCTION perevod (n%,p%,a%())
REM Перевод числа n в систему с основанием p
REM Цифры p-ричного числа запоминаются в массиве a
m%=n%
FOR i=0 TO 15
  a%(i)=m% MOD p%
  m%=(m%-a%(i))/p%
  IF m%=0 THEN perevod=i: EXIT FUNCTION
NEXT i
END FUNCTION

FUNCTION proba (a%(),k%)
REM Анализ числа, представленного k цифрами в массиве a
REM Если число - палиндром, то proba=1
proba=1
FOR i=0 TO k%/2
  IF a%(i)<>a%(k%-i) THEN proba=0: EXIT FUNCTION
NEXT i
END FUNCTION
```

Программа 2_05.c

```
/* Поиск симметричного разложения числа */
#include <conio.h>
#include <stdio.h>
int perevod(int n,int p,int *a);
int proba(int *a,int k);
  int i;
  main()
{
  int k,p,N,a[16];
  clrscr();
  scanf("%d",&N);
```

```
  for (p=2; p<N; p++)
    {
      k=perevod(N,p,a); /* Перевод в p-ричную систему */
      if (proba(a,k)==0) continue;
      printf("\nминимальное основание = %d\n",p);
      for(i=0; i<=k; i++)  printf("%d ",a[i]);
      break;
    }
  getch();
}
int perevod(int n,int p,int *a)
/* Перевод числа n в систему с основанием p
   Цифры p-ричного числа запоминаются в массиве a */
{
  for(i=0; i<16; i++)
    {
      a[i]=n % p;     n=n/p;
      if(n==0) return i;
    }
}
int proba(int *a,int k)
/* Анализ числа, представленного k цифрами в массиве a
   Если число - палиндром, то proba=1 */
{
  for(i = 0; i <= k/2; i++)
    if(a[i] != a[k-i]) return 0;
  return 1;
}
```

Программа 2_05.pas

```
program min_base;
{ Поиск симметричного разложения числа }
uses Crt;
var
  i,k,p,N:integer;
  a:array [0..15] of integer;
function perevod(n,p:integer):integer;
{ Перевод числа n в систему с основанием p
  Цифры p-ричного числа запоминаются в массиве a }
```

```
begin
  for i:=0 to 15 do
  begin
    a[i]:=n mod p;
    n:=n div p;
    if n=0 then
      begin perevod:=i; exit; end;
  end;
end;
function proba(k:integer):integer;
{ Анализ числа, представленного k цифрами в массиве a
  Если число - палиндром, то proba=1 }
begin
  for i:=0 to k div 2 do
    if a[i]<>a[k-i] then
    begin proba:=0; exit; end;
  proba:=1;
end;
begin
  clrscr;
  writeln('Поиск симметричного разложения с минимальным основанием');
  writeln('Введите число');
  readln(N);
  for p:=2 to N do
    begin
      k:=perevod(N,p);
      if proba(k)=0 then continue;
      writeln('минимальное основание = ',p);
      for i:=0 to k do  writeln(a[i]);
      break;
    end;
  readln;
end.
```

Задание 2.06. Суммирование десятичных цифр

Составить функцию `sum_dig(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно сумме десятичных цифр числа `n`.

Совет 1 (общий)

Обратите внимание на ситуацию, когда аргумент отрицателен. Остаток от деления отрицательного числа на 10 будет также отрицательным.

Совет 2 (QBasic)

Будьте внимательны при определении очередного частного от деления числа на 10. QBasic производит округления, и результат деления, например, целого числа 15 на целое число 10 даст в частном не 1, а 2.

Совет 3 (Си, Паскаль)

Обратите внимание на то, что циклы `do` (Си) и `repeat` (Паскаль) требуют противоположных условий выхода из цикла.

Программа 2_06.bas

```
REM Суммирование десятичных цифр числа
DECLARE FUNCTION SUMDIG (N&)
CLS
INPUT "Введите целое число";M&
K=SUMDIG(M&)
PRINT "Сумма его цифр = ";K
END

FUNCTION SUMDIG(N&)
IF N&<0 THEN N&=-N& : ' Смена знака у отрицательного числа
RESULT=0
DO
  RESULT=RESULT+(N& MOD 10&) : ' Накопление суммы цифр
  N&=(N&-(N& MOD 10&))/10& : ' Удаление обработанной цифры
LOOP WHILE N&<>0
SUMDIG=RESULT
END FUNCTION
```

Программа 2_06.c

```
/* Суммирование десятичных цифр числа */
#include <stdlib.h>
int sum_digits(long N);
  main()
{
```

```
  long M;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  printf("\nСумма его цифр = %d",sum_digits(M));
  getch();
}
int sum_digits(long N)
{
  int Result=0;
  N=labs(N); /* Смена знака у отрицательного числа */
  do
    {
      Result=Result+N%10; /* Накопление суммы цифр */
      N=N/10; /* Удаление обработанной цифры */
    }
    while(N != 0);
    return Result;
}
```

Программа 2_06.pas

```
program sumdigits;
{ Суммирование десятичных цифр числа }
var
  M:longint;
  function sum_digits(N:longint):integer;
  const
    Result:integer=0;
  begin
    if N<0 then N:=-N; { Смена знака у отрицательного числа }
    repeat
      Result:=Result+N mod 10; { Накопление суммы цифр }
      N:=N div 10; { Удаление обработанной цифры }
    until N=0;
    sum_digits:=Result;
  end;
begin
  write('Введите целое число:');
  readln(M);
```

```
  writeln('Сумма его цифр =',sum_digits(M));
  readln;
end.
```

Задание 2.07. "Счастливый" билет

Составить логическую функцию `luck(n)`, аргументом которой является число `n` из диапазона [0, 999999]. Функция должна возвращать значение `true` (Паскаль) или `1` (Си, QBasic), если ее аргумент представлен "счастливым" числом, у которого суммы трех старших и трех младших цифр совпадают.

Программа 2_07.bas

```
REM Анализ "счастливого" билета
DECLARE FUNCTION LUCK(M AS LONG)
INPUT "Введите номер билета ";N&
IF LUCK(N&)=1 THEN
  PRINT "Радуйтесь - счастливый"
ELSE
  PRINT "Нет счастья в жизни"
END IF
END

FUNCTION LUCK(M AS LONG)
REM Подсчет и сравнение сумм старших и младших цифр M
REM Если суммы совпадают LUCK=1
DIM A(6)
LUCK=0
IF M<0 OR M>999999 THEN
  PRINT "luck : недопустимый аргумент": EXIT FUNCTION
END IF
FOR I=0 TO 5
  A(I)=M MOD 10 : ' Выделение очередной цифры
  M=(M-A(I))/10 : ' Удаление обработанной цифры
NEXT I
IF (A(0)+A(1)+A(2)=A(3)+A(4)+A(5)) THEN LUCK=1
END FUNCTION
```

Программа 2_07.c

```
/* Анализ "счастливого" билета */
#include <stdio.h>
int luck(long M);
  main()
{
  long N;
  printf("\nВведите номер билета ");
  scanf("%ld",&N);
  if (luck(N)==1)
    printf("\nРадуйтесь - счастливый");
  else
    printf("\nНет счастья в жизни");
  getch();
}
int luck(long M)
/* Подсчет и сравнение сумм старших и младших цифр M
   Если суммы совпадают luck=1*/
{
  int i;
  char a[6];
  if((M<0) || (M>999999))
    {
      printf("\nluck : недопустимый аргумент");
      exit(0);
    }
  for(i=0; i<6; i++)
     {
       a[i]=M % 10; /* Выделение очередной цифры */
       M=M/10; /* Удаление обработанной цифры */
     }
  if(a[0]+a[1]+a[2]==a[3]+a[4]+a[5]) return 1;
  return 0;
}
```

Программа 2_07.pas

```
program lucky_ticket;
{ Анализ "счастливого" билета }
```

```
var
  N:longint;
function luck(M:longint):boolean;
{ Подсчет и сравнение сумм старших и младших цифр M
   Если суммы совпадают luck=true }
var
  i:integer;
  a:array [1..6] of byte;
begin
  if (M<0) or (M>999999) then
  begin
    writeln('luck : недопустимый аргумент');
    exit;
  end;
  for i:=1 to 6 do
  begin
    a[i]:=M mod 10; { Выделение очередной цифры }
    M:=M div 10;    { Удаление обработанной цифры }
  end;
  luck:=(a[1]+a[2]+a[3])=(a[4]+a[5]+a[6]);
end;
begin
  writeln('Введите номер билета');
  readln(N);
  if luck(N) then
    writeln('Радуйтесь - счастливый')
  else
    writeln('Нет счастья в жизни');
  readln;
end.
```

Задание 2.08. Количество различных цифр в десятичном числе

Составить функцию `num_digits(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно количеству различных цифр в десятичной записи `n`. Например: `num_digits(1998)=3`.

Совет 1 (общий)

Предлагается завести массив из 10 счетчиков, каждый из которых будет подсчитывать количество своих цифр — первый счетчик считает нули, второй —

единицы и т. д. Вначале массив счетчиков обнуляется, а после выделения всех цифр числа проверяется, сколько счетчиков хранят ненулевые значения. Вообще говоря, можно не заниматься подсчетом количества появлений каждой цифры — достаточно фиксировать сам факт появления той или иной цифры.

Совет 2 (Си, Паскаль)

Обратите внимание на различное оформление функций `num_digits` на Си и Паскале. Локальные переменные в функциях на Си, которым присваиваются начальные значения, при повторных обращениях к функциям инициализируются заново. В процедурах и функциях Паскаля такая инициализация производится только один раз, поэтому повторный вызов может привести к неверным результатам.

Программа 2_08.bas

```
REM Определение количества различных цифр в числе
DECLARE FUNCTION NumDig!(N&)
CLS
INPUT "Введите число : ", N&
PRINT "Количество разных цифр в его записи = ";NumDig(N&)
END

FUNCTION NumDig(N&)
REM Выделение и подсчет количества разных цифр в числе N
DIM d(10) : ' Массив для фиксации обнаруженных цифр
REM При вызове функции d(i)=0
IF N&<10 THEN NumDig=1: EXIT FUNCTION
DO
  k%=N& MOD 10 :  ' Выделение очередной цифры
  d(k%)=d(k%)+1 : ' Подсчет количества цифр, равных k
  N&=(N&-k%)/10 : ' Удаление обработанной цифры
LOOP UNTIL N&=0
FOR k%=0 TO 9 : ' Цикл подсчета количества обнаруженных цифр
  IF d(k%)<>0 THEN s=s+1 : ' Если d(i)=0, цифры i не было
NEXT k%
NumDig=s
END FUNCTION
```

Программа 2_08.c

```
/* Определение количества различных цифр в числе */
#include <stdio.h>
#include <conio.h>
```

```
int num_digits(long N);
  main()
{
  long N;
  printf("\nВведите число - ");
  scanf("%ld",&N);
  printf("\Количество разных цифр в его записи = %d ",
          num_digits(N));
  getch();
}
int num_digits(long N)
/* Выделение и подсчет количества разных цифр в числе N */
{
  char d[10]={0,0,0,0,0,0,0,0,0,0}; /* Массив счетчиков цифр */
  int k,s=0;
  if(N<10) return 1;
  while (N)
    {
      d[N % 10]++; /* Выделение и учет очередной цифры */
      N=N/10; /* Удаление обработанной цифры */
    }
/* Цикл подсчета количества обнаруженных цифр */
  for(k=0; k<10; k++) if(d[k]) s++;
  return s;
}
```

Программа 2_08.pas

```
program NumDigits;
{ Определение количества различных цифр в числе }
var
  N:longint;
function num_digits(N:longint):byte;
{ Выделение и подсчет количества разных цифр в числе N }
var
   d:array [0..9] of byte; { Массив счетчиков }
   s,k:byte;
begin
  if N<10 then num_digits:=1
```

```
  else
    begin
      for k:=0 to 9 do d[k]=0; { Сброс счетчиков }
      s:=0;
      while N <> 0 do
        begin
          inc(d[N mod 10]); { Выделение и учет очередной цифры }
          N := N div 10; { Удаление обработанной цифры }
        end;
{ Цикл подсчета количества обнаруженных цифр }
      for k:=0 to 9 do if(d[k]<>0) then inc(s);
      num_digits:=s;
    end;
end;

begin
  write('Введите число : ');
  readln(N);
  writeln('Количество разных цифр в его записи = ',num_digits(N));
  readln;
end.
```

Задание 2.09. Определение цифры в заданной позиции

Составить функцию `digit_in_pos(n,i)`, аргументами которой являются длинное целое число `n` и номер `i` позиции цифры в десятичном представлении `n`. Отсчет номеров позиций ведется справа налево от 0 и соответствует "весу" цифры (степени основания), с которым цифра входит в число. Например:

```
n=1985
  в  0-й позиции находится цифра 5;
  в  1-й позиции находится цифра 8;
  во 2-й позиции находится цифра 9;
  в  3-й позиции находится цифра 1;
```

Функция `digit_in_pos` должна возвращать цифру, расположенную в `i`-й позиции числа `n`.

Программа 2_09.bas

```
REM Анализ цифр в каждой позиции заданного числа
DECLARE FUNCTION DIGINPOS(N AS LONG,J AS INTEGER)
```

```
INPUT "Введите целое число: ";M&
FOR K%=0 TO 9
  DIGIT=DIGINPOS(M&,K%)
  PRINT "В позиции ";K%;" находится ",DIGIT
NEXT K%
END

FUNCTION DIGINPOS(N AS LONG,J AS INTEGER)
REM Определение десятичной цифры числа N в позиции j
N1&=N
FOR K%=0 TO J
  RESULT=N1& MOD 10
  N1&=N1&-RESULT)/10
NEXT K%
DIGINPOS=RESULT
END FUNCTION
```

Программа 2_09.c

```
/* Анализ цифр в каждой позиции заданного числа */
#include <stdlib.h>
int digit_in_pos(long N,int j);
  main()
{
  long M;
  int k;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  for(k=0; k<10; k++)
    printf("\nВ позиции %d находится %d",k,digit_in_pos(M,k));
  getch();
}
int digit_in_pos(long N,int j)
/* Определение десятичной цифры числа N в позиции j */
{
  int Result,k;
  for (k=0; k<=j; k++)
   {
     Result=N % 10;
     N=N/10;
```

```
    }
  return Result;
}
```

Программа 2_09.pas

```
program DigitInPos;
{ Анализ цифр в каждой позиции заданного числа }
var
  M:longint;
  k:integer;
function digit_in_pos(N:longint;j:integer):integer;
{ Определение десятичной цифры числа N в позиции j }
var
  Result,k:integer;
begin
  for k:=0 to j do
  begin
    Result:=N mod 10;
    N:=N div 10;
  end;
  digit_in_pos:=Result;
end;
begin
  writeln('Введите целое число:');
  readln(M);
  for k:=0 to 9 do
    writeln('В позиции ',k,' находится ',digit_in_pos(M,k));
  readln;
end.
```

Задание 2.10. Генерация чисел с заданной суммой цифр

Составить программу, которая выдает все числа из диапазона [0, 999], сумма цифр которых равна вводимому числу `N` (`0 ≤ N ≤ 27`).

Программа 2_10.bas

```
REM Генерация чисел с заданной суммой цифр
INPUT "Введите число в диапазоне от 0 до 27 : ",n
```

```
PRINT "Список чисел, сумма цифр которых равна ";n;" :"
FOR a2%=0 TO 9
  FOR a1%=0 TO 9
    FOR a0%=0 TO 9
      IF a2%+a1%+a0%=n THEN
        PRINT USING "####";100*a2%+10*a1%+a0%;
      END IF
NEXT a0%,a1%,a2%
END
```

Программа 2_10.c

```
/* Генерация чисел с заданной суммой цифр */
#include <stdio.h>
#include <conio.h>
  main()
{
  int a2,a1,a0,n;
  printf("\nВведите число в диапазоне от 0 до 27 : ");
  scanf("%d",&n);
  printf("\nСписок чисел, сумма цифр которых равна %d :\n",n);
  for(a2=0; a2<10; a2++)
    for(a1=0; a1<10; a1++)
      for(a0=0; a0<10; a0++)
        if(a2+a1+a0==n) printf("%4d",100*a2+10*a1+a0);
  getch();
}
```

Программа 2_10.pas

```
program numbers;
{ Генерация чисел с заданной суммой цифр }
var
  a2,a1,a0,n:integer;
begin
  write('Введите число в диапазоне от 0 до 27 : ');
  readln(n);
  writeln('Список чисел, сумма цифр которых равна ',n, ' :');
  for a2:=0 to 9 do
```

```
    for a1:=0 to 9 do
      for a0:=0 to 9 do
        if (a2+a1+a0)=n then write(100*a2+10*a1+a0:4);
  readln;
end.
```

Задание 2.11. Вывод числа словами

Составить строковую функцию `num_to_str(n)`, аргументом которой является целое число (`|n| < 1000`). Возвращаемое значение должно быть строкой, в которой величина `n` представлена словами. Например:

`num_to_str(-87) = "минус восемьдесят семь".`

Совет 1 (общий)

Очевидно, что придется завести набор строк с символьными эквивалентами некоторых чисел. Конечно, нежелательно, чтобы в таком массиве было использовано 1000 констант — "ноль", "один", ..., "девятьсот девяносто девять". Однако без минимального набора из трех массивов — малые числа ("`ноль`", "`один`", ..., "`девятнадцать`"), десятки ("`двадцать`", "`тридцать`", ..., "`девяносто`") и сотни ("`сто`", "`двести`", ..., "`девятьсот`") — нам не обойтись. Перед анализом цифр числа у него целесообразно удалить знак и, в случае необходимости, приклеить в начале результирующей строки значение "`минус`".

Совет 2 (Си)

Для правильной работы функции `num_to_str` следует подключить заголовочный файл `string.h`. Обратите внимание на объявление локальной переменной `Result` в теле функции. Если исключить из этого описания спецификатор `static`, то переменная `Result` превратится в локальную строку и память под нее будет выделена только на время работы функции. Поэтому нет никакой гарантии, что сформированный в ней результат работы функции сохранится после возврата в головную программу. Попробуйте удалить описатель `static` и сравните значение `Result` в теле функции с тем, что выдаст головная программа, например при `N=999`.

Совет 3 (QBasic)

Система QBasic не допускает использования операторов `DATA-READ` в теле функции. Поэтому можно сформировать значения элементов символьных массивов `num1$`, `num2$` и `num3$` в головной программе, а затем передать их в списке параметров при вызове функции. На этом примере вы можете ознакомиться с использованием массивов в качестве формальных (при описании заголовка функции) и фактических (при обращении к функции) параметров. Вторая возможность, реализованная в программе `2_11a.bas`, заключается в объявлении совместного доступа к массивам `num1$`, `num2$` и `num3$` с помощью оператора `DIM SHARED`.

Совет 4 (Паскаль)

В качестве наиболее простого способа задания значений элементов массивов num1, num2 и num3 предлагается поместить соответствующие описания в раздел типизированных "констант".

Программа 2_11.bas

```
REM Формирование словесного описания числа
DECLARE FUNCTION NumToStr$(m%,num1$(),num2$(),num3$())
CLS
DATA "ноль","один","два","три","четыре","пять","шесть","семь"
DATA "восемь","девять","десять","одиннадцать","двенадцать"
DATA "тринадцать","четырнадцать","пятнадцать","шестнадцать"
DATA "семнадцать","восемнадцать","девятнадцать"
DIM num1$(20)
FOR k=0 TO 19: READ num1$(k): NEXT k

DATA "двадцать ","тридцать ","сорок ","пятьдесят "
DATA "шестьдесят ","семьдесят ","восемьдесят ","девяносто "
DIM num2$(8)
FOR k=0 TO 7: READ num2$(k): NEXT k

DATA "","сто ","двести ","триста ","четыреста ","пятьсот "
DATA "шестьсот ","семьсот ","восемьсот ","девятьсот "
DIM num3$(10)
FOR k=0 TO 9: READ num3$(k): NEXT k
INPUT "Введите целое число от -999 до 999: ",n%
PRINT n%;"= ";NumToStr$(n%,num1$(),num2$(),num3$())
END

FUNCTION NumToStr$(m%,num1$(),num2$(),num3$())
IF m%=0 THEN NumToStr$=num1$(0): EXIT FUNCTION
IF m%<0 THEN m%=-m%: Res$="минус " : 'Учет знака числа
dig100%=m%\100 :` Выделение сотен
Res$=Res$+num3$(dig100%) :'Приклеили обозначение сотен
m%=m%-100*dig100% : 'Удаление обработанных сотен
IF m%=0 THEN NumToStr$=Res$: EXIT FUNCTION
IF m%<20 THEN NumToStr$=Res$+num1$(m%): EXIT FUNCTION
dig10%=m%\10 :' Выделение десятков, если dig10 >=20
```

```
Res$=Res$+num2$(dig10%-2)
dig1%=m% MOD 10 :' Приклеили обозначение десятков
REM Если в числе присутствуют ненулевые разряды единиц
IF dig1%<>0 THEN Res$=Res$+num1$(dig1%)
NumToStr$=Res$
END FUNCTION
```

Программа 2_11a.bas

```
REM Формирование словесного описания числа
DECLARE FUNCTION NumToStr$(m%)
CLS
DATA "нуль","один","два","три","четыре","пять","шесть","семь"
DATA "восемь","девять","десять","одиннадцать","двенадцать"
DATA "тринадцать","четырнадцать","пятнадцать","шестнадцать"
DATA "семнадцать","восемнадцать","девятнадцать"
DIM SHARED num1$(20)
FOR k=0 TO 19: READ num1$(k): NEXT k

DATA "двадцать ","тридцать ","сорок ","пятьдесят "
DATA "шестьдесят ","семьдесят ","восемьдесят ","девяносто "
DIM SHARED num2$(8)
FOR k=0 TO 7: READ num2$(k): NEXT k

DATA "","сто ","двести ","триста ","четыреста ","пятьсот "
DATA "шестьсот ","семьсот ","восемьсот ","девятьсот "
DIM SHARED num3$(10)
FOR k=0 TO 9: READ num3$(k): NEXT k
INPUT "Введите целое число от -999 до 999: ",n%
PRINT n%;"= ";NumToStr$(n%)
END

FUNCTION NumToStr$(m%)
  IF m%=0 THEN NumToStr$=num1$(0): EXIT FUNCTION
  IF m%<0 THEN m%=-m%: Res$="минус " : 'Учет знака числа
  dig100%=m%\100 : 'Выделение сотен
  Res$=Res$+num3$(dig100%) :' Приклеили обозначение сотен
  m%=m%-100*dig100% : ' Удаление обработанных сотен
  IF m%=0 THEN NumToStr$=Res$: EXIT FUNCTION
```

```
  IF m%<20 THEN NumToStr$=Res$+num1$(m%): EXIT FUNCTION
  dig10%=m%\10 :' Выделение десятков, если dig10 >=20
  Res$=Res$+num2$(dig10%-2) :' Приклеили обозначение десятков
  dig1%=m% MOD 10
REM Если в числе присутствуют ненулевые разряды единиц
  IF dig1%<>0 THEN Res$=Res$+num1$(dig1%)
  NumToStr$=Res$
END FUNCTION
```

Программа 2_11.c

```
/* Формирование словесного описания числа */
#include <stdio.h>
#include <conio.h>
#include <string.h>
char *num_to_str(long m);
  main()
{
  long n;
  clrscr();
m:
  printf("\nВведите целое число : ");
  scanf("%ld",&n);
  printf("\n%ld = %s",n,num_to_str(n));
  goto m;
  getch();
}
char *num_to_str(long m)
/* Преобразование числа в словесное описание */
{
  char *num1[]={"ноль","один","два","три",
  "четыре","пять","шесть","семь","восемь","девять","десять",
  "одиннадцать","двенадцать","тринадцать","четырнадцать",
  "пятнадцать","шестнадцать","семнадцать", "восемнадцать",
  "девятнадцать"};
  char *num2[]={
  "двадцать ","тридцать ","сорок ","пятьдесят ","шестьдесят ",
  "семьдесят ","восемьдесят ","девяносто "};
  char *num3[]={"",
```

```
  "сто ","двести ","триста ","четыреста ","пятьсот ",
  "шестьсот ","семьсот ","восемьсот ","девятьсот "};
  static char Result[50]="";
  int dig1,dig10,dig100;

  if(m==0) return num1[0];
  Result[0]=0x0;
  if(m<0) /* Учет знака числа */
    {
      m=-m;
      strcpy(Result,"минус ");
    }
  dig100=m/100; /* Выделение сотен */
  strcat(Result,num3[dig100]); /* Приклеили обозначение сотен */
  m=m-100*dig100; /* Удаление обработанных сотен */
  if(m==0) return Result; /* Если две оставшиеся цифры - нули */
  if(m<20) /* Если две младшие цифры меньше 20 */
    {
      strcat(Result,num1[m]);
      return Result;
    }
  dig10=m/10; /* Выделение десятков, если dig10 >=20 */
  strcat(Result,num2[dig10-2]);
  dig1=m % 10;
/* Если в числе присутствуют ненулевые разряды единиц */
  if(dig1 != 0)
     strcat(Result,num1[dig1]);
  return Result;
}
```

Программа 2_11.pas

```
program nd_10;
{ Формирование словесного описания числа }
uses Crt;
var
  n:longint;
function num_to_str(m:longint):string;
{ Преобразование числа в словесное описание }
```

```
label
  ret;
const
  num1:array [0..19] of string=(
      'ноль', 'один', 'два', 'три', 'четыре','пять','шесть',
      'семь','восемь','девять','десять','одиннадцать',
      'двенадцать','тринадцать','четырнадцать',
      'пятнадцать','шестнадцать','семнадцать',
      'восемнадцать','девятнадцать');

  num2:array [2..9] of string=(
      'двадцать ','тридцать ','сорок ','пятьдесят ',
      'шестьдесят ','семьдесят ','восемьдесят ','девяносто ');

  num3:array [0..9] of string = ('',
      'сто ','двести ','триста ','четыреста ','пятьсот ',
      'шестьсот ','семьсот ','восемьсот ','девятьсот ');

var
  dig1,dig10,dig100: byte;
  Result:string;
begin
  if m=0 then begin
     Result:=num1[0]; goto ret;
  end;
  Result:='';
  if m<0 then  { Учет знака числа }
  begin
    m:=-m;
    Result:='минус ';
  end;
  dig100:=m div 100; { Выделение сотен }
  Result:=Result + num3[dig100]; { Приклеили обозначение сотен }
  m:=m-100*dig100; { Удаление обработанных сотен }
  if m=0 then goto ret; { Если две оставшиеся цифры - нули }
  if m<20 then begin { Если две младшие цифры меньше 20 }
    Result:=Result+num1[m]; goto ret;
  end;
  dig10:=m div 10; { Выделение десятков, если dig10 >=20 }
```

```
  Result:=Result+num2[dig10];
  dig1:=m mod 10;
{ Если в числе присутствуют ненулевые разряды единиц }
  if dig1<>0 then Result:=Result + num1[dig1];
ret:
  num_to_str:=Result;
  end;
begin
  clrscr;
  write('Введите целое число : ');
  readln(n);
  writeln(n,' = ',num_to_str(n));
  readln;
end.
```

Задание 2.12. Суммирование двоичных цифр

Составить функцию `sum_bits(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно количеству единиц в двоичном представлении числа `n`.

Совет 1 (общий)

Обратите внимание на тот факт, что отрицательные числа представлены в памяти компьютера в дополнительном коде. Поэтому число –1 содержит 32 единицы в своем двоичном разложении.

Совет 2 (QBasic)

Так как входной язык QBasic не содержит операции сдвига, схитрим — разделим на 2 битовую шкалу `J&`, заставив сдвинуться влево ее единственную единицу. Начать с `J&=&H80000000` нельзя, т. к. это очень странное "число". Поэтому приходится знак `N` проверять отдельно, а в цикле анализ производить, начиная с 30-го разряда.

Программа 2_12.bas

```
REM Суммирование двоичных цифр введенного числа
INPUT "Введи N"; N&
J&=&H40000000
RESULT=0
IF N&<0 THEN RESULT=1 : ' Учет 1 в знаке отрицательного числа
FOR K=31 TO 1 STEP -1
```

```
  IF N& AND J& THEN RESULT=RESULT+1 : ' Учет k-того разряда
  J&=J&/2 : ' Сдвиг шкалы на разряд вправо
NEXT K
PRINT RESULT
END
```

Программа 2_12.c

```
/* Суммирование двоичных цифр введенного числа */
#include <stdlib.h>
int sum_bits(long N);
  main()
{
  long M;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  printf("\nСумма его цифр = %d",sum_bits(M));
  getch();
}
int sum_bits(long N)
/* Подсчет единиц в двоичном представлении N */
{
  int Result=0,k;
  unsigned long j=0x80000000;
  for(k=31; k >= 0; k--)
    {
     if(j & N) Result++; /* Учет k-того разряда */
     j=j >> 1;
    }
  return Result;
}
```

Программа 2_12.pas

```
program sumbits;
{ Суммирование двоичных цифр введенного числа }
var
  M:longint;
function sum_bits(N:longint):integer;
```

```
{ Подсчет единиц в двоичном представлении N }
const
  Result:integer=0;
  j:longint=$80000000;
var
  k:integer;
begin
  for k:=31 downto 0 do
  begin
    if (N and j)<>0 then inc(Result); { Учет k-того разряда }
    j:=j shr 1;
  end;
  sum_bits:=Result;
end;
begin
  write('Введите целое число: ');
  readln(M);
  writeln('Сумма его цифр = ',sum_bits(M));
  readln;
end.
```

Задание 2.13. Позиция старшей единицы

Составить функцию `left_bit(n)`, аргументом которой является длинное целое положительное число. Возвращаемое значение должно совпадать с номером старшей единицы в двоичном представлении `n`. Нумерация двоичных разрядов ведется справа налево от 0 до 31.

Совет 1 (общий)

Наиболее прямой способ заключается в проверке битов заданного числа слева (от самого старшего разряда) направо. Как только в цикле проверки появляется первая единица, работа подпрограммы завершается. Так как аргумент функции неотрицателен, проверку можно начинать с 30-го разряда. Для сдвига битовой шкалы в Си следует использовать операцию сдвига вправо (`>>`), в Паскале — операцию `shr`, а в QBasic придется прибегнуть к делению на 2.

Программа 2_13.bas

```
REM Определение позиции старшей единицы в двоичном числе
DECLARE FUNCTION LeftBit!(N&)
INPUT "Введите целое число: ",M&
```

```
IF M&=0 THEN PRINT "В этом числе единиц нет" : END
PRINT "Старший разряд находится в позиции номер ";LeftBit(M&)
END

FUNCTION LeftBit(N&)
REM Определение позиции старшей единицы в числе N
REM Если N=0, то LeftBit=-1
LeftBit=-1
j&=&H40000000
FOR k=30 TO 0 STEP -1
  IF (j& AND N&) THEN ' Анализ k-го разряда
    LeftBit=k
    EXIT FUNCTION
  END IF
  j&=j&/2
NEXT k
END FUNCTION
```

Программа 2_13.c

```
/* Определение позиции старшей единицы в двоичном числе */
#include <stdlib.h>
int left_bit(long N);
  main()
{
  long M;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  if(M==0)
    {
     printf("\nВ этом числе единиц нет"); exit(0);
    }
  printf("\nСтарший разряд находится в позиции номер %d", left_bit(M));
  getch();
}
int left_bit(long N)
/* Определение позиции старшей единицы в числе N
   Если N=0, функция возвращает -1 */
{
  int k;
```

```
  long j=0x80000000;
  for(k=31; k >= 0; k--)
    {
      if(j & N) return k;
      j=j >> 1;
    }
  return -1;
}
```

Программа 2_13.pas

```
program LeftBit;
{ Определение позиции старшей единицы в двоичном числе }
var
  M:longint;
function left_bit(N:longint):byte;
  {Определение позиции старшей единицы в числе N
 Если N=0, функция возвращает -1 }
var
  k:byte;
const
  j:longint=$80000000;
begin
  left_bit:=-1;
  for k:=31 downto 0 do
    begin
      if (j and N)<> 0 then
         begin
           left_bit:=k;
           exit;
         end;
      j:=j shr 1;
    end;
end;

begin
  write('Введите целое число: ');
  readln(M);
  if M=0 then writeln('В этом числе единиц нет')
  else
```

```
  writeln('Старший разряд находится в позиции номер ', left_bit(M));
  readln;
end.
```

Задание 2.14. Максимальное количество подряд стоящих единиц

Составить функцию `max_bits(n)`, аргументом которой является длинное целое положительное число. Возвращаемое значение должно быть равно максимальному числу подряд расположенных единиц в двоичном представлении `n`. Например:

$n = 33_{10} = 10001_2$ `max_bits(33)=1`

$n = 28_{10} = 11100_2$ `max_bits(28)=3`

Совет 1 (общий)

Основная идея заключается в том, чтобы вести подсчет подряд стоящих единиц в двоичном представлении числа `n` до ближайшего нуля и сравнивать это количество с предшествующим претендентом на максимум. При этом не следует забывать, что отрицательные числа в памяти компьютера представлены в дополнительном коде и, например, такое число, как –1 (шестнадцатеричный эквивалент `0xFFFFFFFF`) содержит 32 единицы. Это означает, что знаковый разряд должен учитываться наравне с остальными. В QBasic такой учет вызывает дополнительные трудности.

Совет 2 (QBasic)

Так как входной язык не предлагает операцию сдвига кодов, то для сдвига вправо можно воспользоваться умножением на 2.

Совет 3 (Си)

Для большей наглядности тестирующей программы можно воспользоваться библиотечной функцией `ltoa` и вывести двоичное представление числа `n`. Проверку битов целесообразно начать с самого старшего знакового разряда и в цикле сдвигать тестируемый код на 1 разряд влево. Если начать проверку с младшего разряда и сдвигать проверяемое число на 1 разряд вправо, то при отрицательном аргументе возникает бесконечный цикл из-за размножения знакового разряда. Условие окончания цикла (`N` не равно 0) при этом никогда не выполнится. Конечно, эту неприятность можно обойти, организовав фиксированное количество повторений цикла (точно 32 раза). Однако в этом случае зачастую совершается лишняя работа — единиц уже нет, а цикл еще не закончился. Можно предложить и такой вариант, когда вместо сдвига аргумента сдвигается тестирующая шкала.

Программа 2_14.bas

```
REM Определение максимальной группы единиц в двоичном числе
DECLARE FUNCTION MaxBits!(N&,s$)
```

```
CLS
INPUT "Введите целое число : ",M&
k%=MaxBits(M&,a$)
PRINT "Двоичное представление этого числа :"
PRINT a$
PRINT "Максимальное число подряд стоящих единиц = ";k%
END

FUNCTION MaxBits(N&,s$)
REM Перевод числа N в двоичное символьное представление
REM Поиск самой длинной группы единиц (MaxBits = длина)
IF N&=0 THEN MaxBits=0: s$="0": EXIT FUNCTION
FOR j=0 TO 31 : ' Формирование двоичного числа в строке s$
  IF N& AND &H1 THEN
    kBits=kBits+1 : ' Подсчет подряд идущих единиц
    IF kBits>max THEN max=kBits : ' Запомнили максимум
    s$="1"+s$ : ' Приписали очередную единицу
  ELSE
    kBits=0 : ' Сброс счетчика при встрече нуля
    s$="0"+s$ : ' Приписали очередной ноль
  END IF
  N&=(N&-k)/2 : ' Сдвиг на один разряд вправо
NEXT j
MaxBits=max
END FUNCTION
```

Программа 2_14.c

```
/* Определение максимальной группы единиц в двоичном числе */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int max_bits(long N);
  main()
{
  long M;
  char str[33];
  clrscr();
```

```
  printf("\nВведите целое число : ");
  scanf("%ld",&M);
  printf("Двоичное представление этого числа :");
  ltoa(M,str,2);
  printf("\n%32s",str);
  printf("\nМаксимальное число подряд стоящих единиц = %d",
          max_bits(M));
  getch();
}
int max_bits(long N)
/* Перевод числа N в двоичное символьное представление
   Поиск самой длинной группы единиц (max_bits = длина) */
{
  int k_bits=0,max=0;
  char s[33];
  if(N==0) return 0;
  while (N) /* Формирование двоичного числа в строке s$ */
   {
    if(N & 0x80000000) /* если очередной разряд равен 1 */
      { /* подсчет подряд идущих единиц */
       k_bits++;
       if(k_bits>max) max=k_bits; /* запоминание максимума */
      }
    else
      k_bits=0; /* сброс счетчика, если очередной разряд = 0 */
    N=N << 1 ;  /* сдвиг на 1 разряд влево */
  }
  return max;
}
```

Программа 2_14.pas

```
program MaxBits;
{ Определение максимальной группы единиц в двоичном числе }
uses Crt;
var
  M:longint;
  str:string[33];
function max_bits(N:longint):shortint;
```

```
{ Перевод числа N в двоичное символьное представление
  Поиск самой длинной группы единиц (max_bits = длина) }
var
  k_bits, max:byte;
begin
  k_bits:=0;
  max:=0;
  if N=0 then
    begin
      max_bits:=0;
      exit;
    end;
  while N<>0 do { Формирование двоичного числа в строке s$ }
    begin
      if (N and $00000001)<>0 then { если разряд равен 1 }
      begin
        str:='1'+str;
        inc(k_bits); { подсчет подряд идущих единиц }
        if k_bits>max then max:=k_bits
      end
      else
      begin
        str:='0'+str;
        k_bits:=0; { сброс счетчика, если очередной разряд = 0 }
      end;
      N:= N shr 1; { сдвиг на 1 разряд вправо }
    end;
  max_bits:=max;
end;

begin
  clrscr;
  write('Введите целое число : ');
  readln(M);
  writeln('Максимальное число подряд стоящих единиц = ',max_bits(M));
  readln;
end.
```

Задание 2.15. Расстояние между двоичными кодами

Под "расстоянием" между двумя n-разрядными двоичными кодами понимают количество несовпадений в каждой из n позиций. Например:

```
0 1 0 1 1 0 1 0
0 0 1 0 1 1 1 0
--------------
= * * * = * = =   "расстояние"=4
```

Составить функцию `rasst(n1,n2)`, аргументами которой являются два длинных положительных числа, рассматриваемые как 32-разрядные двоичные коды. Возвращаемое функцией значение должно совпадать с "расстоянием" между `n1` и `n2`.

Совет 1 (общий)

Наиболее рациональный алгоритм — воспользоваться логической операцией `XOR`, осуществляющей поразрядное сложение двоичных кодов по модулю 2, а затем подсчитать общее количество двоичных единиц в полученном результате. Менее эффективный по числу тактов способ состоит в сравнении одноименных разрядов каждого аргумента функции.

Программа 2_15.bas

```
REM Определение расстояния между двоичными кодами
DECLARE FUNCTION NumToBin$(N&)
DECLARE FUNCTION SumBits!(N&)
CLS
INPUT "Введи первое число :",N1&
INPUT "Введи второе число :",N2&
PRINT "Двоичное представление этих чисел :"
PRINT NumToBin$(N1&) : ' Двоичное разложение N1
PRINT NumToBin$(N2&) : ' Двоичное разложение N2
PRINT "Расстояние между этими кодами = "; SumBits(N1& XOR N2&)
END

FUNCTION NumToBin$(N&)
REM Формирование в строке двоичного представления числа N
a$="": K=&H1
IF N&=0 THEN NumToBin$="0": EXIT FUNCTION
FOR J=0 TO 30
  IF N& AND K THEN s$="1"+s$ ELSE s$="0"+s$
  K=K*2 : ' Сдвиг шкалы на 1 разряд влево
```

```
NEXT J
IF N&<0 THEN s$="1"+s$ : ' Учет единицы в знаковом разряде
NumToBin$=s$
END FUNCTION

FUNCTION SumBits (N&)
REM Подсчет количества единиц в двоичном числе
J&=&H40000000
RESULT=0
IF N&<0 THEN RESULT=1
FOR K=31 TO 1 STEP -1
  IF N& AND J& THEN RESULT=RESULT+1
  J&=J&/2
NEXT K
SumBits=RESULT
END FUNCTION
```

Программа 2_15.c

```
/* Определение расстояния между двоичными кодами */
#include <stdlib.h>
int rasst(long n1, long n2);
char s[40];
  main()
{
  long M1,M2;

  printf("\nВведите два целых числа : ");
  scanf("%ld %ld",&M1,&M2);
  printf("\nM1 =%32s",ltoa(M1,s,2)); /* Двоичное разложение N1*/
  printf("\nM2 =%32s",ltoa(M2,s,2)); /* Двоичное разложение N2*/
  printf("\nXOR=%32s",ltoa(M1^M2,s,2));
  printf("\nРасстояние между этим кодами = %5d",rasst(M1,M2));
  getch();
}
int rasst(long n1,long n2)
{
  int Result=0,k;
```

```
  long j=0x00000001;
  for(k=0; k <32; k++)
    { /* Сравнение одноименных двоичных разрядов */
     if((j & n1)!=(j & n2)) Result++;
     j=j << 1; /* сдвиг шкалы на разряд влево */
    }
  return Result;
}
```

Программа 2_15.pas

```
program distance;
{ Определение расстояния между двоичными кодами }
var
  M1,M2:longint;
function rasst(n1,n2:longint):byte;
const
  Result:byte=0;
  j:longint=$00000001;
var
  k:byte;
  n3:longint;
begin
  n3:=n1 xor n2;
  for k:=0 to 31 do { цикл подсчета единиц в двоичном числе }
    begin
      if (j and n3)<>0 then inc(Result);
      j:=j shl 1; { сдвиг шкалы на разряд влево }
    end;
  rasst:=Result;
end;
begin
  write('Введите два целых числа : ');
  readln(M1,M2);
  writeln('Расстояние между этим кодами = ',rasst(M1,M2));
  readln;
end.
```

Задание 2.16. Чтение числа справа налево

Составить функцию `invert(n)`, аргументом которой является длинное целое число. Возвращаемое значение должно быть равно числу, полученному из `n` перестановкой цифр в обратном порядке. Знак числа при этом сохраняется на прежнем месте. Например:

```
invert(314159) = 951413
invert(-2718) = -8172
```

Совет 1 (общий)

Очевидно, что самый простой алгоритм состоит в том, чтобы последовательно находить десятичные цифры, начиная с младшей, и тут же использовать их для вычисления перевернутого числа. Нужно только не забыть отделить от числа его знак, иначе каждый остаток от деления на 10 будет наследовать знак исходного числа.

Программа 2_16.bas

```
REM Перестановка старших и младших разрядов в числе
DECLARE FUNCTION invert!(N&)
CLS
INPUT "Введите целое число : ",M&
PRINT "Справа налево оно выглядит так ";invert(M&)
END

FUNCTION invert(N&)
Res&=0: sign=SGN(N&) : ' Учет знака числа
IF N&<0 THEN N&=-N&
DO
  k%=(N& MOD 10) : ' очередная цифра справа
  Res&=Res&*10+k% : ' формирование перевернутого результата
  N&=(N&-k%)/10 : ' удаление обработанной цифры
LOOP UNTIL N&=0
invert=Res&*sign : ' приклеили знак
END FUNCTION
```

Программа 2_16.c

```
/* Перестановка старших и младших разрядов в числе */
#include <stdlib.h>
long invert(long N);
```

```
  main()
{
  long M;
  printf("\nВведите целое число : ");
  scanf("%ld",&M);
  printf("\nСправа налево оно выглядит так  %ld",invert(M));
  getch();
}
long invert(long N)
{/* инвертирование числа */
  long Result=0, sign=1; /* Учет знака числа */
  if(N<0) { N=-N; sign=-1;}
  while (N!=0)
    {
     Result=Result*10 + (N % 10); /* формирование результата */
     N=N/10; /* удаление обработанной цифры */
      }
  return Result*sign; /* приклеили знак */
}
```

Программа 2._16.pas

```
program rotate;
{ Перестановка старших и младших разрядов в числе }
var
  N:longint;
function invert(N:longint):longint;
const
  Result:longint=0;
  sign:shortint=1;
begin
  if N<0 then { Учет знака числа }
    begin
      N:=-N;
      sign:=-1;
    end;
  while (N<>0) do
    begin
      Result:=Result*10+(N mod 10);{ формирование результата }
```

```
      N:=N div 10; { удаление обработанной цифры }
    end;
  invert:=Result*sign; { приклеили знак }
end;
begin
  write('Введите целое число : ');
  readln(N);
  writeln('Справа налево оно выглядит так : ',invert(N));
  readln;
end.
```

Задание 2.17. Числовые палиндромы

Натуральное число $N = a_1a_2...a_k$ называют палиндромом, если его величина совпадает со значением, прочитанным справа налево, $N1 = a_k...a_2a_1$. При этом предполагается, что $a_1 > 0$. Например, 1881 — палиндром, а 1812 — нет. Составить функцию `palindrom(n)`, аргументом которой является длинное положительное целое число. Функция должна возвращать значение `true` (Паскаль) или `1` (QBasic, Си), если ее аргумент является числовым палиндромом.

Совет 1 (общий)

Конечно, функцию `palindrom` можно построить на базе функции `invert`. Достаточно возвратить в качестве значения функции результат сравнения `N` и `invert(N)`. Но для разнообразия мы предлагаем вам другой алгоритм, в котором формируется массив десятичных цифр числа и производится сравнение его симметрично расположенных элементов.

Совет 2 (общий)

Более компактный вариант программы можно построить, заменив цикл формирования массива десятичных цифр стандартной функцией преобразования числа в его символьный эквивалент. Такого рода функции имеются в каждом алгоритмическом языке — `STR$` в QBasic, `ltoa` в Си, процедура `str` в Паскале. Один из примеров — программа `2_17a.pas` — демонстрирует указанный подход. Кроме того, в ее реализации показана уникальная особенность Паскаля, позволяющего в качестве индексов у элементов массива использовать нечисловые значения.

Совет 3 (QBasic)

В программе `2_17.bas` также реализован второй подход. Но здесь следует иметь в виду, что функция `STR$` выдает результат с пробелом в первой позиции для положительных чисел. Поэтому сравнение цифр начинается со второй позиции. Для выделения симметрично расположенных цифр использована функция `MID$`, второй аргумент которой определяет начальную позицию в строке, а третий — количество выделяемых символов.

Программа 2_17.bas

```
REM Является ли введенное число палиндромом?
DECLARE FUNCTION palindrom!(N&)
CLS
DIM noyes$(2)
noyes$(0)="не"
INPUT "Введите целое число: ",M&
PRINT "Это - "; noyes$(palindrom(M&));" палиндром"
END

FUNCTION palindrom(N&)
REM Если N - палиндром, то palindrom = 1
palindrom=1
IF N&<10 THEN EXIT FUNCTION : ' Одноразрядное - всегда палиндром
s$=STR$(N&) : ' Перевод числа в строку
k%=LEN(s$)
FOR j=2 TO 1+k%/2 : ' Цикл проверки симметрии цифр
 IF MID$(s$,j,1)<>MID$(s$,k%+2-j,1) THEN palindrom=0
NEXT j
END FUNCTION
```

Программа 2_17.c

```
/* Является ли введенное число палиндромом? */
#include <stdio.h>
#include <conio.h>
int palindrom(long N);
  main()
{
  long M;
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  if (palindrom(M)) printf("\nЭто - палиндром");
  else printf("\nЭто - не палиндром");
  getch();
}
int palindrom(long N)
{/* Если N - палиндром, то palindrom = 1 */
```

```
  int j,k=1;
  char digit[10];
  if(N<10) return 1; /* Одноразрядное - всегда палиндром */
  for(j=0; j<10; j++)
    { /* цикл выделения десятичных цифр */
      digit[j]=N%10;
      N=N/10; /* удаление обработанной цифры */
      if(N!=0) k++;
    }
  for(j=0; j<=k/2; j++) /* Цикл проверки симметрии цифр */
    if(digit[j]!=digit[k-j-1]) return 0;
  return 1;
}
```

Программа 2_17.pas

```
program Palindroms;
{ Является ли введенное число палиндромом ? }
var
  M: longint;
function
palindrom(N:longint):boolean;
{ Если N - палиндром, то palindrom = true }
var
  j,k:integer;
  digit:array [0..9] of byte;
begin
  palindrom:=true;
  if N<10 then exit; { Одноразрядное - всегда палиндром }
  k:=0;
  for j:=0 to 9 do { цикл выделения десятичных цифр }
  begin
    digit[j]:=N mod 10; { очередная цифра числа N }
    N:=N div 10; { удаление обработанной цифры }
    if(N<>0) then k:=k+1; { счетчик цифр в числе N }
  end;
  for j:=1 to (k div 2) do { Цикл проверки симметрии цифр }
    if digit[j]<>digit[k-j] then palindrom:=false;
end;
```

```
begin
  write('Введите целое число: ');
  readln(M);
  if palindrom(M) then
    writeln('Это - палиндром')
  else
    writeln('Это не палиндром');
  readln;
end.
```

Программа 2_17a.pas

```
program Palindroms;
{ Является ли введенное число палиндромом? }
var
  M: longint;
const
  noyes:array [false..true] of string=('не', '');
function palindrom(N:longint):boolean;
{ Если N - палиндром, то palindrom = true }
var
  j,k:byte;
  s:string[16];
begin
  palindrom:=true;
  if N<10 then exit; { Одноразрядное - всегда палиндром }
  str(N,s); { перевод числа в символьную строку }
  k:=length(s);
  for j:=1 to (k div 2) do { Цикл проверки симметрии цифр }
    if s[j]<>s[k+1-j] then palindrom:=false;
end;
begin
  write('Введите целое число: ');
  readln(M);
  writeln('Это - ',noyes[palindrom(M)],' палиндром');
  readln;
end.
```

Задание 2.18. Генерация палиндромов с заданным свойством

Составить программу, выводящую на экран все четырехзначные палиндромы, квадраты которых тоже являются палиндромами. Задачу можно несколько усложнить, если считать количество цифр в исходном палиндроме параметром `k` (`k < 5`).

Совет 1 (общий)

Вместо перебора всех чисел в диапазоне [1000, 9999] целесообразно организовать цикл по формированию четырехзначных палиндромов вида $a_1a_2a_2a_1$, который будет повторяться всего 90 раз.

Программа 2_18.bas

```
REM Генерация палиндромов, квадраты которых тоже палиндромы
DECLARE FUNCTION palindrom!(N&)
CLS
FOR A1=1 TO 9: FOR A2=0 TO 9
  M&=(((A1*10)+A2)*10+A2)*10+A1
  M2&=M&*M&
  IF palindrom(M2&) THEN PRINT M&,M2&
NEXT A2,A1
END

FUNCTION palindrom (N&)
REM Если N - палиндром, то palindrom = 1
palindrom=1
IF N&<10 THEN EXIT FUNCTION : ' Одноразрядное - всегда палиндром
s$=STR$(N&) : ' перевод числа в символьную строку
k%=LEN(s$)
FOR j=2 TO 1+k%/2 : ' Цикл проверки симметрии цифр
 IF MID$(s$,j,1)<>MID$(s$,k%+2-j,1) THEN palindrom=0
NEXT j
END FUNCTION
```

Программа 2_18.c

```
/* Генерация палиндромов, квадраты которых тоже палиндромы */
#include <stdio.h>
#include <conio.h>
```

```
int palindrom(long N);
  main()
{
  long M,M2;
  int a1,a2;
  for(a1=1; a1<10; a1++)
    for(a2=0; a2<10; a2++)
      {
        M=(((a1*10)+a2)*10+a2)*10+a1;
        M2=M*M;
        if(palindrom(M2))
           printf("\nчисло=%ld квадрат=%ld",M,M2);
      }
  getch();
}
int palindrom(long N)
{/* Если N - палиндром, то palindrom = 1 */
  int j,k=1;
  char digit[10];
  if(N<10) return 1;  /* Одноразрядное - всегда палиндром */
  for(j=0; j<10; j++) /* Цикл выделения цифр числа N */
    {
      digit[j]=N % 10;
      N=N/10;
      if(N!=0) k++;
    }
  for(j=0; j <= k/2; j++) /* Цикл проверки симметрии цифр */
      if(digit[j]!=digit[k-j-1]) return 0;
  return 1;
}
```

Программа 2_18.pas

```
program sqr_pal;
{ Генерация палиндромов, квадраты которых тоже палиндромы }
var
  M,M2:longint;
  a1,a2:integer;
function palindrom(N:longint):boolean;
{ Если N - палиндром, то palindrom = true }
```

```
var
  j,k:integer;
  digit:array [0..9] of byte;
begin
  k:=1;
  palindrom:=true;
  if N < 10 then exit; { Одноразрядное - всегда палиндром }
  for j:=0 to 9 do     { Цикл выделения цифр числа N }
    begin
      digit[j]:=N mod 10;
      N:=N div 10;
      if N<>0 then inc(k);
    end;
  for j:=0 to k div 2 do { Цикл проверки симметрии цифр }
      if digit[j]<>digit[k-j-1] then palindrom:=false;
end;
begin
  for a1:=1 to 9 do
    for a2:=0 to 9 do
      begin
        M:=(((a1*10)+a2)*10+a2)*10+a1;
        M2:=M*M;
        if palindrom(M2) then
           writeln('число=',M,' квадрат=',M2);
      end;
  readln;
end.
```

Задание 2.19. Числовые преобразования до обнаружения зацикливания

Составить программу, которая вводит длинное целое число N и подвергает его многократному преобразованию по следующей схеме:

$N_{i+1} = a(i,k)^3 + a(i,k-1)^3 + \ldots + a(i,0)^3$

Здесь через a(i,k), a(i,k-1),... , a(i,0) обозначены цифры числа N_i. Обработка прекращается при обнаружении зацикливания ($N_i = N_j$, $i > j$).

Совет 1 (общий)

Очень большое девятиразрядное число 1999999999 дает максимальную сумму кубов цифр, равную 6562. Поэтому один из возможных вариантов решения —

завести массив Q из 6562 элементов, предварительно обнулив все его элементы. Затем в процессе преобразования текущего числа N_i записывать в элемент $Q[N_i]$ индекс i, если значение этого элемента было равно 0. Цикл будет обнаружен, как только получится число N_j, для которого в элемент $Q[N_j]$ уже что-то заносилось. В связи с тем, что в используемых системах программирования глобальные числовые массивы перед запуском программы чистятся, можно отказаться от обнуления элементов массива Q.

Совет 2 (QBasic)

В предположении, что длина цикла не очень велика, можно пойти на запоминание всех промежуточных результатов в небольшом массиве MAS&. Для проверки зацикливания можно сравнивать очередное число со всеми ранее полученными.

Совет 3 (общий)

Попробуйте модифицировать программу таким образом, чтобы обнаружить все числовые последовательности, образующие цикл с самого первого шага. В качестве примера приведем некоторые из них:

1 → 1 (одношаговый цикл),
136 → 244 → 136 (двухшаговый цикл),
55 → 250 → 133 → (трехшаговый цикл).

Программа 2_19.bas

```
REM Поиск циклов при суммировании кубов цифр
DECLARE FUNCTION nkub& (N&)
  DIM MAS&(100) : ' Массив для последовательности чисел
  CLS
  INPUT "Введите число - ", N&
  K=0: MAS&(0)=N&
m1: K=K+1 : ' Счетчик числа шагов
  MAS&(K)=nkub&(MAS&(K-1))
  PRINT "Шаг =";K," N =";MAS&(K)
  FOR I=0 TO K-1 : ' Поиск совпадения с предыдущими числами
    IF MAS&(I)=MAS&(K) THEN GOTO m2
  NEXT I
  GOTO m1
m2: PRINT "Обнаружен цикл": PRINT "Начало цикла шаг ";I
  PRINT "Конец цикла шаг ";K-1
  END

FUNCTION nkub&(N&)
REM Преобразование числа в сумму кубов его цифр
```

```
  DIM S AS INTEGER, M AS LONG
  S=0: M=N&
  WHILE M<>0
    D=M MOD 10: M=(M-D)/10 : ' Выделение очередной цифры
    S=S+D*D*D : ' Накопление суммы кубов
  WEND
  nkub&=S
END FUNCTION
```

Программа 2_19.c

```
/* Поиск циклов при суммировании кубов цифр */
#include <stdio.h>
#include <conio.h>
long n_kub(long N);
  long N;
  int i=1,Q[6562];
  main()
{
  clrscr();
  printf("\nВведите число ");
  scanf("%ld",&N);
  Q[N-1]=i;
m1:
  printf("\nШаг=%2d N=%ld",i,N);
  N=n_kub(N);
  if(Q[N-1]!=0) /* Не было ли раньше такого же числа ?*/
    {
      printf("\nНа %d шаге обнаружен цикл N=%ld",i+1,N);
      getch();
      exit(0);
    }
  Q[N-1]=i++; /* Фиксация очередного числа в массиве Q*/
  goto m1;
}
/*----------------------------------------*/
long n_kub(long N)
{/* Преобразование числа в сумму кубов его цифр */
  int k;
```

```
  long s=0;
  while (N)
    {
      k=N%10;  N=N/10; /* Выделение очередной цифры */
      s+=k*k*k; /* Накопление суммы кубов */
    }
  return s;
}
```

Программа 2_19.pas

```
program loop_cub;
{ Поиск циклов при суммировании кубов цифр }
uses Crt;
label m1;
var
  N:longint;
  Q:array [1..6562] of integer;
  i:integer;
function n_kub(N:longint):integer;
{ Преобразование числа в сумму кубов его цифр }
var
   s,k:integer;
begin
   s:=0;
   while (N<>0) do
     begin
       k:=N mod 10;  N:=N div 10; { Выделение очередной цифры }
       s:=s+k*k*k; { Накопление суммы кубов }
     end;
   n_kub:=s;
end;

begin
  clrscr;
  write('Введите число');
  readln(N);
  for i:=1 to 6562 do Q[i]:=0; { Очистка массива Q}
  i:=1;
  Q[N]:=1;
```

```
m1:
  writeln('Шаг=',i:2,' N=',N);
  N:=n_kub(N);
  if Q[N]<>0 then { Не было ли раньше такого же числа ?}
    begin
      writeln('На шаге ',i+1:2,' обнаружен цикл');
      readln; halt;
    end;
  inc(i);  { Счетчик числа шагов }
  Q[N]:=i; { Запоминание счетчика в массиве Q }
  goto m1;
end.
```

Задание 2.20. Разложение числа на простые множители

Составить программу, которая выдает разложение заданного целого числа `N` на простые множители. Например:

```
128 = 2*2*2*2*2*2*2
17 = простое число
```

Совет 1 (общий)

Самый простой способ заключается в попытке делить `N` на числа `k = 2, 3, ...` до тех пор, пока `N` делится. Очевидно, что перебор делителей `k` не имеет смысл продолжать для `k > N/2`. Переменная `J` в программах используется как признак того, что был найден хотя бы один нормальный делитель. Нулевое значение `J` означает, что исходное число — простое.

Программа 2_20.bas

```
REM Разложение числа на простые множители
CLS
K& = 2: J% = 0
INPUT "Введите целое число: ", M&: M1& = M& / 2
PRINT M&; "=";
M1:
  IF M& MOD K&=0 THEN
     J=1: M&=M&/K&: PRINT K&;
     IF M&<>1 THEN PRINT "*";
  ELSE K&=K&+1
  END IF
```

```
  IF K&<=M1& THEN GOTO M1
  IF J=0 THEN PRINT " простое число"
END
```

Программа 2_20.c

```
/* Разложение числа на простые множители */
#include <stdio.h>
#include <conio.h>
  main()
{
  long M,M1,k=2;
  char j=1;

  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  M1=M/2;
  printf("\n%ld=",M);
m1:
  if(M % k==0)
    {
      j=0;
      M=M/k;
      printf("%ld",k);
      if(M!=1) printf("*");
     }
   else k++;
   if(k<=M1) goto m1;
   if(j==1) printf("простое число");
   getch();
}
```

Программа 2_20.pas

```
program razlojenie;
{ Разложение числа на простые множители }
var
  M,M1,k:longint;
  j:boolean;
```

```
label 1;
begin
  j:=true;
  k:=2;
  write('Введите целое число: ');
  readln(M);
  M1:=M div 2;
  write(M,'=');
1:if(M mod k)=0 then
    begin
      j:=false;
      M:=M div k;
      write(k);
      if M<>1 then write('*');
    end
  else k:=k+1;
  if k<=M1 then goto 1;
  if j then write('простое число');
  readln;
end.
```

Задание 2.21. Проверка числа на "простоту"

Составить функцию `prime(n)`, аргументом которой является длинное целое положительное число. Функция должна возвращать значение `true` (Паскаль) или `1` (Си, QBasic), если ее аргумент является простым числом. В противном случае функция должна возвращать значение `false` или `0`.

Совет 1 (общий)

Один из наиболее легко реализуемых алгоритмов проверки числа на "простоту" заключается в том, что исходное число `N` последовательно делят на `2`, `3`, `5`, `7`, `9`, ..., `2*p+1` (`(2*p+1)2 ≤ N)`. Если ни один из остатков от деления не равен нулю, то `N` — простое. Конечно, этот алгоритм далек от совершенства — например, зачем делить на 9 или 15, если число не делилось на 3. По идее, в проверке должны участвовать только простые делители — `2`, `3`, `5`, `7`, `11`, `13`, ... Но построить такую программу несколько сложнее (см. задание 2.22).

Программа 2_21.bas

```
DECLARE FUNCTION prime!(N&)
REM Проверка числа на простоту
```

```
CLS
INPUT "Введите целое число: ", M&
IF prime(M&)=1 THEN
  PRINT "Это число - простое "
ELSE
  PRINT "Это число - составное"
END IF
END

FUNCTION prime(N&)
REM Если N - простое, то prime = 1
DIM j AS LONG
IF N&<4 THEN GOTO M1 : ' Числа 2, 3 - простые
IF N& MOD 2=0 THEN GOTO M0 : ' Четные числа - составные
REM Проверка делимости на нечетные числа
FOR j=3 TO SQR(N&)+1 STEP 2
  IF N& MOD j=0 THEN GOTO M0
NEXT j
M1: prime=1: EXIT FUNCTION
M0: prime=0
END FUNCTION
```

Программа 2_21.c

```
/* Проверка числа на простоту */
#include <stdio.h>
#include <math.h>
int prime(long N);
  main()
{
  long M;
  char *a[]={"составное","простое"};
  printf("\nВведите целое число: ");
  scanf("%ld",&M);
  printf("\nЭто число - %s",a[prime(M)]);
  getch();
}
int prime(long N)
{ /* Если N - простое, то prime = 1 */
```

```
  long kmax,j;
  if(N<4) return 1; /* Числа 2, 3 - простые */
  if(N % 2==0) return 0; /* Четные числа - составные */
/* Проверка делимости на нечетные числа */
  for(j=3; j*j<=N; j+=2)
    if( N % j==0) return 0;
  return 1;
}
```

Программа 2_21.pas

```
program prostota;
{ Проверка числа на простоту }
var
  M:longint;
const
  a:array [0..1] of string=('составное','простое');
function prime(N:longint):byte;
{ Если N - простое, то prime = 1 }
var
  j:longint;
label m1;
begin
  prime:=1;
  if N<4 then exit; { Числа 2, 3 - простые }
  prime:=0;
  if N mod 2=0 then exit; { Четные числа - составные }
  j:=3;
{ Проверка делимости на нечетные числа }
m1: if N mod j=0 then exit;
  j:=j+2;
  if j*j<=N then goto m1;
  prime:=1;
end;
begin
  writeln('Введите целое число: ');
  readln(M);
  writeln('Это число - ',a[prime(M)]);
  readln;
end.
```

Задание 2.22. Решето Эратосфена

История сохранила память о древнегреческом математике Эратосфене Киренском, который применил оригинальный алгоритм нахождения всех простых чисел. На длинном папирусе он выписал числа 2, 3, 4, ... , 1000. Затем, сохранив первое число 2, он проколол каждое второе число, т. е. все четные числа, делящиеся на 2. В следующий раз, пропустив второе, не проколотое число 3, удалил после него каждое третье число, т. е. все числа, делящиеся на 3. Такая же участь постигла и каждое пятое число после 5, каждое седьмое число после 7 и т. д. К концу эксперимента на папирусе остались не проколотыми только простые числа. Возможно, что Эратосфен не догадался вдвое облегчить свою работу и использовать вдвое более короткий папирус, изначально выписав на нем только нечетные числа. Зато сейчас мы составим программу, которая использует идею Эратосфена и находит все простые числа среди первых `maxk` нечетных чисел.

Совет 1 (общий)

Сформируем массив `is`, `i`-й элемент которого будет представлять нечетное число `2*i+3` и равняться `1`, если `i`-е число еще не "проколото", и равняться `0` в противном случае. Действуем по алгоритму Эратосфена — принимаем 3 в качестве первого нечетного простого числа и присваиваем нулевые значения каждому третьему элементу `is`. В качестве второго нечетного простого числа принимается следующая "не проколотая" позиция, соответствующая 5. "Прокалываем" каждую пятую позицию в `is` и т. д. В Бейсике идентификатор `IS` является служебным словом, поэтому в программе его пришлось заменить на `IS1`.

Совет 2 (Паскаль)

Решето Эратосфена очень изящно выглядит с применением данных типа множество (`set`). Регулярное множество, содержащее отрезок чисел натурального ряда, легко формируется с помощью интервального набора данных. К множеству можно добавлять новые элементы (операция сложения), исключать ненужные (операция вычитания), проверять принадлежность данного элемента множеству (операция `in`). Единственный недостаток множеств в Паскале — ограничение на количество элементов (не более 255).

Программа 2_22.bas

```
DEFINT A-Z
CLS
MAXK=500
DIM IS1(MAXK+1)
FOR I=0 TO MAXK-1: IS1(I)=1: NEXT I
FOR I=0 TO MAXK
  IF IS1(I)=1 THEN
```

```
    'если число еще не "проколото"
    PRIME=I+I+3: ' сформировали очередное простое число
    N=N+1:   ' увеличили счетчик простых чисел
    PRINT USING "#####"; PRIME; : ' вывод очередного простого числа
    K=I+PRIME: ' индекс для первого "прокола"
    WHILE K<=MAXK
      REM  "проколы" чисел, делящихся на PRIME
      IS1(K)=0: K=K+PRIME
    WEND
  END IF
NEXT I
  PRINT
  PRINT "Среди первых ";MAXK+1; " нечетных чисел";
  PRINT " найдено "; N; " простых"
END
```

Программа 2_22.c

```
#include <stdio.h>
#include <conio.h>

#define maxk 500

  main()
{
  int pr_num,n=0;
  char is[maxk+1];
  register int i,k;
  clrscr();
  for(i=0; i<=maxk; i++) is[i]=1;  /* все числа "на папирусе" */
  for(i=0; i<=maxk; i++)
    {if(is[i]==1)      /* если число еще не "проколото" */
      { pr_num=i+i+3; /* сформировали очередное простое число */
        n++;          /* увеличили счетчик простых чисел */
     printf("%5d",pr_num); /* вывод очередного простого числа */
        k=i+pr_num;   /* индекс для первого "прокола" */
        while(k<=maxk)
          { /* "проколы" чисел, делящихся на pr_num */
            is[k]=0; k+=pr_num;
```

```
          }
      }
    }
  printf("\nСреди первых %d нечетных чисел", maxk+1);
  printf("найдено %d простых", n);
  getch();
}
```

Программа 2_22.pas

```
program sieve1;
uses Crt;
const
  maxk=500;
  n:integer=0;
var
  is: array [0..maxk] of byte;
  i,k,prime:integer;
begin
  clrscr;
  for i:=0 to maxk do is[i]:=1;   { все числа "на папирусе" }
  for i:=0 to maxk do
    begin
      if is[i]=1 then   { если число еще не "проколото" }
      begin
        prime:=i+i+3;   { сформировали очередное простое число }
        inc(n);         { увеличили счетчик простых чисел }
        write(prime:5); { вывод очередного простого числа }
        k:=i+prime;     { индекс для первого "прокола" }
        while k<=maxk do
          begin         { "проколы" чисел, делящихся на pr_num }
             is[k]:=0; inc(k,prime);
          end
      end;
    end;
  writeln;
  write(' Среди первых ',maxk+1,' нечетных чисел ');
  writeln(' найдено ',n,' простых');
  readln;
end.
```

Программа 2_22a.pas

```
program sieve;
uses Crt;
const
  maxN=255; {не более 255}
var
  primes: set of 2..maxN;
  i,j:integer;
begin
  clrscr;
  primes:=[2..maxN]; { первоначальная роспись папируса }
  for i:=2 to maxN do
    if i in primes then { если i принадлежит множеству }
    begin
      write(i:5); { вывод очередного простого числа }
      for j:=1 to maxN div i do
        primes:=primes-[i*j]; { удаление чисел, кратных i}
    end;
  readln;
end.
```

Задание 2.23. Разложение четного числа на сумму простых чисел

В 1742 г. Христиан Гольдбах высказал предположение, что любое четное число можно представить в виде суммы двух простых чисел, а любое нечетное — в виде суммы трех простых чисел. На самом деле, вторая часть утверждения Гольдбаха является следствием первой. Если из нечетного числа вычесть подходящее простое, то остаток будет четным, и как только его удастся разложить на сумму двух простых, первоначальное нечетное число будет равно сумме трех простых чисел. Несмотря на кажущуюся простоту проблемы Гольдбаха, до сих пор ее справедливость ни доказана, ни опровергнута. Более того, в начале 2000 г. английский книгоиздатель Фейбер предложил награду в размере 1 миллиона долларов тому, кто сумеет доказать или опровергнуть предположение Гольдбаха.

Предлагается провести численный эксперимент, который, к сожалению, не претендует на получение объявленной награды. Составим программу, которая находит все возможные разложения числа `n` на сумму двух простых чисел.

Совет 1 (общий)

Самый простой вариант поиска нужного разложения заключается в переборе сумм вида `k + (n - k)`. Если оба слагаемых оказались простыми, то искомое разложение найдено. Анализ слагаемых можно организовать с помощью функции `prime`, построенной в одном из предыдущих заданий. Естественно, что начать перебор можно с `k = 2`, а затем в качестве первого слагаемого пробовать все нечетные числа, не превосходящие `n/2`.

Программа 2_23.bas

```
DECLARE FUNCTION prime!(N&)
REM Разложение четного числа на сумму двух простых
CLS
INPUT "Введите четное число: ",M&
IF prime(M&-2)=1 THEN PRINT M&;"=2+";M&-2
FOR j&=1 TO M&/2 STEP 2
  IF prime(j&)=1 AND prime(M&-j&)=1 THEN
     PRINT M&;"=";j&;"+";M&-j&
   END IF
NEXT j&
END

FUNCTION prime(N&)
DIM j AS LONG
IF N&<4 THEN GOTO M1
IF N& MOD 2=0 THEN GOTO M0
FOR j=3 TO SQR(N&)+1 STEP 2
  IF N& MOD j=0 THEN GOTO M0
NEXT j
M1: prime=1: EXIT FUNCTION
M0: prime=0
END FUNCTION
```

Программа 2_23.c

```
/* Разложение четного числа на сумму двух простых */
#include <stdio.h>
#include <math.h>
int prime(long N);
  main()
```

```
{
  long M,j;
  clrscr();
  printf("\nВведите четное число: ");
  scanf("%ld",&M);
  if(prime(M-2))printf("%ld=  2+%ld\t",M,M-2);
  for(j=1; j <= M/2; j+=2)
    if(prime(j) && prime(M-j)) printf("%ld=%3ld+%ld\t",
                                      M, j, M-j);
  getch();
}
int prime(long N)
{
  long kmax,j;

  if(N<4) return 1;
  if(N%2==0) return 0;
  kmax=sqrt(N)+0.5;
  for(j=3; j<=kmax; j+=2)
    if(N%j==0) return 0;
  return 1;
}
```

Программа 2_23.pas

```
program razlojenie;
{ Разложение четного числа на сумму двух простых }
var
  M:longint;
  j:integer;
label m2;
function prime(N:longint):byte;
var
  j:longint;
label m1;
begin
  prime:=1;
  if N<4 then exit;
  prime:=0;
```

```
  if N mod 2=0 then exit;
  j:=3;
m1: if N mod j=0 then exit;
  j:=j+2;
  if j*j<=N then goto m1;
  prime:=1;
end;
begin
  write('Введите четное число: ');
  readln(M);
  if prime(M-2)=1 then writeln(M,'+',M-2);
  j:=3;
m2:
  if (prime(j)=1) and (prime(M-j)=1) then
    writeln(M,'=',j,'+',M-j);
  j:=j+2;
  if j< M div 2 then goto m2;
  readln;
end.
```

Задание 2.24. Генерация чисел Хэмминга

Числами Хэмминга называются натуральные числа, которые среди своих делителей имеют только степени чисел 2, 3 и 5. Первые 10 упорядоченных по возрастанию чисел Хэмминга образует последовательность 1, 2, 3, 4, 5, 6, 8, 9, 10 и 12. Первому числу этого ряда соответствуют нулевые степени всех сомножителей. Составить программу, которая генерирует первые 1000 чисел Хэмминга.

Совет 1 (общий)

Конечно, можно построить цикл по перебору всех длинных целых чисел (а их — порядка двух миллиардов), в теле которого выделяются все делители, кратные 2, 3 и 5. Если результат деления тестируемого числа оказывается равным 1, т. е. у числа никаких других делителей нет, то найдено очередное число Хэмминга. Однако такая программа будет очень долго работать (для сравнения мы приведем и такой вариант решения).

Гораздо более эффективный алгоритм базируется на определении следующего числа Хэмминга по уже построенной последовательности. При этом приходится запоминать, какое из ранее найденных чисел необходимо домножить на 2, какое — на 3 и какое — на 5. Из трех возможных вариантов следует выбирать минимальное произведение. На первом шаге мы знаем единственное число Хэмминга, равное 1, и именно оно может быть умножено на следующем шаге на один из трех возмож-

ных множителей. Минимальное произведение равно 2, и это — второе число Хэмминга. Теперь умножению на 2 должно быть подвергнуто второе число, а умножению на 3 и 5 — пока еще первое. Поэтому на втором шаге минимальное произведение равно 3. После этого умножению на 2 и 3 может быть подвергнуто второе число, а на 5 — пока еще только первое. Одним словом, когда определено минимальное произведение, использованный множитель должен "переместиться" на следующее, уже найденное число Хэмминга.

Совет 2 (общий)

Обозначим через `k1`, `k2`, `k3` индексы чисел Хэмминга, которые будут множиться соответственно на 2, 3 и 5. Их начальные значения в программе на Бейсике можно не задавать, т. к. все числовые переменные перед началом работы программы сбрасываются в ноль.

Программа 2_24.bas (оптимальный вариант)

```
DEFLNG A-Z
DIM Xam(1000)
CLS
Xam(0)=1
PRINT 1,1
FOR J=1 TO 999
  x2=Xam(k2)*2
  x3=Xam(k3)*3
  x5=Xam(k5)*5
  IF x2<=x3 AND x2<=x5 THEN Xam(J)=x2: k2=k2+1
  IF x3<=x2 AND x3<=x5 THEN Xam(J)=x3: k3=k3+1
  IF x5<=x2 AND x5<=x3 THEN Xam(J)=x5: k5=k5+1
  PRINT USING "###:##########  ";J,Xam(J);
  REM Приостанов после заполнения экрана
  IF J MOD 20=0 THEN INPUT A$
NEXT J
END
```

Программа 2_24.c (оптимальный вариант)

```
#include <stdio.h>
#include <conio.h>
  main()
{
  long Xam[1000], x2, x3, x5;
  int j;
```

```
  int k2=0,k3=0,k5=0;
  clrscr();
  Xam[0]=1;
  printf("%4d %9d",1,1);
  for(j=1; j<1000; j++)
    {
      x2=Xam[k2]*2;
      x3=Xam[k3]*3;
      x5=Xam[k5]*5;
      if(x2<=x3 && x2<=x5)
        {Xam[j]=x2; k2++;}
      if(x3<=x2 && x3<=x5)
        {Xam[j]=x3; k3++;}
      if(x5<=x2 && x5<=x3)
        {Xam[j]=x5; k5++;}
      printf("\n%4d %9ld",j,Xam[j]);
      if(j % 20==0) getch();
    }
  getch();
}
```

Программа 2_24.pas (оптимальный вариант)

```
program Hamming;
uses crt;
var
  Xam:array[1..1000] of longint;
  x2,x3,x5 : longint;
  j:integer;
const
  k2:integer=1;
  k3:integer=1;
  k5:integer=1;
begin
  Xam[1]:=1;
  writeln(1:4,' ',1:10);
for j:=2 to 1000  do
  begin
    x2:=Xam[k2]*2;
```

```
    x3:=Xam[k3]*3;
    x5:=Xam[k5]*5;
    if(x2<=x3) and (x2<=x5) then
      begin Xam[j]:=x2; inc(k2); end;
    if(x3<=x2) and (x3<=x5) then
       begin Xam[j] := x3; inc(k3); end;
    if (x5 <= x2) and (x5 <= x3) then
      begin Xam[j] := x5; inc(k5); end;
    writeln(j:4,' ',Xam[j]:10);
    if(j mod 20)=0 then readln;
  end;
  readln;
end.
```

Программа 2_24a.pas (полный перебор)

```
program Hamming1;
uses crt,WinDos;
var
  j,Hour,Hour1,min,min1,sec,sec1,hsec,hsec1 : word;
  i,k : longint;
begin
  clrscr;
  gettime(Hour,min,sec,hsec);
  i:=1;
  j:=0;
  repeat
    k:=i;
    while (k mod 2)=0 do k:=k div 2;
    while (k mod 3)=0 do k:=k div 3;
    while (k mod 5)=0 do k:=k div 5;
    if k=1 then
      begin
        { write(i:10); } {отключение вывода }
        inc(j);
      end;
    inc(i);
  until j>=1000;
  gettime(Hour1,min1,sec1,hsec1);
```

```
  writeln;
  writeln(Hour,':',min,':',sec,'.',hsec);
  writeln(Hour1,':',min1,':',sec1,'.',hsec1);
  readln;
end.
```

Задание 2.25. Генерация неправильно сокращаемых дробей

Существует очень мало правильных дробей вида `m/n`, которые можно привести к несократимой дроби "незаконным" образом — зачеркнув одинаковые цифры в числителе и знаменателе. Например:

`26/65 = 2/5.`

Построить программу, которая использует заданное ограничение (`n < 100`) и выводит все дроби, обладающие указанным выше свойством. Тривиальные дроби типа 10/20, 10/30, ... , 20/30, ... , 80/90, ... желательно не генерировать.

Совет 1 (общий)

Очевидно, что единственным вариантом является полный перебор правильных дробей от 10/11 до 98/99, в процессе которого надо сравнивать исходную дробь с четырьмя возможными отношениями старших и младших цифр. Так как программа оперирует с вещественными числами, необходимо избегать проверок на точное совпадение. Кроме того, следует обходить деление на ноль.

Программа 2_25.bas

```
REM  Генерация неправильно сокращаемых дробей
CLS
REM Двойной цикл по перебору дробей от 10/11 до 98/99
FOR N=11 TO 99
  FOR M=10 TO N-1
    T=M/N :           ' Настоящее значение дроби
    Nlo=N MOD 10   : ' Младшая цифра знаменателя
    Nhi=(N-Nlo)\10 : ' Старшая цифра знаменателя
    Mlo=M MOD 10   : ' Младшая цифра числителя
    Mhi=(M-Mlo)\10 : ' Старшая цифра числителя
    IF Mlo=0 THEN GOTO 100
    REM Анализ различных сочетаний "зачеркиваемых" цифр
    IF Nlo=Mlo AND ABS(T-Mhi/Nhi)<.001 THEN
       PRINT M;"/";N;"=";Mhi;"/";Nhi
    END IF
```

```
    IF Nlo=Mhi AND ABS(T-Mlo/Nhi)<.001 THEN
       PRINT M;"/";N;"=";Mlo;"/";Nhi
    END IF
    IF Nlo<>0 THEN
      IF Nhi=Mlo AND ABS(T-Mhi/Nlo)<.001 THEN
         PRINT M;"/";N;"=";Mhi;"/";Nlo
      END IF
      IF Nhi=Mhi AND ABS(T-Mlo/Nlo)<.001 THEN
         PRINT M;"/";N;"=";Mlo;"/";Nlo
      END IF
    END IF
100 :
  NEXT M
NEXT N
END
```

Программа 2_25.c

```
/* Генерация неправильно сокращаемых дробей */
#include <stdio.h>
#include <conio.h>
#include <math.h>
  main()
{
  char n,m,n_lo,n_hi,m_lo,m_hi;
  float t;
  clrscr();
  /* Двойной цикл по перебору дробей от 10/11 до 98/99 */
  for(n=11; n<100; n++)
    for(m=10; m<n; m++)
      {
        t=(float)m/n; /* Настоящее значение дроби */
        n_lo=n % 10;  /* Младшая цифра знаменателя */
        n_hi=n/10;    /* Старшая цифра знаменателя */
        m_lo=m % 10;  /* Младшая цифра числителя */
        m_hi=m/10;    /* Старшая цифра числителя */
        if(m % 10 == 0) continue;
        /* Анализ различных сочетаний "зачеркиваемых" цифр */
        if(n_lo==m_lo && n_hi!=0 && fabs(t-m_hi/n_hi)<1e-3)
```

```
            printf("\n%d/%d=%d/%d",m,n,m_hi,n_hi);
         if(n_lo==m_hi && n_hi!=0 && fabs(t-m_lo/n_hi)<1e-3)
            printf("\n%d/%d=%d/%d",m,n,m_lo,n_hi);
         if(n_hi==m_lo && n_lo!=0 && fabs(t-m_hi/n_lo)<1e-3)
            printf("\n%d/%d=%d/%d",m,n,m_hi,n_lo);
         if(n_hi==m_hi && n_lo!=0 && fabs(t-m_lo/n_lo)<1e-3)
           printf("\n%d/%d=%d/%d",m,n,m_lo,n_lo);
      }
  getch();
}
```

Программа 2_25.pas

```
program drobi;
{Генерация неправильно сокращаемых дробей}
uses crt;
var
  n,m,n_lo,n_hi,m_lo,m_hi:byte;
  t:real;
begin
  clrscr;
{ Двойной цикл по перебору дробей от 10/11 до 98/99 }
  for n:=11 to 99 do
    for m:=10 to n-1 do
      begin
        t:=m/n; { Настоящее значение дроби }
        n_lo:=n mod 10; { Младшая цифра знаменателя }
        n_hi:=n div 10; { Старшая цифра знаменателя }
        m_lo:=m mod 10; { Младшая цифра числителя }
        m_hi:=m div 10; { Старшая цифра числителя }
        if m_lo=0 then continue;
        { Анализ различных сочетаний "зачеркиваемых" цифр }
        if (n_lo=m_lo) and (n_hi<>0) and (abs(t-m_hi/n_hi)<1e-3) then
            writeln(m:3,'/',n:2,'=',m_hi:1,'/',n_hi);
        if (n_lo=m_hi) and (n_hi<>0) and (abs(t-m_lo/n_hi)<1e-3) then
            writeln(m:3,'/',n:2,'=',m_lo:1,'/',n_hi);
        if (n_hi=m_lo) and (n_lo<>0) and (abs(t-m_hi/n_lo)<1e-3) then
            writeln(m:3,'/',n:2,'=',m_hi:1,'/',n_lo);
        if (n_hi=m_hi) and (n_lo<>0) and (abs(t-m_lo/n_lo)<1e-3) then
```

```
            writeln(m:3,'/',n:2,'=',m_lo:1,'/',n_lo);
      end;
  readln;
end.
```

Задание 2.26. Разложение натурального числа на сумму квадратов

Утверждение, принадлежащее Лагранжу, гласит, что любое натуральное число N можно представить в виде суммы не более чем четырех квадратов целых чисел ($N = a^2 + b^2 + c^2 + d^2$). Составить программу, которая вводит длинное целое число N и находит хотя бы одно такое разложение. О неединственности результата свидетельствует пример: $4 = 2^2 = 1^2 + 1^2 + 1^2 + 1^2$. Кроме того, существует довольно много прямоугольных треугольников с целочисленными сторонами, которые позволяют заменить сумму квадратов катетов квадратом гипотенузы, а сумму четырех квадратов, следовательно, — суммой трех.

Совет 1 (общий)

Скорее всего, для решения этой задачи придется пойти на полный перебор. Оценку сверху на количество циклов, которое может потребоваться, легко получить, упорядочив слагаемые по величине:

$$a \le b \le c \le d \le \mathrm{sqrt}(N).$$

Совет 2 (общий)

Решение может быть найдено и меньше чем за N^2 циклов, если проанализировать четность N и в цикле шагать не через 1, а через 2. Так, например, если N нечетно, то среди искомых слагаемых имеется либо одно, либо три нечетных числа. Если N четно, то искомое разложение может состоять либо из четырех четных, либо из четырех нечетных, либо из двух четных и двух нечетных слагаемых. Цикл перебора можно оформить в виде подпрограммы, которой передаются начальные значения управляющих переменных. Подпрограмма PROBA получает 5 параметров, первые 4 из которых задают четность (параметр = 1) или нечетность (параметр = 0) подбираемых слагаемых a, b, c и d.

Программа 2_26.bas

```
REM Разложение числа на сумму квадратов
DECLARE SUB PROBA (A1 AS INTEGER,B1 AS INTEGER,C1 AS INTEGER, D1 AS
INTEGER,N AS LONG)
INPUT "Введите натуральное число - ", N&
IF (N& MOD 2)=0 THEN
  REM Если число N - четное
  PROBA 0,0,1,1,N&
```

```
  PROBA 0,0,0,0,N&
  PROBA 1,1,1,1,N&
ELSE
  REM Если число N - нечетное
  PROBA 0,0,0,1,N&
  PROBA 0,1,1,1,N&
END IF
END

SUB PROBA (A1 AS INTEGER,B1 AS INTEGER,C1 AS INTEGER,D1 AS INTEGER,N AS
LONG)
'Подпрограмма перебора вариантов
Q=INT(SQR(N)+.5) : ' Определение верхней границы
FOR A%=A1 TO Q STEP 2: S1=A%*A%
FOR B%=B1 TO Q STEP 2: S2=S1+B%*B%
FOR C%=C1 TO Q STEP 2: S3=S2+C%*C%
FOR D%=D1 TO Q STEP 2
  IF S3+D%*D%=N THEN
    PRINT "Искомые слагаемые :"
    PRINT "a=";A%,"b=";B%,"c=";C%,"d=";D%
    END
  END IF
NEXT D%: NEXT C%: NEXT B%: NEXT A%
END SUB
```

Программа 2_26.c

```
/* Разложение числа на сумму квадратов */
#include <math.h>
#include <stdlib.h>
#include <conio.h>
void proba(int a1,int b1,int c1,int d1,long N);
  main()
{
  long N;
  printf("\nВведите натуральное число - ");
  scanf("%ld",&N);
  if((N % 2)==0) /* Если число N - четное */
    { proba(0,0,1,1,N);
```

```
      proba(0,0,0,0,N);
      proba(1,1,1,1,N);  }
  else /* Если число N - нечетное */
    { proba(0,0,0,1,N);
      proba(0,1,1,1,N);  }
}

void proba(int a1,int b1,int c1,int d1,long N)
/* Подпрограмма перебора вариантов */
{ int a,b,c,d,q;
  long s1,s2,s3;
  q=floor(sqrt(N)+.5); /* Определение верхней границы */
  for(a=a1; a<=q; a+=2) { s1=a*a;
    for(b=b1; b<=q; b+=2) { s2=s1+b*b;
      for(c=c1; c<=q; c+=2) { s3=s2+c*c;
        for(d=d1; d<=q; d+=2)
          if(s3+d*d==N)
            { printf("\nИскомые слагаемые :");
              printf("\na=%d b=%d c=%d d=%d",a,b,c,d);
              getch(); exit(0);
      } } } }
  return;
}
```

Программа 2_26.pas

```
program razlojenie;
{ Разложение числа на сумму квадратов }
var
  N:longint;
procedure proba(a1,b1,c1,d1:integer; N:longint);
{ Подпрограмма перебора вариантов }
  var
    a,b,c,d,q:integer;
    s1,s2,s3:longint;
  begin
    q:=round(sqrt(N)); { Определение верхней границы }
    a:=a1;
    while a<=q do begin
```

```
      s1:=a*a; b:=b1;
      while b<=q do begin
        s2:=s1+b*b; c:=c1;
        while c<=q do begin
          s3:=s2+c*c; d:=d1;
          while d<=q do begin
            if(s3+d*d=N) then begin
            writeln('Искомые слагаемые :');
            writeln('a=',a,' b=',b,' c=',c,' d=',d);
            readln; exit; end;
          inc(d,2); end;
        inc(c,2);end;
      inc(b,2);end;
    inc(a,2);end;
  end;
begin
  write('Введите натуральное число - ');
  readln(N);
  if odd(N) then begin  { Если число N - нечетное }
    proba(0,0,0,1,N);
    proba(0,1,1,1,N);  end
  else begin  { Если число N - четное }
    proba(0,0,1,1,N);
    proba(0,0,0,0,N);
    proba(1,1,1,1,N); end;
end.
```

Задание 2.27. Анализ взаимного расположения точки и треугольника

Составить программу, которая определяет, лежит ли точка с заданными координатами внутри или вне треугольника. В реальности возможны три ситуации: либо точка принадлежит хотя бы одной из сторон (то есть находится на границе треугольника), либо она расположена строго внутри, либо — строго снаружи.

Совет 1 (общий)

Воспользуемся известным фактом — любая прямая (`A*x+B*y+C=0`) делит плоскость на две полуплоскости. Все точки, принадлежащие одной полуплоскости, при подстановке в уравнение прямой дают значение `A*x+B*y+C` одного знака, а все точки другой полуплоскости — значение противоположного знака.

Поэтому, если точка P находится строго внутри треугольника ABC, то точки A и P находятся в одной полуплоскости по отношению к прямой BC, точки B и P — в одной полуплоскости по отношению к прямой AC, точки C и P — в одной полуплоскости по отношению к прямой AB.

Программа 2_27.bas

```
REM Анализ взаимного расположения точки и треугольника
DECLARE FUNCTION R!(U!,V!,U1!,V1!,U2!,V2!)
PRINT "Введи координаты вершин треугольника"
INPUT X1,Y1,X2,Y2,X3,Y3
'X1 = 0: Y1 = 0: X2 = 0: Y2 = 1: X3 = 1: Y3 = 0
PRINT "Введи координаты точки"
INPUT X, Y
IF (R(X,Y,X1,Y1,X2,Y2)*R(X3,Y3,X1,Y1,X2,Y2)>0
  AND R(X,Y,X2,Y2,X3,Y3)*R(X1,Y1,X2,Y2,X3,Y3)>0
  AND R(X,Y,X1,Y1,X3,Y3)*R(X2,Y2,X1,Y1,X3 Y3)>0) THEN
  PRINT "Точка находится внутри треугольника": END
END IF
IF (R(X,Y,X1,Y1,X2,Y2)*R(X3,Y3,X1,Y1,X2,Y2)<0
  OR R(X,Y,X2,Y2,X3,Y3)*R(X1,Y1,X2,Y2,X3,Y3)<0
  OR R(X,Y,X1,Y1,X3,Y3)*R(X2,Y2,X1,Y1,X3,Y3)<0) THEN
  PRINT "Точка находится вне треугольника": END
END IF
PRINT "Точка принадлежит границе"
END

FUNCTION R(U,V,U1,V1,U2,V2)
REM Анализ взаимного расположения точки (U,V) и прямой,
REM проходящей через точки (U1,V1) и (U2,V2)
R=0
IF ((U-U1)*(V2-V1)-(V-V1)*(U2-U1)>0) THEN R=1
IF ((U-U1)*(V2-V1)-(V-V1)*(U2-U1)<0) THEN R=-1
END FUNCTION
```

Программа 2_27.c

```
/* Анализ взаимного расположения точки и треугольника */
#include <stdio.h>
#include <conio.h>
```

```
int r(float u,float v,float u1,float v1, float u2,float v2);
  main()
{
  float x0,y0,x1,y1,x2,y2,x3,y3;
  float k012,k312,k023,k123,k013,k213;
  printf("\nВведи координаты вершин треугольника\n");
  scanf("%f %f %f %f %f %f",&x1,&y1,&x2,&y2,&x3,&y3);
  printf("\nВведи координаты точки\n");
  scanf("%f %f",&x0,&y0);
  k012=r(x0,y0,x1,y1,x2,y2);
  k312=r(x3,y3,x1,y1,x2,y2);
  k023=r(x0,y0,x2,y2,x3,y3);
  k123=r(x1,y1,x2,y2,x3,y3);
  k013=r(x0,y0,x1,y1,x3,y3);
  k213=r(x2,y2,x1,y1,x3,y3);
  if(k012*k312>0 && k023*k123>0 && k013*k213>0)
    {
      puts("Точка находится внутри треугольника");
      getch(); exit(0);
    }
  if(k012*k312<0 || k023*k123<0 || k013*k213<0)
    {
      puts("Точка находится вне треугольника");
      getch();  exit(0);
    }
  puts("Точка находится на границе");
  getch();
}
/*-------------------------------------------------------*/
int r(float u,float v,float u1,float v1, float u2,float v2)
/* Анализ взаимного расположения точки (U,V) и прямой,
   проходящей через точки (U1,V1) и (U2,V2) */
{
  float s=(u-u1)*(v2-v1)-(v-v1)*(u2-u1);
  if(s>0) return 1;
  if(s<0) return -1;
  return 0;
}
```

Программа 2_27.pas

```
program triangle;
{ Анализ взаимного расположения точки и треугольника }
var
  x0,y0,x1,y1,x2,y2,x3,y3:real;
  k012,k312,k023,k123,k013,k213:integer;
function r(u,v,u1,v1,u2,v2:real):integer;
{ Анализ взаимного расположения точки (U,V) и прямой,
  проходящей через точки (U1,V1) и (U2,V2) }
var
  s:real;
begin
  s:=(u-u1)*(v2-v1)-(v-v1)*(u2-u1);
  r:=0; if s>0 then r:=1;
  if s<0 then r:=-1;
end;
begin
  writeln('Введи координаты вершин треугольника');
  readln(x1,y1,x2,y2,x3,y3);
  writeln('Введи координаты точки');
  readln(x,y);
  k012:=r(x0,y0,x1,y1,x2,y2);
  k312:=r(x3,y3,x1,y1,x2,y2);
  k023:=r(x0,y0,x2,y2,x3,y3);
  k123:=r(x1,y1,x2,y2,x3,y3);
  k013:=r(x0,y0,x1,y1,x3,y3);
  k213:=r(x2,y2,x1,y1,x3,y3);
  if (k012*k312>0) and (k023*k123>0) and (k013*k213>0) then
    begin
      write('Точка находится внутри треугольника');
      halt;
    end;
  if (k012*k312<0) or (k023*k123<0) or (k013*k213<0) then
    begin
      write('Точка находится вне треугольника');
      halt;
    end;
```

```
  write('Точка находится на границе');
  readln;
end.
```

Задание 2.28. Игра на вычитание

Игра заключается в последовательном вычитании партнерами из числа, находящегося на кону. Программа генерирует случайное число от 50 до 100 и передает ход человеку. Ходят по очереди. За один ход можно вычесть из оставшегося числа от 1 до 9 (в общем случае от 1 до `N`). Побеждает тот, кто сумеет оставить на кону нулевое число.

Совет 1 (общий)

Если на кону находится число `N+1`, то любой ход приводит к выигрышу партнера. Поэтому выигрышная стратегия программы довольно проста. Если на кону находится число `M > N+1`, то своим ходом программа должна оставить на кону `M1 = k*(N+1)`, т. е. вычесть из `M` остаток от деления `M` на `N+1`. Это не всегда удается, т. к. остаток может оказаться равным 0. Но если это удалось, то последующие ходы делаются по стандартной схеме: если противник вычитает `q`, то программа должна вычесть `N+1-q`. В итоге на кону останется `N+1` и любой ход человека приведет к его поражению. В том случае, когда программа не может сделать выигрышный ход, надо вычесть 1 в надежде, что противник ошибется, и тогда можно будет перехватить инициативу.

Программа 2_28.bas

```
REM Программа игры на вычитание до нуля
DEFINT A-Z
m1:
  CLS
  RANDOMIZE 32767
  N = INT(RND * 25) + 26
  PRINT "На кону - "; N; " Брать можно от 1 до 5"
m2:
  PRINT "Ваш ход. Сколько берем? - "
m4: a$ = INKEY$: IF a$ = "" THEN GOTO m4
  k = ASC(a$) - 48
  IF 1 > k OR k > 5 THEN PRINT "Так ходить нельзя!": GOTO m2
  N = N - k
  PRINT "После Вашего хода осталось "; N
  IF N = 0 THEN PRINT "Поздравляю! Вы выиграли!": GOTO ex
  IF (N MOD 6) = 0 THEN k = 1 ELSE k = N MOD 6
```

```
  N = N - k
  PRINT "Я беру "; k; " остается "; N
  IF N = 0 THEN
    PRINT "К сожалению, Вы проиграли!": GOTO ex
  ELSE
    GOTO m2
  END IF
ex:
  PRINT "Хотите еще? - (y/n) : "
m3: a$ = INKEY$: IF a$ = "" THEN GOTO m3
  IF a$ = "y" THEN GOTO m1
  END
```

Программа 2_28.c

```
//Программа игры на вычитание до нуля
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
  void main()
{ int N,k,j;
m1:
  clrscr();
  randomize();
  N=random(25)+26;
  printf("На кону - %d. Брать можно от 1 до 5.\n",N);
m2:
  printf("\nВаш ход. Сколько берем? - ");
  k=getche()-48;
  if(1>k || k>5) {printf("\nТак ходить нельзя!"); goto m2;}
  N=N-k;
  printf("\nПосле Вашего хода осталось %d",N);
  if(N==0){printf("\nПоздравляю! Вы выиграли!"); goto ex;}
  k=(N%6==0)?1:N%6;
  N=N-k;
  printf("\nЯ беру %d остается %d",k,N);
  if(N==0){printf("\nК сожалению, Вы проиграли!"); goto ex;}
  goto m2;
```

```
ex:
  printf("\nХотите еще? - (y/n) : ");
  if(getch()=='y') goto m1;
}
```

Программа 2_28.pas

```
{Программа игры на вычитание до нуля}
program game_num;
uses Crt;
var
  N,k,j:integer;
label
  m1,m2,ex;
begin
m1:
  clrscr;
  randomize;
  N:=random(25)+26;
  writeln('На кону - ',N,'. Брать можно от 1 до 5.');
m2:
  writeln('Ваш ход. Сколько берем? - ');
  k:=Ord(ReadKey)-48;
  if(1>k) or (k>5) then
    begin writeln('Так ходить нельзя!'); goto m2; end;
  N:=N-k;
  writeln('После Вашего хода осталось ',N);
  if N=0 then
     begin writeln('Поздравляю! Вы выиграли!'); goto ex; end;
  if (N mod 6)=0 then k:=1 else k:=N mod 6;
  N:=N-k;
  writeln('Я беру ',k,' остается ',N);
  if N=0 then
     begin writeln('К сожалению, Вы проиграли!'); goto ex; end;
  goto m2;
ex:
  writeln('Хотите еще? - (y/n) : ');
  if ReadKey ='y' then goto m1;
end.
```

Задания 2.29—2.31. Путешествия по узлам целочисленной решетки

Существует довольно много задач, **связанных** с нумерацией узлов целочисленной решетки. Одна из них известна под названием "скатерти Улама" (Ulam S. M.). Развлекаясь на одном из скучных собраний, Улам пронумеровал по спирали клетки тетрадного листа. Когда он обвел номера клеток, соответствующие простым числам, на листе возник довольно занятный узор, который мог оказаться полезным для изучения закономерностей в числовых последовательностях.

В отличие от Улама мы будем нумеровать не клеточки, а узлы целочисленной решетки, начав эту процедуру с нулевого узла в начале системы координат (рис. 2.1).

Рис. 2.1. Нумерация точек на скатерти Улама

Одна из наиболее естественных задач заключается в определении координат точки по ее номеру `N`. Конечно, ее решение можно построить на базе прямого перебора всех точек спирали до узла с заданным номером. При этом движение, начинающееся из точки с номером `0`, состоит сначала из одного шага вправо (`x=x+1`) и вверх (`y=y+1`), затем из двух последовательных шагов влево (`x=x-1`) и вниз (`y=y-1`), из трех последовательных шагов вправо и вверх, из четырех последовательных шагов влево и вниз и т. д. Однако при достаточно больших значениях `N` такая процедура требует слишком много времени.

Гораздо интереснее построить более эффективный алгоритм решения задачи. Одна из идей заключается в определении координат угловой точки, расположенной на общей с нашей точкой горизонтальной или вертикальной ветви спирали. Заметим, что на луче, проведенном из начала координат через узлы с координатами `(-n,n)`, располагаются точки с номерами $(2*n)^2$.

На аналогичном луче, начинающемся из первой точки и проходящем через точки с координатами (n,-n+1), расположены точки с номерами $(2*n-1)^2$. Поэтому нам достаточно обнаружить ближайший к N квадрат числа q, чтобы сообразить, на какой из четырех ветвей "спирали" находится точка с номером N. Если q — четное, то в зависимости от разности q^2-N интересующая нас точка находится на верхней или левой ветви. И для вычисления координат достаточно откорректировать одну из координат угловой точки (-q,q) на величину разности. Если q оказалось нечетным, то ветвь, содержащая точку с номером N, находится либо внизу, либо справа. И тогда коррекции надо подвергнуть одну из координат точки (q,-q+1).

Аналогичная идея может быть использована и для обратного преобразования координат заданной точки (x,y) в ее порядковый номер на спирали. Достаточно определить ближайшую угловую точку с координатами (-q,q) или (q,-q+1) и произвести соответствующую коррекцию номера выбранной угловой точки. Несмотря на кажущуюся простоту идеи, ее реализация требует тщательной проверки многочисленных условий.

Обратим внимание на то, что диагональ y=x делит узлы нашей решетки на два множества. Для всех узлов, расположенных слева от диагонали, выполняется условие $x \geq y$. Пусть точка с координатами (x,y) расположена либо на горизонтальном витке спирали, находящемся в левой полуплоскости, либо на вертикальном витке, спускающемся до диагонали. Разобьем каждый из этих витков на две части и пронумеруем их следующим образом:

- 1 — горизонтальный участок от диагонали до оси y;
- 2 — горизонтальный участок от оси y до угла поворота;
- 3 — вертикальный участок от угла поворота до оси x;
- 4 — вертикальный участок от оси x до очередного поворота.

На каждом из этих участков справедливы следующие соотношения:

$1 : x \geq 0,\ y > 0$

$2 : x < 0,\ y > 0,\ y \geq |x|$

$3 : x < 0,\ y \geq 0,\ y \leq |x|$

$4 : x < 0,\ y \leq 0$

Определив принадлежность точки тому или иному участку, мы можем вычислить координаты левого верхнего угла поворота (x0,y0) и найти порядковый номер этого узла $N = (2*x0)^2$. Далее, в зависимости от номера участка, уже не сложно найти номер точки с координатами (x,y).

Аналогичные рассуждения можно повторить и для участков горизонтальной и вертикальной ветвей, принадлежащих правой полуплоскости. В программе для обозначения соответствующих четырех частей использованы буквенные обозначения — a, b, c, d. Для вычисления номера нашей точки здесь используется правый нижний угол поворота.

Ниже приводятся тексты программ, демонстрирующие работу процедуры `coord_to_n`.

Программа 2_29.bas

```
DECLARE FUNCTION CoordToN! (X&, Y&)
REM Вычисление номера узла по его координатам
INPUT "Введите координаты узла : ", X&, Y&
M& = CoordToN(X&, Y&)
PRINT "Его номер = "; M&
END

FUNCTION CoordToN (X&, Y&)
  IF X& <= Y& THEN 'если узел в левой полуплоскости
   IF X& >= 0 AND Y& > 0 THEN GOTO m12'случай 1
   IF X& < 0 AND Y& >= 0 AND Y& >= ABS(X&) THEN 'случай 2
     'обработка случаев 1 и 2
m12: x0 = -Y&: y0 = Y&: N& = (2 * x0) ^ 2'координаты и номер угла
     CoordToN = N& + x0 - X&: EXIT FUNCTION
   ELSE
    'обработка случаев 3 и 4
     x0 = X&: y0 = -X&: N& = (2 * x0) ^ 2
     CoordToN = N& + y0 - Y&: EXIT FUNCTION
   END IF
  END IF
' если узел в правой полуплоскости
   IF X& <= 0 AND Y& < 0 THEN GOTO mab' случай a
   IF X& >= 0 AND Y& < 0 AND ABS(Y&) >= X& THEN ' случай b
   ' обработка случаев a, b
mab: x0 = -Y& + 1: y0 = -Y&: N& = (2 * x0 - 1) ^ 2
     CoordToN = N& - x0 + X&: EXIT FUNCTION
   ELSE
    'обработка случаев c, d
     x0 = X&: y0 = -X& + 1: N& = (2 * x0 - 1) ^ 2
     CoordToN = N& + Y& - y0
   END IF
END FUNCTION
```

Программа 2_29.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
long sqr(long x);
long coord_to_n(long x,long y);
  void main()
{ long x,y,M;
  printf("\nВведите координаты узла : ");
  scanf("%ld %ld",&x,&y);
  M=coord_to_n(x,y);
  printf("\nЕго номер = %ld",M);
  getch();
}
long coord_to_n(long x,long y)
{ long N,x0,y0;
  if(x<=y) //если узел в левой полуплоскости
  { if(x>=0 && y>0) goto m12; //случай 1
    if(x<0 && y>=0 && y>=labs(x)) //случай 2
    {//обработка случаев 1 и 2
m12: x0=-y; y0=y; N=sqr(2*x0); //координаты и номер угла
     return N+x0-x;
    }
//обработка случаев 3 и 4
    x0=x; y0=-x; N=sqr(2*x0);
    return N+y0-y;
  }
//если узел в правой полуплоскости
  if(x<=0 && y<0) goto mab; //случай a
  if(x>=0 && y<0 && labs(y)>=x) //случай b
  {//обработка случаев a, b
mab: x0=-y+1; y0=-y; N=sqr(2*x0-1);
     return N-x0+x;
  }
//обработка случаев c, d
  x0=x; y0=-x+1; N=sqr(2*x0-1);
  return N+y-y0;
}
```

```
long sqr(long x)
{ return x*x; }
```

Программа 2_29.pas

```
program CoordToN;
var
  M,x,y:longint;
function coord_to_n(x,y:longint):longint;
var
  x0,y0,N:longint;
label m12,mab;
begin
  if x<=y  then { если узел в левой полуплоскости }
  begin if (x>=0) and(y>0) then goto m12; { случай 1 }
    if (x<0) and (y>=0) and (y>=abs(x)) then { случай 2 }
    begin { обработка случаев 1 и 2 }
m12: x0:=-y; y0:=y; N:=sqr(2*x0); { координаты и номер угла }
     coord_to_n:= N+x0-x; exit;
    end;
{ обработка случаев 3 и 4 }
    x0:=x; y0:=-x; N:=sqr(2*x0);
    coord_to_n:=N+y0-y; exit;
  end;
{ если узел в правой полуплоскости }
  if(x<=0) and (y<0) then goto mab; { случай a }
  if(x>=0) and (y<0) and (abs(y)>=x) then { случай b }
  begin { обработка случаев a, b }
mab: x0:=-y+1; y0:=-y; N:=sqr(2*x0-1);
     coord_to_n:= N-x0+x; exit;
  end;
{ обработка случаев c, d }
  x0:=x; y0:=-x+1; N:=sqr(2*x0-1);
  coord_to_n:= N+y-y0;
end;

begin
  writeln('Введите координаты узла : ');
  readln(x,y);
  M:=coord_to_n(x,y);
```

```
  writeln('Его номер = ',M);
  readln;
end.
```

На базе процедур `n_to_coord` и `coord_to_N` можно построить программы определения расстояния между точками с номерами `N1` и `N2`, нахождения номеров ближайших соседей к точке с номером `N` и т. п.

Программа 2_30.bas

```
DECLARE SUB NtoCoord (N&, X&, Y&)
REM Определение расстояния между двумя точками
  INPUT "Введи номер первой точки : ", N&
  NtoCoord N&, x1&, y1&
  INPUT "Введи номер второй точки : ", N&
  NtoCoord N&, x2&, y2&
  dx = x1& - x2&: dy = y1& - y2&
  PRINT "Расстояние = ", SQR(dx * dx + dy * dy)
END

SUB NtoCoord (N&, X&, Y&)
DIM k AS LONG, j AS LONG, d AS LONG
  j = INT(SQR(N&) + .5)
  d = N& - j * j
  k = j \ 2
  PRINT "j="; j, "d="; d, "k="; k
  IF (j MOD 2) <> 0 THEN
     IF d > 0 THEN
       X& = k + 1: Y& = -k + d
     ELSE
       X& = k + 1 + d: Y& = -k
     END IF
  ELSE
     IF d > 0 THEN
       X& = -k:  Y& = k - d
     ELSE
       X& = -k - d: Y& = k
     END IF
  END IF
END SUB
```

Программа 2_30.c

```
//Определение расстояния между двумя точками
#include <stdio.h>
#include <conio.h>
```

```
#include <math.h>
void n_to_coord(long N, long *x, long *y);
  void main()
{ long N,x1,y1,x2,y2;
  double dx,dy;
  printf("\nВведи номер первой точки : ");
  scanf("%ld",&N);
  n_to_coord(N,&x1,&y1);
  printf("\nВведи номер второй точки : ");
  scanf("%ld",&N);
  n_to_coord(N,&x2,&y2);
  dx=x1-x2;   dy=y1-y2;
  printf("\nРасстояние = %f",sqrt(dx*dx + dy*dy));
  getch();
}
void n_to_coord(long N,long *x,long *y)
{ long  k,j,d;
  j=floor(sqrt(N)+0.5);
  d=N-j*j;  k=j / 2;
  if (j % 2)
     { if (d>0) { *x=k+1;  *y=-k+d; }
        else { *x=k+1+d;*y=-k;   }
     }
  else
     { if (d>0) { *x=-k;   *y=k-d; }
        else { *x=-k-d; *y=k;   }
     }
}
```

Программа 2_30.pas

```
program spiral;
var
  x1,y1,x2,y2,N:longint;
  dx,dy:double;
procedure n_to_coord(N:longint;var x,y:longint);
  var
    k,j,d:longint;
  begin
```

```
    j:=trunc(sqrt(N)+0.5);
    d:=N-sqr(j);  k:=j div 2;
    if odd(j) then
      begin
        if d>0 then begin x:=k+1;   y:=-k+d;end
               else begin x:=k+1+d; y:=-k;  end;
      end
    else if d>0 then begin x:=-k;   y:=k-d; end
                 else begin x:=-k-d; y:=k;   end;
  end;
begin
  writeln('Введи номер первой точки');
  readln(N);
  n_to_coord(N,x1,y1);
  writeln('Введи номер второй точки');
  readln(N);
  n_to_coord(N,x2,y2);
  dx:=x1-x2;   dy:=y1-y2;
  writeln('Расстояние = ',sqrt(dx*dx + dy*dy));
  readln;
end.
```

Вторая схема нумерации точек целочисленной решетки, расположенной в первом квадранте, заключается в движении по раскручивающейся диагонали (рис. 2.2).

Рис. 2.2. Нумерация точек по диагоналям

Базовые процедуры по-прежнему сводятся к установлению прямых и обратных связей между координатами точек и их номерами. Пронумеруем диагонали в порядке их удаления от начала координат — 0, 1, 2, ... В качестве нулевой "диагонали" будем рассматривать вырожденный отрезок, содержащий единственную точку с номером `0`. Очевидно, что на диагонали с номером `k` находится `k+1` точек, удовлетворяющих уравнению соответствующей прямой:

```
X + y = b(k).
```

Заметим, что на диагоналях с нечетными номерами номера точек возрастают при движении снизу вверх, а на диагоналях с четными номерами — при движении сверху вниз. Осталось только подметить закономерность между номером диагонали, ее уравнением и координатами начальной и конечной точек. В этом нам поможет анализ табл. 2.1.

Таблица 2.1. *Закономерность между номером диагонали, ее уравнением и координатами начальной и конечной точек*

Номер диагонали	**Номера начальной и конечной точек**		**Координаты начальной и конечной точек**	
`nDiag`	`nBeg`	`nEnd`	`(xBeg, yBeg)`	`(xEnd, yEnd)`
0	0	0	(0, 0)	(0, 0)
1	1	2	(1, 0)	(0, 1)
2	3	5	(0, 2)	(2, 0)
3	6	9	(3, 0)	(0, 3)
4	10	14	(0, 4)	(4, 0)
5	15	20	(5, 0)	(0, 5)
6	21	27	(0, 6)	(6, 0)
7	28	35	(7, 0)	(0, 7)
8	36	44	(0, 8)	(8, 0)
9	45	54	(9, 0)	(0, 9)
10	55	65	(0,10)	(10,0)
.....	.....	....	....	....

По содержимому табл. 2.1 можно установить следующие факты:

```
nEnd = nBeg + nDiag
nBeg(nDiag+1) = nEnd(nDiag) + 1
```

Если `nDiag` — четное, то координаты начальной точки — `(0, nDiag)` и при перемещении по диагонали `x` уменьшается, а `y` возрастает. В противном случае начальная точка имеет координаты `(nDiag, 0)` и при перемещении по диагонали `x` возрастает, а `y` уменьшается.

Ниже приводятся тексты программ, определяющие номера смежных точек для точки с заданным номером. Для уменьшения количества передаваемых параметров характеристики диагонали вынесены в глобальные переменные. Массивы `dx` и `dy` предназначены для определения координат смежных точек, количество которых может варьироваться от трех (для точки с номером 0) или пяти (для начальной и конечной точек диагонали) до восьми (в случае внутренних точек). Анализ того, какая из восьми возможных точек является допустимой, вынесен в процедуру `nextpoint`. Она проверяет принадлежность точки первому квадранту, восстанавливает по координатам порядковый номер допустимой точки и выводит его на экран.

Программа 2_31.bas

```
DECLARE SUB NEXTPOINT (X&, Y&)
  DIM nDiag AS LONG   'номер диагонали
  DIM xBeg AS LONG, yBeg AS LONG 'начальная точка на диагонали
  DIM nPoints AS LONG 'количество точек на диагонали
  DIM nBeg AS LONG    'номер начальной точки на диагонали
  DIM dx(8), dy(8)
  DATA -1,-1,-1,0,1,1, 1, 0
  FOR J = 0 TO 7: READ dx(J): NEXT J
  DATA -1, 0, 1,1,1,0,-1,-1
  FOR J = 0 TO 7: READ dy(J): NEXT J
m1:
  CLS
  INPUT "Введите номер точки: ", N
REM Определение параметров диагонали
  nBeg = 0: M = N: xBeg = 0: yBeg = 0
  FOR k = 0 TO 200000000
    M = M - k
    IF M < 0 THEN EXIT FOR
    nBeg = nBeg + k
  NEXT k
  nDiag = k - 1: nPoints = k
  IF (nDiag MOD 2) = 0 THEN 'диагональ с четным номером
      yBeg = nDiag
      xN = xBeg + (N - nBeg)
      yN = yBeg - (N - nBeg)
  ELSE                      'диагональ с нечетным номером
      xBeg = nDiag
```

```
      xN = xBeg - (N - nBeg)
      yN = yBeg + (N - nBeg)
  END IF
  PRINT "Номера смежных точек: "
  FOR k = 0 TO 7
      X& = xN + dx(k)
      Y& = yN + dy(k)
      NEXTPOINT X&, Y&
  NEXT k
  PRINT
  PRINT "Хотите повторить - (y/n) : ";
m2:
  A$ = INKEY$: IF A$ = "" THEN GOTO m2
  IF A$ = "y" THEN GOTO m1
END

SUB NEXTPOINT (X&, Y&)
  SHARED nDiag AS LONG, xBeg AS LONG, yBeg  AS LONG, nPoints  AS LONG
  DIM N AS LONG, J AS INTEGER
  IF X& < 0 OR Y& < 0 THEN EXIT SUB
  nBeg = 0: nDiag = X& + Y&: xBeg = yBeg = 0
  nPoints = nDiag + 1
  FOR J = 0 TO nPoints - 1: nBeg = nBeg + J: NEXT J
  IF (nDiag MOD 2) = 0 THEN
      yBeg = nDiag: N = nBeg + yBeg - Y&
  ELSE
      xBeg = nDiag: N = nBeg + xBeg - X&
  END IF
  PRINT N; ",";
END SUB
```

Программа 2_31.c

```
#include <stdio.h>
#include <conio.h>
void nextpoint(long x,long y);
int dx[8]={-1,-1,-1,0,1,1, 1, 0};
int dy[8]={-1, 0, 1,1,1,0,-1,-1};
long nDiag,     //номер диагонали
```

```
     xBeg,yBeg, //координаты начальной точки на диагонали
     nPoints,   //количество точек на диагонали
     nBeg;      //номер начальной точки на диагонали
  void main()
{ long N,M,
       xN,yN,   //координаты введенной точки
       x,y,k;
  clrscr();
m:
  printf("\n\nВведите номер точки: ");
  scanf("%ld",&N);
//Определение параметров диагонали
  nBeg=0;
  M=N;
  xBeg=yBeg=0;
  for(k=0;;k++)
    { M=M-k;
      if(M<0) break;
      nBeg=nBeg+k;
    }
  nDiag=k-1;
  nPoints=k;
  if((nDiag % 2)==0)  //диагональ с четным номером
    { yBeg= nDiag;
      xN=xBeg+(N-nBeg);
      yN=yBeg-(N-nBeg);

    }
  else                //диагональ с нечетным номером
    { xBeg=nDiag;
      xN=xBeg-(N-nBeg);
      yN=yBeg+(N-nBeg);
    }
  printf("Номера смежных точек:\n" );
  for(k=0;k<8;k++)
    { x = xN + dx[k];
      y = yN + dy[k];
      nextpoint(x,y);
    }
```

```
  printf("\nХотите повторить - (y/n) : ");
  if(getch()=='y') goto m;
}
void nextpoint(long x,long y)
{ long N;

  if(x<0||y<0) return;
  nBeg=0;
  nDiag=x+y;
  xBeg=yBeg=0;
  nPoints=nDiag+1;
  for(int j=0;j<nPoints;j++) nBeg+=j;
  if((nDiag % 2)==0) { yBeg=nDiag; N=nBeg+yBeg-y;}
                 else { xBeg=nDiag; N=nBeg+xBeg-x; }
  printf("%ld ",N);
}
```

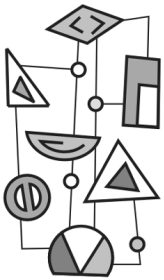
Программа 2_31.pas

```
program spiral1;
uses Crt;
label m1;
var
   nDiag,     {номер диагонали}
   xBeg,yBeg, {координаты начальной точки на диагонали}
   nPoints,   {количество точек на диагонали}
   nBeg,      {номер начальной точки на диагонали}
   N,M,
   xN,yN,     {координаты введенной точки}
   x,y,k: longint;
const
   dx : array [0..7] of integer =(-1,-1,-1,0,1,1, 1, 0);
   dy : array [0..7] of integer =(-1, 0, 1,1,1,0,-1,-1);

procedure nextpoint(x,y:longint);
var
  N:longint;
  j:integer;
begin
  if (x>=0)and(y>=0) then
```

```
  begin
    nBeg:=0;
    nDiag:=x+y;
    xBeg:=0;
    yBeg:=0;
    nPoints:=nDiag+1;
    for j:=0 to nPoints-1 do nBeg:=nBeg+j;
    if ((nDiag mod 2)=0) then
      begin
        yBeg:=nDiag;
        N:=nBeg+yBeg-y;
      end
     else
      begin
        xBeg:=nDiag;
        N:=nBeg+xBeg-x;
      end;
    write(N,',');
  end;
end;
begin
  clrscr;
m1:
  write('Введите номер точки: ');
  readln(N);
{ Определение параметров диагонали }
  nBeg:=0;
  M:=N;
  xBeg:=0;
  yBeg:=0;
  k:=0;
  repeat
      M:=M-k;
      if M<0 then break;
      nBeg:=nBeg+k;
      k:=k+1;
  until k<0;
  nDiag:=k-1;
  nPoints:=k;
```

```
  if(nDiag mod 2)=0 then  {диагональ с четным номером}
    begin
      yBeg:=nDiag;
      xN:=xBeg+(N-nBeg);
      yN:=yBeg-(N-nBeg);
     end
  else                {диагональ с нечетным номером}
    begin
      xBeg:=nDiag;
      xN:=xBeg-(N-nBeg);
      yN:=yBeg+(N-nBeg);
    end;

  writeln('Номера смежных точек : ');
  for k:=0 to 7 do
    begin
      x := xN + dx[k];
      y := yN + dy[k];
      nextpoint(x,y);
    end;
  writeln;
  writeln('Хотите повторить - (y/n) : ');
  if ReadKey='y' then goto m1;
end.
```

Глава 3



Обработка текстовой информации

Символьные данные и их внутреннее представление

Символьная (текстовая) информация — самый простой тип данных с точки зрения его представления в памяти ЭВМ. Каждому символу текста в памяти соответствует байт с 8-разрядным кодом этого символа в том или ином стандарте. Буквам латинского алфавита, цифрам, знакам операций и различным разделителям (скобки, точки, запятые и т. п.) в некотором смысле повезло больше, т. к. их кодировка практически универсальна. Она предложена фирмой IBM и составляет первую половину большинства 8-разрядных кодировочных таблиц, используемых в разных странах. В терминологии IBM PC такие таблицы принято называть кодовыми страницами. Вторая половина кодовых страниц отведена под национальную символику.

В отечественной практике чаще всего приходится иметь дело либо с символикой MS-DOS в стандарте ASCII (American Standard Code for Information Interchange), либо с одной из кодовых страниц ANSI (American National Standard Institute), применяемых в среде Windows. Между этими таблицами существует принципиальное различие, связанное с кодировкой букв русского алфавита.

В таблице ASCII (кодовая страница 866) заглавные русские буквы начинаются со 128-й позиции. Вплотную вслед за ними располагается цепочка строчных букв от буквы а (код — 160) до буквы п (код — 175). Далее следуют символы псевдографики, используемые для формирования таблиц с одинарными и двойными линиями (диапазон кодов от 176 до 223). Начиная с 224-й позиции располагаются остальные буквы от р до я. И наконец, вслед за ними, выбиваясь из алфавитного порядка, размещены буквы Ё (код — 240) и ё (код — 241).

В таблице ANSI (кодовая страница 1251) коды русских букв начинаются с позиции 192 (код буквы А) и расположены сплошным интервалом до 255-й позиции (код буквы я). Символы псевдографики в графической оболочке Windows смысла не имеют и поэтому в таблице ANSI отсутствуют. Буквы Ё и ё здесь тоже не включены в общепринятую алфавитную последовательность и

имеют, соответственно, коды 168 и 184. Более того, их обозначение на большинстве русифицированных клавиатур отсутствует, и для включения таких букв в набираемый текст приходится нажимать самую левую клавишу в верхнем ряду (на латинском регистре этой клавише соответствует символ "~").

Все достаточно развитые алгоритмические языки включают серию процедур по обработке символьных (каждый объект — отдельный символ) и строковых (цепочки символов) данных. В системе QBasic текстовые данные находятся в символьных переменных, имена которых либо заканчиваются знаком `$`, либо начинаются с одной из букв, упомянутых в объявлении вида `DEFSTR C-J`, либо описаны в операторе `DIM` сочетанием ...`AS STRING`. Базовым текстовым элементом в Си являются объекты типа `char`, значением каждого их которых является единственный символ. Объединение таких объектов в одномерный массив позволяет работать с цепочками символов. Паскаль обеспечивает работу как с одиночными символами (данные типа `char`), так и с их последовательностями (данные типа `string`).

Каждый из рассматриваемых языков обеспечивает возможность доступа к отдельному символу строки.

В Си это вытекает из способа представления текстовой информации. Начальное присвоение (`char name[5]="Вася";`) или копирование одной строки в другую (`strcpy(name,"Вася");`) эквивалентно 5 обычным операторам присваивания:

```
name[0]='В';
name[1]='а';
name[2]='с';
name[3]='я';
name[4]='\0';
```

Паскаль позволяет одновременно обращаться и к переменной типа `string` целиком и к отдельным ее символам как к значениям элементов одномерного массива:

```
const
  name : string[4]='Вася';
...............
  writeln(name[1],name);
```

Байт с нулевым индексом используется в Паскале для хранения информации о длине текущего значения.

С точки зрения внутреннего представления текстовых данных в памяти ЭВМ в языках программирования преобладают два похода. В первом случае перед цепочкой символов располагается один (Паскаль) или два (QBasic) байта, в которых хранится общее количество символов — длина цепочки. Для неинициированных данных она обычно равна 0. Во втором случае (Си)

вслед за цепочкой символов располагается байт с признаком конца текста (обычно это — нулевой код). Первый подход более прост с точки зрения программирования соответствующих процедур обработки строк. Однако он накладывает ограничение на максимальную длину строкового объекта, например, в Паскале длина строки не должна превышать 255 символов. Для преодоления этого недостатка и обеспечения совместимости по данным с другими системами программирования в последних версиях Паскаля были введены объекты типа `PChar`.

Ввод и вывод текстовой информации

Наиболее простые средства ввода символьных и строковых данных предлагают Паскаль и QBasic. Здесь достаточно указать в списке ввода имя переменной соответствующего типа:

```
INPUT A$
```

или

```
var
  c1:char;
  s1:string;
............
  readln(c1);
  readln(s1);
```

В Си имеется довольно много возможностей для ввода одиночных символов и цепочек строк. Форматный ввод с помощью функции `scanf` использует для этой цели два спецификатора:

`"%s"` — при вводе строковых значений в массив типа `char`;

`"%c"` — при вводе одиночных символов в переменную типа `char`.

Например:

```
  char c1,s1[80];
................
  scanf("%c",&c1); // не забывайте указывать адрес переменной
  scanf("%s",s1);  // имя массива одновременно является адресом
```

Ввод символьных и строковых данных завершается после нажатия клавиши <Enter>, однако в переменную `c1` поступит только один символ, а в массив `s1` при этом будет введена начальная цепочка символов до первого пробела. В операторе форматного ввода пробел или несколько подряд идущих пробелов рассматриваются как признак конца информации, считываемой из очередного поля. Конечно, при наборе вы можете набрать в одной строке несколько символов или несколько слов, разделенных пробелами. Все они

будут введены в системный буфер, выделяемый для работы функции `scanf` (стандартный размер этого буфера позволяет набрать строку длиной до 128 символов, включая и разделители-пробелы). При последующих обращениях к функции ввода данные из буфера продолжают извлекаться до тех пор, пока не будут исчерпаны, после чего вновь придется продолжить набор на клавиатуре.

Конечно, такой способ ввода может доставить неприятности и привести к непредусмотренному продолжению работы программы. Понятно также, что для ввода одного символа не хочется нажимать две клавиши, а вводимые строки могут состоять и из нескольких слов. Поэтому ввод символьных данных лучше выполнять с помощью других процедур:

```
int c1;
c1=getch();     //Ввод кода нажатой клавиши без отображения
                //соответствующего символа на экране
c1=getche();    //Ввод кода нажатой клавиши с соответствующего
                //символа на экране
c1=getchar();   //Ввод кода нажатой клавиши вслед за нажатием
                //клавиши Enter
```

Обратите внимание на то, что вводимый символ передается не в переменную типа `char`, а в двухбайтовую целочисленную переменную. Именно такие значения возвращают указанные выше функции. Вторая особенность их применения связана с разбиением клавиатуры на две категории клавиш — отображаемые и управляющие. Окраска клавиш не совсем точно передает принадлежность клавиши той или иной группе. Например, функциональные клавиши <F1>, <F2> ... <F12> имеют разный цвет, но все они относятся к категории управляющих. А клавиша <Enter>, несмотря на серую окраску, причислена к разряду обычных.

Дело в том, что при нажатии обычной клавиши в буфер входного потока (`stdin`) поступает единственный байт с ненулевым кодом. Именно он и извлекается при первом же обращении к одной из функций `getch`, `getchar` или `getche`. От нажатия управляющих клавиш в буфер `stdin` поступают два байта, первый из которых содержит нулевой код, а второй представляет уже собственно числовой код, соответствующий выбранной клавише. Для извлечения второго байта к этим функциям приходится обращаться повторно, и таким образом программа может проанализировать сложившуюся ситуацию.

Есть некоторые нюансы и при нажатии клавиши <Enter>. Так, функция `getch` сообщает, что нажата клавиша с числовым кодом 13, а функция `getchar` считает, что последним введен символ с кодом 10. На самом деле она перекодирует код клавиши <Enter> в символ `LF` (`line feed` — строка заполнена), действительно имеющий код 10 (`0x0A`) в таблице ASCII. По-разному воспринимают эти функции и комбинацию <Ctrl>+<z>, сообщая в

одном случае, что поступил признак конца файла (`end-of-file`) с кодом 26, а в другом — что встретился признак `EOF`, которому соответствует числовой код −1. Функция `getchar` не позволяет ввести код клавиши <Esc>, тогда как функции `getch` и `getche` успешно справляются с этой клавишей.

Зато с функций `gets`, осуществляющей ввод строки, у вас никаких проблем не возникнет:

```
gets(s);
```

Вы можете набирать любые предложения, содержащие любое количество пробелов, и все они будут введены в строку `s`. Однако длина вводимой строки ограничена емкостью буфера (128 байт).

Дополнительные возможности по вводу текстовых данных связаны с использованием потоков:

```
#include <iostream.h>
.....................
  char c1,s1[80];
  ................
  cin >> c1;
  cin >> s1;
```

С выводом символьных и строковых значений все обстоит гораздо проще. В Паскале и QBasic достаточно в списке выводимых значений указать данные соответствующего типа:

```
PRINT A$,"Вася"
```

или

```
var
  c1:char;
  s1:string;
............
  writeln(c1,s1,'Вася');
```

Форматный вывод в Си при помощи функции `printf` использует указанные выше спецификаторы, однако константные текстовые данные, которыми перемежаются выводимые значения, здесь располагаются между спецификаторами формата:

```
printf("%c Вася %s",c1,s1);
```

Вывод отдельного символа или одиночной строки в Си можно выполнить и с помощью функций `putchar(c1)` и `puts(s1)`.

Выводимое значение обычно располагается на экране, начиная с текущей позиции курсора. Если необходимо разместить текст в заранее предусмот-

ренном месте, следует прибегнуть к одной из служебных процедур предварительного перемещения курсора в заданную позицию:

```
LOCATE col,row    'Так это выглядит на QBasicgotoxy(col,row);  // А так
делается на Си и Паскале
```

Параметр `col` задает номер колонки в диапазоне от 1 до 80, а второй аргумент (`row`) определяет номер строки в диапазоне от 1 до 25.

Еще один способ управления по размещению текста связан с заданием ширины поля, отведенного под выводимое значение. Отображаемый текст при этом прижимается к правой границе поля. Ширина поля в Паскале задается числовым выражением, которое записывается в операторе вывода через двоеточие вслед за выводимым текстовым значением:

```
writeln('Вася':10,c1:k+5,s1:7);
```

В QBasic для указания ширины поля используется оператор `PRINT USING`:

```
PRINT USING "##### ### #######";"Вася",A$,B$
```

В Си ширина поля включается в спецификатор формата (`"%3c %10s"`). Однако здесь имеется дополнительная возможность указать, что выводимое значение требуется прижать к левой границе выделенного поля (`"%-3c" %-10s"`).

Обработка фрагментов строк

Под фрагментом строки понимают цепочку символов заданной длины, выделенную из исходной строки, начиная с указанной позиции. В частности, выделяемый фрагмент может состоять и из единственного символа. Кроме выделения фрагмента тем или иным способом к числу наиболее распространенных операций относятся действия по определению длины строки, объединению (конкатенации) символьных цепочек и поиску вхождения одной строки в другую.

В QBasic для выделения фрагментов используют системные функции `LEFT$` (выделение левой подстроки), `RIGHT$` (выделение правой подстроки) и `MID$` (выделение внутренней подстроки). Последняя функция может выступать и в роли оператора, заменяющего старый фрагмент новым значением. Например:

```
LEFT$("Вася",1)                 'выделяет "В"
LEFT$("Вася",2)                 'выделяет "Ва"
RIGHT$("Вася",1)                'выделяет "я"
RIGHT$("Вася",2)                'выделяет "ся"
MID$("Вася",2,2)                'выделяет "ас"
MID$("Вася",3)="илиса"          'заменяет на "Василиса"
```

Функция (оператор) `MID$` позволяет опустить третий аргумент, и тогда в соответствующей операции участвуют все конечные символы, начиная с указанного. В операторе `MID$` присваиваемое значение может быть пустым, что эквивалентно удалению фрагмента.

В Паскале для выделения подстроки используется функция `copy`, аналогичная функции `MID$`:

```
s2 := copy('Вася',2,3);  {выделяется 'ася'}
```

Для удаления или вставки фрагмента здесь используются процедуры `delete` и `insert`:

```
delete(s1,start,len);{удаляется len символов, начиная с позиции start}
insert(s1,s2,start);{в строку s2, начиная с позиции start, вставляется
                     строка s1}
```

Функции работы со строками в Си включены в состав заголовочного файла `string.h`. Для копирования строки или ее части в другую здесь можно воспользоваться одной из следующих функций:

```
strcpy(s1,s2);    //копирует строку s2 в строку s1
strncpy(s1,s2,n); //копирует первые n символов из строки s2 в s1
```

Задавая аргумент-источник не ссылкой на начало символьного массива, а адресом любого его элемента, мы можем скопировать либо правую, либо среднюю подстроку:

```
strcpy(s1,&s2[k]);    //копирует правую подстроку из s2 в s1
strncpy(s1,&s[2],n);  //копирует среднюю подстроку из s2 в s1
```

Длина строк в рассматриваемых системах программирования определяется одной из системных функций `LEN` (QBasic), `Length` (Паскаль) или `strlen` (Си). Единственным аргументом у каждой из них является анализируемая строка.

Для конкатенации (объединения) строк в Паскале и QBasic используется довольно естественная операция сложения:

```
A$="Здравствуй, "+NAME$+"!"
s1:='Здравствуй, '+name+'!'
```

В Си эта операция реализуется с помощью одной из следующих функций:

```
strcat(s1,s2);       //добавляет s2 к s1
strncat(s1,s2,n);    //добавляет n первых символов из s2 к s1
```

Поиск вхождения одной строки в другую дает ответ на вопрос, содержится ли значение одного текста в другом и с какой позиции обнаружено это вхождение. Нулевая позиция в качестве результата такой операции соответствует отрицательному ответу.

Функция, определяющая в QBasic, входит ли значение строки `A2$` в строку `A1$`, имеет вид:

`INTSR(A1$,A2$)` или `INSTR(k,A1$,A2$)`

В первом случае анализ вхождения ведется с начала строки `A1$`, во втором случае — начиная с `k`-й позиции строки `A1$`. Последний вариант позволяет последовательно определить все вхождения искомого образца.

Примерно такими же возможностями обладает функция `pos(s1,s2)` в Паскале. Для поиска повторного вхождения можно удалить уже исследованный фрагмент и снова обратиться к функции `pos`.

Гораздо более разнообразные варианты поиска вхождений предлагает Си:

```
strstr(s1,s2);  //ищет вхождение строки s2 в s1
strchr(s1,c);   //ищет вхождение символа "c" с начала строки s1
strrchr(s1,c);  //ищет вхождение символа "c" с конца строки s1
strpbrk(s1,s2); //ищет вхождение любого символа из s2 в s1
strspn(s1,s2);  //ищет вхождение любого фрагмента, составленного
                //из символов s2 в s1
```

Сравнение и сортировка текстовых данных

Операции сравнения отдельных символов или строк основаны на последовательном анализе отношений числовых значений соответствующих кодов. В кодовых страницах символы букв упорядочены в соответствии с их расположением в латинском или национальном алфавитах. Поэтому код буквы `A` меньше кода буквы `F`, код буквы `Г` меньше кода буквы `Ю` и т. д.

Некоторое неудобство вызывает тот факт, что одноименные заглавные и строчные буквы имеют разные коды — в одной клетке кодировочной таблицы можно разместить только один символ, кроме того, заглавные и строчные буквы имеют разный смысл. Это не позволяет напрямую упорядочить слова в соответствии с их лексикографическим расположением в словарях. Поэтому приходится предварительно заменять коды всех малых букв в тексте на коды больших (или наоборот) и только после этого выполнять операцию сравнения. Такая замена для букв латинского алфавита особых проблем не представляет, т. к. смещение между кодами соответствующих заглавных и строчных букв — величина постоянная. А вот с русскими буквами приходится повозиться — в кодировке ASCII цепочка строчных букв между `п` и `р` разорвана символами псевдографики, а буквы `Ё` и `ё` вообще находятся не на своих местах.

Учитывая эту специфику, следует достаточно внимательно использовать языковые средства, связанные с преобразованием или игнорированием разницы в кодировке заглавных и строчных букв. Для русскоязычных текстов их применять нельзя.

QBasic позволяет преобразовывать содержимое символьных строк к верхнему (`UCASE$(A$)`) или нижнему (`LCASE$(A$)`) регистру. В Паскале имеется только одна функция `UpCase(c)`, заменяющая строчную букву заглавной. Но коды символов, не принадлежащих множеству букв латинского алфавита, она оставляет без изменения. Такая же оговорка распространяется на функции `strupr(s)` и `strlwr(s)` в Си.

В Паскале и QBasic операции сравнения символьных данных ничем не отличаются от аналогичных операций над числовыми величинами:

```
REM Упорядочение слов по "возрастанию"
IF A$>B$ THEN
  TMP$=A$ : A$=BS : B$=TMP$
END IF
```

или

```
{Упорядочение слов по "убыванию"}
if s1<s2 then
begin
  tmp:=s1;  s1:=s2;  s2:=tmp;
end;
```

Для сравнения строк Си предлагает довольно много системных функций, но не забывайте, что их действие не всегда допустимо над русскими словами. Каждая из описываемых ниже функций принимает положительное значение, если ее первый операнд строго "больше" второго, нулевое значение при "равенстве" операндов, и отрицательное значение, если первый операнд оказался "меньше".

```
strcmp(s1,s2);       //сравнивает строки s1 и s2
strcmpi(s1,s2);      //сравнивает s1 и s2 с игнорированием
                     //разницы между большими и малыми буквами
stricmp(s1,s2);      //эквивалентна функции strcmpi
strncmp(s1,s2,k);    //сравнивает первые k символов в s1 и s2
strncmpi(s1,s2,k);   //сравнивает первые k символов в s1 и s2
                     //с игнорированием разницы между большими
                     //и малыми буквами
strnicmp(s1,s2,k);   //эквивалентна функции strncmpi
```

Управление цветом в текстовом режиме

Наиболее важная информация, которую можно почерпнуть из многочисленных источников (книги, руководства по системам программирования, файлы помощи), сводится к трем следующим фактам:

- существует несколько текстовых режимов работы монитора, различающихся по цветовой гамме, а также по числу строк и столбцов, отображаемых на экране;
- содержимое экрана отражает состояние активной страницы видеопамяти;
- каждой символьной позиции экрана (так называемому знакоместу) в видеопамяти соответствуют 2 байта, в первом из которых находится код ASCII отображаемого символа, а во втором — атрибуты, управляющие цветом пикселов переднего плана (контуры буквы) и фоновых пикселов.

Самый распространенный текстовый режим допускает отображение в цвете 25 строк, каждая из которых содержит по 80 символов. При этом объем соответствующей страницы видеопамяти равен 4000 байт и каждый символ, отображаемый на экране, в принципе, может иметь индивидуальные цветовые характеристики, не зависящие от цветовой гаммы в соседних позициях.

В любом руководстве приводится следующее описание формата байта цветовых атрибутов:

7	6	5	4	3	2	1	0
B	b	b	b	I	f	f	f

Самый старший бит управляет режимом мерцания символа, которое осуществляется аппаратным способом примерно с частотой 1 раз в секунду при `B = 1`. Три следующих бита (`bbb`) представляют код цветности фона, который должен принадлежать интервалу [0, 7]. Четверка младших битов (`Ifff`) определяет код цветности переднего плана. Среди них особо выделяют бит `I`, единичное значение которого обычно соответствует повышенной яркости, т. е. цвету, сопровождаемому приставкой "ярко" или "светло".

Из сказанного выше следует, что стандартный текстовый режим на цветном мониторе позволяет задавать для каждого символа один из 8-ми фоновых оттенков и один из 16-ти цветов, выделяющих контур символа. А если вы хотите вывести мерцающий текст, то байты атрибутов соответствующих символов должны быть увеличены на 128.

Управление содержимым байта цветовых атрибутов осуществляется с помощью одной или нескольких системных процедур (операторов), действие которых распространяется на все последующие операции вывода до новой переустановки характеристик цветности.

В Си и Паскале код цветности символов задается с помощью процедуры `textcolor(fc)`. Значение фонового цвета изменяется процедурой `textbackground(bc)`. В этих же системах программирования существует и более универсальная процедура `textattr`, позволяющая за одно обращение изменить оба цвета и установить признак мерцания:

```
textattr(B + bc*16 + fc);
```

Следует отметить, что при работе с мониторами SVGA сравнительно давно разработанные системы Borland C++ 3.1 и Turbo Pascal 7.0 не очень точно следуют ограничениям, вытекающим из структуры байта цветовых атрибутов. В указанных выше процедурах вы можете задавать цвет фона из диапазона [0, 15], а эффект мерцания символов может и не наблюдаться.

Кроме того, в Си для вывода разноцветного текста следует использовать не стандартную функцию вывода `printf`, а функцию `cprintf`, использующую "прямое" обращение к видеопамяти.

Система QBasic абсолютно точно придерживается описанного выше формата цветовых атрибутов и управляет значениями соответствующих характеристик с помощью оператора `COLOR`:

```
COLOR fc, bc
```

Для вывода мерцающего текста к коду цветности символа `fc` (`0≤fc≤15`) следует добавить 16. Выход за пределы допустимых значений цвета фона или цвета букв приводит к аварийному завершению работы программы.

Задачи, советы и ответы

Задание 3.01. Формирование таблицы ASCII

Сформировать на экране таблицу ASCII таким образом, чтобы в ней были представлены все отображаемые символы 866-й кодовой страницы и их числовые коды. Один из возможных вариантов заполнения экрана такой таблицей представлен на рис. 3.1.

Совет 1 (общий)

Приведенная выше таблица содержит 21 кодовую строку, и смещение начальных кодов в смежных колонках равно 21. Поэтому внешний цикл повторяется 21 раз, начинаясь с первого отображаемого кода (32 — код пробела). Так как заголовок отделен от таблицы одной пустой строкой, вывод начинается с 3-й строки (3 = 32 – 29). А дальше остается только перебирать все коды отображаемого диапазона (от 32 до 255) и выводить пары "символ-код" по фиксированному формату — одна позиция под символ, разделяющий пробел, три позиции — под числовой код и пара пробелов между колонками.

Таблица ASCII

	32	5	53	J	74	_	95	t	116	Й	137	Ю	158	│	179	└	200	▌	221	Є	242
!	33	6	54	K	75	`	96	u	117	К	138	Я	159	┤	180	╔	201	▐	222	є	243
"	34	7	55	L	76	a	97	v	118	Л	139	а	160	╡	181	╩	202	▄	223	Ї	244
#	35	8	56	M	77	b	98	w	119	М	140	б	161	╢	182	╦	203	р	224	ї	245
$	36	9	57	N	78	c	99	x	120	Н	141	в	162	╖	183	╠	204	с	225	Ў	246
%	37	:	58	O	79	d	100	y	121	О	142	г	163	╕	184	═	205	т	226	ў	247
&	38	;	59	P	80	e	101	z	122	П	143	д	164	╣	185	╬	206	у	227	°	248
'	39	<	60	Q	81	f	102	{	123	Р	144	е	165	║	186	╧	207	ф	228	∙	249
(	40	=	61	R	82	g	103	|	124	С	145	ж	166	╗	187	╨	208	х	229	·	250
)	41	>	62	S	83	h	104	}	125	Т	146	з	167	╝	188	╤	209	ц	230	√	251
*	42	?	63	T	84	i	105	~	126	У	147	и	168	╜	189	╥	210	ч	231	№	252
+	43	@	64	U	85	j	106	⌂	127	Ф	148	й	169	╛	190	╙	211	ш	232	¤	253
,	44	A	65	V	86	k	107	А	128	Х	149	к	170	┐	191	╘	212	щ	233	▪	254
-	45	B	66	W	87	l	108	Б	129	Ц	150	л	171	└	192	╒	213	ъ	234		255
.	46	C	67	X	88	m	109	В	130	Ч	151	м	172	┴	193	╓	214	ы	235		
/	47	D	68	Y	89	n	110	Г	131	Ш	152	н	173	┬	194	╫	215	ь	236		
0	48	E	69	Z	90	o	111	Д	132	Щ	153	о	174	├	195	╪	216	э	237		
1	49	F	70	[	91	p	112	Е	133	Ъ	154	п	175	─	196	┘	217	ю	238		
2	50	G	71	\	92	q	113	Ж	134	Ы	155	░	176	┼	197	┌	218	я	239		
3	51	H	72	]	93	r	114	З	135	Ь	156	▒	177	╞	198	█	219	Ё	240		
4	52	I	73	^	94	s	115	И	136	Э	157	▓	178	╟	199	▄	220	ё	241		

Рис. 3.1. Таблица ASCII

Программа 3_01.bas

```
REM Вывод таблицы ASCII
CLS : PRINT TAB(31); "Таблица ASCII"
FOR I=32 TO 52: LOCATE I-29,1
  FOR J=I TO 255 STEP 21
    PRINT USING "! ###  ";CHR$(J);J;
  NEXT J
NEXT I
```

Программа 3_01.c

```
/* Вывод таблицы ASCII */
#include <stdio.h>
  main()
{
  int i,j;
  clrscr();
  gotoxy(31,1);
  printf("Таблица ASCII");
  for (i=32; i<=52; i++)
    {
```

```
      gotoxy(1,i-29);
      for (j=i; j<=255; j+=21)
      printf("%c %3d  ",j,j);
    }
  getch();
}
```

Программа 3_01.pas

```
program ASCII;
{Вывод таблицы ASCII}
uses Crt;
var
  i,j:word;
begin
  clrscr;
  gotoxy(31,1);
  write('Таблица ASCII');
  for i:=32 to 52 do
  begin
    gotoxy(1,i-29);
    j:=i;
    repeat
      write(chr(j):1,j:4,'  ');
      j:=j+21;
    until j>255;
  end;
  readln;
end.
```

Задание 3.02. Преобразование строк к верхнему регистру

Составить подпрограмму (функцию) `Up(S)`, которая заменяет в строке `S` коды малых букв кодами аналогичных больших букв с учетом их расположения в таблице ASCII.

Совет 1 (общий)

Заглянув в таблицу ASCII, нетрудно убедиться в том, что замене подлежат коды символов, попадающие в следующие интервалы:

- [97,122] — строчные латинские буквы от "a" до "z";

- [160,175] — строчные русские буквы от "а" до "п";
- [224,239] — строчные русские буквы от "р" до "я";
- [241,241] — буква "ё".

По этой же таблице можно установить, что смещения в кодах между выделенными группами малых букв и их прописными эквивалентами, соответственно, равны 32, 32, 80 и 1 позициям.

Совет 2 (QBasic)

Для выделения и изменения `j`-го символа строки `S` удобнее всего воспользоваться функцией (оператором) `MID$(S,J,1)`. Анализ на принадлежность одному из кодовых интервалов, подлежащему замене, целесообразно оформить в виде переключателя `SELECT CASE` — `END SELECT`. Наконец, функции `ASC` и `CHR$` понадобятся, соответственно, для преобразования символа в код ASCII и модифицированного кода в символ эквивалентной большой буквы.

В приводимых ниже двух вариантах программы преобразование к верхнему регистру выполнено в виде подпрограммы `UP` и функции `UP$`. Чтобы не портить аргумент функции и другие глобальные переменные головной программы, в теле функции объявлены свои локальные переменные `B` и `J`.

Совет 3 (Си)

Обратите внимание на то, что локальный массив `b`, объявленный в теле функции `up`, имеет тип `unsigned char`, а не `char`. Объясняется это тем, что проверяемые коды (а они в теле функции выступают как числа) принадлежат диапазону [0,255].

Совет 4 (Паскаль)

Обратите внимание на то, что в теле функции не понадобился промежуточный массив `b`, т. к. параметр `a` передавался не по адресу (не `var a:string`), а по значению. Поэтому исходная строка оказалась в стеке, и ею можно было воспользоваться в теле функции, не опасаясь порчи параметра в вызывающей программе.

Программа 3_02.bas

```
REM Замена малых русских букв большими
DECLARE SUB UP (A$)
PRINT "Введите строку, содержащую малые и большие буквы"
INPUT "",A$
UP A$
PRINT A$
END

SUB UP(A$)
FOR j=1 TO LEN(A$)
```

```
  SELECT CASE MID$(A$,j,1)
    CASE "а" TO "z": MID$(A$,j,1)=CHR$(ASC(MID$(A$,j,1))-32)
    CASE "а" TO "п": MID$(A$,j,1)=CHR$(ASC(MID$(A$,j,1))-32)
    CASE "р" TO "я": MID$(A$,j,1)=CHR$(ASC(MID$(A$,j,1))-80)
    CASE "ё": MID$(A$,j,1)="Ё"
  END SELECT
NEXT j
END SUB
```

Программа 3_02a.bas

```
REM Замена малых русских букв большими
DECLARE FUNCTION UP$(A$)
PRINT "Введите строку, содержащую малые и большие буквы"
INPUT "",A$
B$=UP$(A$)
PRINT A$
PRINT B$
END

FUNCTION UP$(A$)
'Чтобы не испортить аргумент A$, введена локальная переменная
DIM B AS STRING
B=A$
FOR J=1 TO LEN(A$)
  SELECT CASE MID$(B,J,1)
    CASE "а" TO "z": MID$(B,J,1)=CHR$(ASC(MID$(B,J,1))-32)
    CASE "а" TO "п": MID$(B,J,1)=CHR$(ASC(MID$(B,J,1))-32)
    CASE "р" TO "я": MID$(B,J,1)=CHR$(ASC(MID$(B,J,1))-80)
    CASE "ё": MID$(B,J,1)="Ё"
  END SELECT
NEXT J
UP$=B
END FUNCTION
```

Программа 3_02.c

```
/* Замена малых русских букв большими */
#include <stdio.h>
```

```
#include <conio.h>
#include <string.h>
char *up(char *a);

  void main()
{
  char a[80],*b;
  printf("\nВведите строку, содержащую малые и большие буквы\n");
  gets(a);
  b=up(a);
  printf("\na=%s",a);
  printf("\nb=%s",b);
  getch();
}

char *up(char *a)
{
  unsigned char b[80];
  int j;
  strcpy(b,a);
  for(j=0; j<strlen(a); j++)
    {
     if ( 97<=b[j] && b[j]<=122) b[j]-=32;
     if (160<=b[j] && b[j]<=175) b[j]-=32;
     if (224<=b[j] && b[j]<=239) b[j]-=80;
     if (241==b[j]) b[j]--;
    }
 return b;
}
```

Программа 3_02.pas

```
program UperCase;
{ Замена малых русских букв большими }
var
  a,b:string[80];
function up(a:string):string;
var
  j:integer;
```

```
begin
  for j:=1 to length(a) do
    case a[j] of
      'a'..'z': a[j]:=chr(ord(a[j])-32);
      'a'..'п': a[j]:=chr(ord(a[j])-32);
      'р'..'я': a[j]:=chr(ord(a[j])-80);
      'ё': a[j]:='Ё';
    end;
  up:=a;
end;
begin
  writeln('Введите строку, содержащую малые и большие буквы ');
  readln(a);
  b:=up(a);
  writeln(a);
  writeln(b);
  readln;
end.
```

Задание 3.03. Сортировка фамилий

Составить программу, которая запрашивает у пользователя 10 фамилий, сортирует их по алфавиту и выводит отсортированный список. Пока сортируемый массив мал, мы не будем обращать внимание на эффективность алгоритма сортировки. Один из следующих разделов специально будет посвящен этому достаточно сложному и важному вопросу.

Так как программа должна сортировать фамилии, то следует установить правила их набора на клавиатуре, исключающие неоднозначность результатов сравнений:

- вначале набирается фамилия, вслед за которой через пробел могут следовать однобуквенные инициалы, завершаемые точками;
- первая буква фамилии и символы инициалов должны быть большими;
- фамилии должны набираться либо только русскими, либо только латинскими буквами;
- допускается набор составных фамилий с использованием разделяющего символа (например, `Петров-Водкин`);
- буквы `Ё` и `ё` в фамилиях заменяются буквами `Е` и `е`.

Такие правила позволят вам не обращать внимания на различие между кодами больших и малых букв, т. к. сравниваться между собой, как правило,

будут буквы одного регистра, коды которых соответствуют лексикографическому порядку. Коды инициалов и символы-разделители (точки, пробелы, тире) в таблице ASCII расположены раньше кодов малых букв и поэтому на результаты сравнений не повлияют.

Совет 1 (общий)

Для повышения наглядности результатов работы программы можно вывести на экран два списка фамилий: слева — неупорядоченный, справа — упорядоченный.

Совет 2 (QBasic)

Для перевода курсора в позицию, соответствующую очередной строке списка, надо воспользоваться оператором `LOCATE row, col` (`row` — номер строки, `col` — номер столбца).

Совет 3 (Си)

В программе 3_03a.c демонстрируется одна из уловок, повышающая эффективность работы любой программы сортировки достаточно длинных документов. С подобного рода приемами вы неоднократно сталкивались, но, возможно, не обращали на них внимания. Например, на панелях Norton Commander'a можно выбрать один из нескольких способов упорядочения оглавления — по именам файлов, по их расширениям, по размерам, по датам создания, в порядке появления файлов на диске. Естественно, что такого рода сортировки не производят никаких перемещений файлов на диске. Вместо этого происходят перестановки каких-то указателей в оперативной памяти.

В программе 3_03a.c для размещения фамилий использованы динамически запрашиваемые строки, на начало которых "смотрят" элементы массива указателей типа `char` (см. обращение к функции `malloc`). Вместо перестановки фамилий, расположенных не в алфавитном порядке, меняются местами соответствующие 4-байтовые указатели.

Перевод курсора в начало очередной строки списка осуществляется функцией `gotoxy(col,row)`, последовательность параметров которой прямо противоположна соответствующему оператору QBasic.

Программа 3_03b.c построена менее замысловато. В ней использован двумерный массив для хранения фамилий, да и вывод результатов оформлен попроще.

Программа 3_03.bas

```
REM Сортировка фамилий
DIM NAME$(10)
N=10
CLS
FOR J=1 TO 10
  INPUT "Введи очередную фамилию - ", NAME$(J)
```

```
NEXT J
CLS
PRINT "Фамилии до упорядочения : "
FOR J=1 TO N
  LOCATE J+2,1: PRINT NAME$(J)
NEXT J
FOR J=1 TO N-1
  FOR K=J+1 TO N
    IF NAME$(J)>NAME$(K) THEN
       TMP$=NAME$(J)
       NAME$(J)=NAME$(K)
       NAME$(K)=TMP$
    END IF
  NEXT K
NEXT J
LOCATE 1,40
PRINT "Фамилии после упорядочения :"
FOR J=1 TO N
  LOCATE J+2,40: PRINT NAME$(J)
NEXT J
END
```

Программа 3_03a.c

```
/* Сортировка фамилий */
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>

  main()
{
#define n_max 10
#define len_max 20
  int  j,k;
  char tmp[len_max], *names[n_max], *p;
  clrscr();
  for(j=0; j< n_max; j++)
    {
```

```
      printf("\nВведи очередную фамилию - ");
      scanf("%s",tmp);
      names[j]=(char *)malloc(len_max);
      strcpy(names[j],tmp);
    }
  clrscr();
  printf("Фамилии до упорядочения :");
  for(j=0; j<n_max; j++)
     {
       gotoxy(1,j+2);
      printf("%s",names[j]);
     }
  for(j=0; j<n_max-1; j++)
    for(k=j+1; k<n_max; k++)
      {
       if (strcmp(names[j],names[k]) > 0)
        {
         p=names[j];
         names[j]=names[k];
         names[k]=p;
        }
      }
  gotoxy(40,1);
  printf("Фамилии после упорядочения :");
  for(j=0; j<n_max; j++)
    {
      gotoxy(40,j+2);
      printf("%s",names[j]);
    }
  getch();
}
```

Программа 3_03b.c

```
/* Сортировка фамилий */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define N 10
```

```
  main()
{
  char a[N][20],tmp[20];
  int i,j;
  clrscr();
  puts("Введите 10 фамилий по одной в строке");
  for(i=0; i<N; i++)
    gets(&a[i][0]);
  for(i=0; i<N-1; i++)
    for(j=i+1; j<N; j++)
      if(strcmp(&a[i][0],&a[j][0])>0)
        {
          strcpy(tmp,&a[i][0]);
          strcpy(&a[i][0],&a[j][0]);
          strcpy(&a[j][0],tmp);
        }
  puts("А теперь они же по алфавиту");
  for(i=0; i<N; i++)
    puts(&a[i][0]);
  getch();
}
```

Программа 3_03.pas

```
program sort;
{ Сортировка фамилий }
uses crt;
const
  n=10;
var
  j,k:integer;
  tmp:string[20];
  name:array [1..n] of string[20];
begin
  clrscr;
  for j:=1 to n do
    begin
      writeln('Введи очередную фамилию');
      readln(name[j]);
    end;
```

```
  clrscr;
  writeln('Фамилии до упорядочения :');
  for j:=1 to n do
    begin
      gotoxy(1,j+2);
      write(name[j]);
    end;
  for j:=1 to n-1 do
    for k:=j+1 to n do
      begin
        if name[j]>name[k] then
          begin
            tmp:=name[j];
            name[j]:=name[k];
            name[k]:=tmp;
          end
      end;
  gotoxy(40,1);
  writeln('Фамилии после упорядочения :');
  for j:=1 to n do
    begin
      gotoxy(40,j+2);
      write(name[j]);
    end;
  readln;
end.
```

Задание 3.04. Подсчет числа слов в строке

Предполагая, что вводимая строка содержит не более 80-ти символов и состоит из "слов", разделенных одним или более пробелами, подсчитать количество слов в строке. Обратите внимание на критические ситуации — строка пуста, строка состоит из одних пробелов, строка содержит единственное слово, слова могут содержать единственный символ, отличный от пробела.

Совет 1 (общий)

В начале следует проверить длину введенной строки и, если она равна 0, сообщить, что число слов тоже равно 0. Затем имеет смысл проверить первый символ строки и, если он отличен от пробела, зафиксировать начало первого слова. Дальнейший анализ можно оформить в виде цикла, в котором проверяется очередной символ строки с учетом одной из двух возможных ситуаций —

ему предшествовал пробел или символ, отличный от пробела. При переходе с пробела на непробел счетчик слов следует увеличить на 1. Переменная `space` хранит информацию о предшествующем символе — она равна `1` или `true`, если предшествовавшим символом был пробел. Ее значение меняется на противоположное, если только один из двух смежных символов является пробелом.

Программа 3_04.bas

```
REM Подсчет числа слов в строке
CLS
SPACE=1: WORDS=0
PRINT "Введите строку"
INPUT "",S$
LENS=LEN(S$)
IF LENS=0 THEN GOTO RET
IF LEFT$(S$,1)<>" " THEN SPACE=0: WORDS=1
FOR J=2 TO LENS
  IF SPACE=1 AND MID$(S$,J,1)<>" " THEN SPACE=0: WORDS=WORDS+1
  IF SPACE=0 AND MID$(S$,J,1)=" " THEN SPACE=1
NEXT J
RET:
  PRINT "Число слов в строке = ";WORDS
END
```

Программа 3_04.c

```
/* Подсчет числа слов в строке */
#include <stdio.h>
#include <conio.h>
#include <string.h>
  main()
{
  char s[81], space=1, words=0, i,len;
  clrscr();
  puts("Введите строку");
  gets(s);
  len=strlen(s);
  if (len==0) goto ret;
  if (s[0]!=' ')
    {
```

```
      space=0;
      words=1;
    }
  for(i=1; i<len; i++)
    {
     if (space==1 && s[i]!=' ')
       {
         space=0;
         words++;
       }
     if (space==0 && s[i]==' ')space=1;
    }
ret:
  printf("\nЧисло слов в строке = %d",words);
  getch();
}
```

Программа 3_04.pas

```
program num_words;
{ Подсчет числа слов в строке }
label ret;
var
  s:string[81];
  space:boolean;
  words:byte;
  i,len:byte;
begin
  writeln('Введите строку');
  readln(s);
  len:=length(s);
  if len=0 then goto ret;
  space:=(s[1]=' ');
  if not space then words:=1;
  for i:=2 to len do
    begin
      if (not space)and(s[i]=' ') then space:=true;
      if (space)and(s[i]<>' ') then
        begin
```

```
          space:=false;
          inc(words);
        end;
    end;
ret:
  writeln('Число слов в строке = ',words);
  readln;
end.
```

Задание 3.05. Анализ нажатой клавиши

Составить программу, которая анализирует код нажатой клавиши и выводит его на экран. Программа должна завершать свою работу после нажатия клавиши <Esc> (код клавиши равен 27).

Совет 1 (общий)

В программе необходимо организовать бесконечный цикл, прерываемый после приема кода клавиши <Esc>. К сожалению, обработку клавиши <Enter> не удается включить в цикл обработки обычных клавиш. В операторах вывода попытка отобразить символ с кодом 13 приводит либо к возврату курсора в начало строки и затиранию ранее напечатанного текста последующим сообщением (Си, Паскаль), либо к переводу курсора в начало следующей строки (QBasic).

Совет 2 (QBasic)

Для ввода кода нажатой клавиши целесообразно использовать функцию (служебную переменную) `INKEY$`. Если нажата обычная клавиша, то `INKEY$` выдает один символ, код которого можно определить по значению функции `ASC`. При нажатии функциональной клавиши в `INKEY$` попадает два символа, первый из которых имеет нулевой код, поэтому выводить на экран надо код второго символа.

Совет 3 (Си)

Для ввода кода нажатой клавиши необходимо один (обычная клавиша) или два раза (функциональная клавиша) обратиться к функции `getch`.

Совет 4 (Паскаль)

Для ввода кода нажатой клавиши необходимо один (обычная клавиша) или два раза (функциональная клавиша) обратиться к функции `readkey`.

Программа 3_05.bas

```
REM Анализ кода нажатой клавиши
CLS
```

```
GETKEY: A$=INKEY$: IF A$="" THEN GOTO GETKEY
IF ASC(A$)=27 THEN STOP
IF LEN(A$)=1 THEN
   IF ASC(A$)=13 THEN
     PRINT "Нажата клавиша Enter с кодом = 13": GOTO GETKEY
   END IF
   PRINT "Нажата обычная клавиша '"; A$; "' с кодом ="; ASC(A$)
ELSE
   PRINT "Нажата управляющая клавиша с кодом ";ASC(RIGHT$(A$,1))
END IF
GOTO GETKEY
```

Программа 3_05.c

```
/* Анализ кода нажатой клавиши */
#include <stdio.h>
  main()
{ unsigned char ch;
  clrscr();
getkey:
  ch=getch();
  if(ch==27) exit(0);
  if(ch==13)
    {
      printf("\nНажата клавиша Enter с кодом = 13");
      goto getkey;
    }
  if(ch==0)
    {
      ch=getch();
      printf("\nНажата управляющая клавиша с кодом = %d",ch);
    }
  else printf("\nНажата обычная клавиша %c с кодом=%d",ch,ch);
  goto getkey;
}
```

Программа 3_05.pas

```
program keyboard;
{ Анализ кода нажатой клавиши }
```

```
uses Crt;
label getkey;
var
  ch:char;
begin
  clrscr;
getkey:
  ch:=readkey;
  if ord(ch)=27 then exit;
  if ord(ch)=13 then
    begin
      writeln('Нажата обычная клавиша Enter с кодом = 13');
      goto getkey;
    end;
  if ord(ch)=0 then
    begin
      ch:=readkey;
      writeln('Нажата управляющая клавиша с кодом = ',ord(ch));
    end
  else
    writeln('Нажата обычная клавиша "',ch,'" с кодом=',ord(ch));
    goto getkey;
end.
```

Программа 3_05a.pas

```
program keyboard;
{ Анализ кода нажатой клавиши }
uses Crt;
var
  ch:char;
begin
  clrscr;
  repeat
    ch:=readkey;
    if ord(ch)=13 then
      writeln('Нажата обычная клавиша Enter с кодом = 13')
    else
      if ord(ch)=0 then
```

```
        begin
          ch:=readkey;
          writeln('Нажата управляющая клавиша с кодом = ',ord(ch));
        end
      else
        writeln('Нажата обычная клавиша "',ch,'" с кодом = ',ord(ch));
  until ord(ch)=27;
end.
```

Задание 3.06. Три цвета радуги

Мнемоническая фраза *"Каждый охотник желает знать, где сидят фазаны"* используется для запоминания последовательности цветовых оттенков радуги — красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый. Составить программу, которая вводит три слова, представляющие разные цвета радуги, и выводит их на экран в том порядке, в каком они должны быть расположены в описанной выше цветовой гамме. Например, введены слова *желтый*, *красный* и *синий*. На экран их следует вывести в "правильном" порядке — *красный*, *желтый*, *синий*.

Совет 1 (общий)

Можно завести массив слов, соответствующих последовательности цветов радуги, и устроить цикл, проверяющий, не содержится ли очередной цвет среди введенных слов.

Программа 3_06.bas

```
REM Упорядочение цветов радуги
DATA "красный","оранжевый","желтый","зеленый"
DATA "голубой","синий","фиолетовый"
DIM A$(7), B$(3)
FOR I=0 TO 6: READ A$(I): NEXT I
PRINT "Введите по одному в строке 3 цвета радуги"
FOR I=0 TO 2: INPUT B$(I): NEXT I
PRINT "В радуге эти цвета следуют в таком порядке:"
FOR J=0 TO 6: FOR I = 0 TO 2
  IF A$(J)=B$(I) THEN PRINT A$(J)
NEXT I: NEXT J
END
```

Программа 3_06.c

```
/* Упорядочение цветов радуги */
#include <stdio.h>
#include <conio.h>
#include <string.h>
  main()
{
  char a[7][11]={"красный","оранжевый",
    "желтый","зеленый","голубой","синий","фиолетовый"};
  char b[3][11];
  int i,j;
  printf("\nВведите по одному в строке 3 цвета радуги\n");
  for(i=0; i<3; i++) gets(&b[i][0]);
  printf("\nВ радуге эти цвета следуют в таком порядке:\n");
  for(j=0; j<7; j++)
    for(i=0; i<3; i++)
      if(strcmp(&a[j][0],&b[i][0])==0)
        puts(&a[j][0]);
  getch();
}
```

Программа 3_06.pas

```
program raduga;
{ Упорядочение цветов радуги }
const
  a:array [1..7] of string[10]=('красный','оранжевый',
    'желтый','зеленый','голубой','синий','фиолетовый');
var
  b:array [1..3] of string[10];
  i,j:byte;
begin
  writeln('Введите по одному в строке 3 цвета радуги');
  for i:=1 to 3 do readln(b[i]);
  writeln('В радуге эти цвета следуют в таком порядке:');
  for j:=1 to 7 do
    for i:=1 to 3 do
```

```
      if a[j]=b[i] then writeln(a[j]);
  readln;
end.
```

Задание 3.07. Левый и правый прижим, центрирование текста при выводе

Составить программу, которая запрашивает у пользователя текст, содержащий не более 20-ти символов, и выводит его в рамках колонки, расположенной с 10-й по 50-ю позиции с разными вариантами прижима. Для контроля правильности работы программы можно ограничиться вводом единственного символа и выдачей на экран строки, маркирующей номера колонок.

Совет 1 (QBasic)

Для смещения текста в нужную позицию строки можно воспользоваться либо функцией `TAB`, либо переводом курсора с помощью оператора `LOCATE`.

Совет 2 (Си, Паскаль)

Для перевода курсора в нужную позицию текущей строки можно воспользоваться процедурой `gotoxy`, у которой в качестве первого аргумента задан номер начальной колонки, а в качестве второго — значение функции `wherey`, соответствующее номеру текущей строки.

Программа 3_07.bas

```
CLS
PRINT "Введите строку, содержащую не более 20 символов"
FOR I = 1 TO 8: PRINT "1234567890"; : NEXT I
INPUT "", A$
PRINT "Вывод, начиная с 10-й позиции :"
FOR I = 1 TO 8: PRINT "1234567890"; : NEXT I
PRINT TAB(10); A$
PRINT "Вывод по центру между 10-й и 50-й позициями :"
FOR I = 1 TO 8: PRINT "1234567890"; : NEXT I
PRINT TAB(10 + (50 - 10) / 2 - LEN(A$) / 2); A$
PRINT "Вывод с прижимом к 50-й позиции :"
FOR I = 1 TO 8: PRINT "1234567890"; : NEXT I
PRINT TAB(51 - LEN(A$)); A$
END
```

Программа 3_07.c

```
#include <stdio.h>
#include <conio.h>
```

```
#include <string.h>
  main()
{
  char str[20], rule[]="1234567890";
  int i;
  clrscr();
  puts("Введите строку, содержащую не более 20 символов");
  for(i=0; i<8; i++) printf("%s",rule);
  gets(str);
  puts("Вывод, начиная с 10-й позиции :");
  for(i=0; i<8; i++) printf("%s",rule);
  gotoxy(10,wherey());  puts(str);
  puts("Вывод по центру между 10-й и 50-й позициями :");
  for(i=0; i<8; i++) printf("%s",rule);
  gotoxy(10+(50-10)/2-strlen(str)/2,wherey());  puts(str);
  puts("Вывод с прижимом к 50-й позиции :");
  for(i=0; i<8; i++) printf("%s",rule);
  gotoxy(51-strlen(str),wherey());      puts(str);
  getch();
}
```

Программа 3_07.pas

```
program text_justify;
uses Crt;
const
  rule:string='1234567890';
var
  s:string[20];
  i:integer;
begin
  clrscr;
  writeln('Введите строку, содержащую не более 20 символов');
  for i:=0 to 7 do write(rule);
  readln(s);
  writeln('Вывод, начиная с 10-й позиции :');
  for i:=0 to 7 do write(rule);
  gotoxy(10,wherey);  writeln(s);
  writeln('Вывод по центру между 10-й и 50-й позициями :');
  for i:=0 to 7 do write(rule);
```

```
  gotoxy(10+(50-10) div 2 - length(s) div 2,wherey);  writeln(s);
  writeln('Вывод с прижимом к 50-й позиции :');
  for i:=0 to 7 do write(rule);
  gotoxy(51-length(s),wherey);      writeln(s);
  readln;
end.
```

Задание 3.08. Сравнение строк с игнорированием пробелов

Написать подпрограмму-функцию `compare`, имеющую своими аргументами две строки, которая сравнивает их, игнорируя пробелы между словами и разницу между кодами больших и малых букв. Функция должна возвращать следующие значения:

`1`, если первая строка "больше" второй;

`-1`, если первая строка "меньше" второй;

`0`, если обе строки равны.

Совет 1 (общий)

Самым простым по реализации является алгоритм, в котором сначала коды всех малых букв заменяются кодами их больших эквивалентов, а затем из строк удаляются пробелы. Для выполнения первой процедуры можно воспользоваться функцией `up(str)` — см. программу 3_02.

Программа 3_08.bas

```
REM Сравнение строк с игнорированием пробелов
DECLARE SUB UP(A$)
DECLARE FUNCTION COMPARE(B$,C$)
PRINT "Введите первую строку"
INPUT "",A1$
PRINT "Введите вторую строку"
INPUT "",A2$
K=COMPARE(A1$,A2$)
IF K=1 THEN PRINT "Первая строка 'больше'"
IF K=-1 THEN PRINT "Первая строка 'меньше'"
IF K=0 THEN PRINT "Обе строки равны"
END

FUNCTION COMPARE(B$,C$)
DIM B1$,C1$
```

```
UP B$: UP C$
FOR J=1 TO LEN(B$)
  IF MID$(B$,J,1)<>" " THEN B1$=B1$+MID$(B$,J,1)
NEXT J
FOR J=1 TO LEN(C$)
  IF MID$(C$,J,1)<>" " THEN C1$=C1$+MID$(C$,J,1)
NEXT J
IF B1$>C1$ THEN COMPARE=1
IF B1$=C1$ THEN COMPARE=0
IF B1$<C1$ THEN COMPARE=-1
END FUNCTION

SUB UP(A$)
FOR J=1 TO LEN(A$)
  SELECT CASE MID$(A$,J,1)
    CASE "a" TO "z": MID$(A$,J,1)=CHR$(ASC(MID$(A$,J,1))-32)
    CASE "а" TO "п": MID$(A$,J,1)=CHR$(ASC(MID$(A$,J,1))-32)
    CASE "р" TO "я": MID$(A$,J,1)=CHR$(ASC(MID$(A$,J,1))-80)
    CASE "ё": MID$(A$,J,1)="Ё"
  END SELECT
NEXT J
END SUB
```

Программа 3_08.c

```
/* Сравнение строк с игнорированием пробелов */
#include <stdio.h>
#include <conio.h>
#include <string.h>
char *up(char *a);
int compare(char *s1,char *s2);

  void main()
{
  char s1[80],s2[80];
  puts("Введите первую строку");
  gets(s1);
  puts("Введите вторую строку");
  gets(s2);
```

```
  switch(compare(s1,s2))
  {
    case  1: puts("Первая строка 'больше'");break;
    case -1: puts("Первая строка 'меньше'");break;
    case  0: puts("Обе строки равны");
  }
  getch();
}
  char *up(char *a)
{ /* Замена кодов малых букв кодами больших */
  static unsigned char b[80];
  int j;
  strcpy(b,a);
  for(j=0; j<strlen(a); j++)
    {
     if(97<=b[j] && b[j]<=122) b[j]-=32;
     if(160<=b[j] && b[j]<=175) b[j]-=32;
     if (224<=b[j] && b[j]<=239) b[j]-=80;
     if (241==b[j]) b[j]--;
    }
 return b;
}
int compare(char *s1,char *s2)
{
  char ss1[80],ss2[80];
  char j,k;
  strcpy(s1,up(s1));    /* Замена малых букв большими */
  strcpy(s2,up(s2));    /* с записью на то же место */
  for(j=0,k=0; j<strlen(s1); j++)
    if (s1[j] != ' ') ss1[k++]=s1[j]; /* Извлекаем непробелы */
  ss1[k]=0x0;                         /* Добавили признак конца */
  for(j=0,k=0; j<strlen(s2); j++)
    if(s2[j]!=' ') ss2[k++]=s2[j];
  ss2[k]=0x0;
  k=strcmp(ss1,ss2);    /* Сравнили преобразованные строки */
  if(k>0)return 1;
  if(k<0)return -1;
  return 0;
}
```

Программа 3_08.pas

```
program comp_string;
{ Сравнение строк с игнорированием пробелов }
var
  s1,s2:string;
function up(a:string):string;
var
  j:integer;
begin
  for j:=1 to length(a) do
    case a[j] of
      'a'..'z': a[j]:=chr(ord(a[j])-32);
      'а'..'п': a[j]:= chr(ord(a[j])-32);
      'р'..'я': a[j]:= chr(ord(a[j])-80);
      'ё': a[j]:='Ё';
    end;
  up:=a;
end;
function compare(s1,s2:string):integer;
var
  ss1,ss2:string;
  j:byte;
begin
  s1:=up(s1);
  s2:=up(s2);
  ss1:='';
  for j:=1 to length(s1) do
    if s1[j]<>' ' then ss1:=ss1+s1[j];
  ss2:='';
  for j:=1 to length(s2) do
    if s2[j]<>' ' then ss2:=ss2+s2[j];
  compare:=0;
  if ss1>ss2 then compare:=1;
  if ss1<ss2 then compare:=-1;
end;
begin
  writeln('Введите первую строку');
  readln(s1);
```

```
  writeln('Введите вторую строку');
  readln(s2);
  case compare(s1,s2) of
     1: writeln('Первая строка "больше"');
    -1: writeln('Первая строка "меньше"');
     0: writeln('Обе строки равны');
  end;
  readln;
end.
```

Задание 3.09. Разноцветный текст

Написать программу, которая выводит на экран текст, меняя цвет букв и цвет фона.

Совет 1 (общий)

В приведенных ниже текстах программ в пределах одного экрана демонстрируются две возможности. Во-первых, выводится строка, все символы которой окрашены в разные цвета. Для этого приходится каждый символ выдавать на экран отдельным оператором, предваряя вывод переустановкой цвета переднего плана. Выбранное для этой цели слово "ПРОГРАММИРОВАНИЕ" содержит ровно 16 букв, но каждый раз при очередной смене цвета фона одна из букв отображаемого слова имеет такой же цвет и не будет видна. Во-вторых, выводится строка, все символы которой имеют одинаковый цвет, не совпадающий с цветом фона. С помощью такой программы можно поэкспериментировать в подборе приятных цветовых сочетаний.

Программа 3_09.bas

```
REM Разноцветный текст
A$ = "ПРОГРАММИРОВАНИЕ"
CLS
'Отображение разноцветных букв на допустимой фоновой гамме
FOR CF=0 TO 7
  FOR CS=0 TO 15
    COLOR CS,CF
    LOCATE CF+1,2*CF+CS+1
    PRINT MID$(A$,CS+1,1)
  NEXT CS
NEXT CF
'Отображение мигающих разноцветных букв
FOR CF=0 TO 7
```

```
  FOR CS=0 TO 15
    COLOR CS+16,CF
    LOCATE CF+1,2*CF+CS+41
    PRINT MID$(A$,CS+1,1)
  NEXT CS
NEXT CF
```

Программа 3_09.c

```
/* Разноцветный текст */
#include <conio.h>
  main()
{
  int i;
  textbackground(0);
  clrscr();
  for(i=0; i<24; i++)
   {
     gotoxy(2*i+1,i+1);
     textcolor(128+i);
     textbackground(i+2);
     cprintf("Цветовая гамма в текстовом режиме");
   }
  getch();
}
```

Программа 3_09a.c

```
/* Разноцветный текст */
#include <conio.h>
  main()
{
  int i;
  textbackground(0);
  clrscr();
  for(i=0; i<24; i++)
    {
      gotoxy(2*i+1,i+1);
      textattr(128+i + ((i+1) << 4));
      cprintf("Цветовая гамма в текстовом режиме");
```

```
    }
  getch();
}
```

Программа 3_09.pas

```
program color1;
{ Разноцветный текст }
uses crt;
var
  i:integer;
begin
  textbackground(0);
  clrscr;
  for i:=0 to 23 do
    begin
      gotoxy(2*i+1,i+1);
      textcolor(128+i);
      textbackground(i+1);
      writeln('Тест цветовой гаммы ');
    end;
  readln;
end.
```

Задание 3.10. Преобразование обычной дроби в десятичную

Составить функцию символьного (строкового) типа, преобразующую два своих целочисленных аргумента — числитель `m` и знаменатель `n` правильной дроби (`m < n < 100`) в строку, представляющую запись десятичной дроби. Для бесконечной дроби период следует заключить в круглые скобки. Например:

```
m=3 n=5      значение функции - "0.6"
m=1 n=6      значение функции - "0.1(6)"
```

Совет 1 (общий)

Очевидно, что количество цифр в десятичной дроби не превосходит 100, т. к. при последовательном делении на `n` мы можем получить не более чем `n` разных остатков (от 0 до `n-1`). Поэтому разумно завести массив для хранения остатков и при получении очередного остатка, отличного от 0, проверять, нет ли такого же среди ранее получавшихся. Как только новый остаток совпадет с одним из предыдущих, будет обнаружен период.

Совет 2 (Си)

К сожалению, среди строковых функций Си, включенных в заголовочный файл `string.h`, нет процедуры вставки символа в строку. Поэтому нам придется сдвигать вправо на одну позицию часть символов от начала периода и вставлять открывающуюся скобку. Не следует забывать и о вставке признака конца строки.

Программа 3_10.bas

```
REM Преобразование обычной дроби в десятичную
DECLARE FUNCTION FRAC2STR$(M%, N%)
CLS : DEFINT A-Z
INPUT "Введите числитель и знаменатель дроби : ",M,N
PRINT M;"/";N;"= ";FRAC2STR$(M,N)
END

DEFSNG A-Z
FUNCTION FRAC2STR$(M AS INTEGER,N AS INTEGER)
DIM S AS STRING, REST(100) AS INTEGER
DEFINT A-Z
I=3: S="0."
DO
  Q=M*10\N : P=M*10 MOD N: REST(I)=P
  IF P=0 THEN
    FRAC2STR=S+CHR$(Q+48)
    EXIT FUNCTION
  END IF
  FOR J=3 TO I-1
    IF REST(J)=P THEN
      FRAC2STR=LEFT$(S,J-1)+"("+RIGHT$(S,LEN(S)-J+1)+")"
       EXIT FUNCTION
    END IF
  NEXT J
  S=S+CHR$(Q+48): I=I+1: M=P
LOOP UNTIL P=0
END FUNCTION
```

Программа 3_10.c

```
/* Преобразование обычной дроби в десятичную */
#include <stdio.h>
```

```
#include <conio.h>
char *frac_to_str(char m,char n);
  main()
{
  char m,n;
  printf("\nВведите числитель и знаменатель дроби : ");
  scanf("%d %d",&m,&n);
  printf("\n%d/%d=%s",m,n,frac_to_str(m,n));
  getch();
}
/*----------------------------------------*/
char *frac_to_str(char m,char n)
{
  int i=2,j,p,q;
  char rest[100];
  static char s[100]="0.";
  do
    {
      q=m*10/n;
      p=m*10%n;
      rest[i]=p;
      if(p==0) { s[i]=q+48;   s[i+1]=0x0;  return s; }
      for(j=2; j<i; j++)
        if(rest[j]==p)
         {
           for(p=i; p>=j; p--)
             s[p+1]=s[p];
           s[j]='(';
           s[i+1]=')';
           s[i+2]=0x0;
           return s;
         }
      s[i]=q+48;
      i++;
      m=p;
    }
  while (p!=0);
}
```

Программа 3_10.pas

```
program drobi;
{ Преобразование обычной дроби в десятичную }
var
  m,n:byte;
function frac_to_str(m,n:byte):string;
  var
    i,j,p,q:word;
    s:string;
    rest:array [1..100] of byte;
  begin
    s:='0.';
    i:=1;
    repeat
      q:=m*10 div n;
      p:=m*10 mod n;
      rest[i]:=p;
      if p=0 then begin
         frac_to_str:=s+chr(q+48);  exit;
      end;
      for j:=1 to i-1 do
        if p=rest[j] then begin
           insert('(',s,j+2);
           frac_to_str:=s+')';
           exit;
        end;
      inc(i);
      s:=s+chr(q+48);
      m:=p;
    until p=0;
  end;
begin
  writeln('Введите числитель и знаменатель дроби :');
  readln(m,n);
  writeln(m,'/',n,'=',frac_to_str(m,n));
  readln;
end.
```

Задание 3.11. Перевод чисел в римскую систему счисления

Составить функцию символьного (строкового) типа, аргументом которой является натуральное число из интервала [1, 3999]. Функция должна возвращать строку с эквивалентным значением в формате римской системы счисления. Напомним, что в римской системе счисления используются большие латинские буквы — `M` (1000), `D` (500), `C` (100), `L` (50), `X` (10), `V` (5) и `I` (1). Если меньшая "цифра" находится левее "большей", то она вычитается (например: `IV = V - I = 4`, `IX = X - I = 9`, `XL = L - X = 40`). Если меньшая "цифра" расположена правее, то она прибавляется (например: `VI = = V + I = 6`, `XI = X + I = 11`, `LX = L + X = 60`). Для некоторых "цифр" римская система допускает наличие до трех идущих подряд одинаковых цифр — `II = I + I = 2`, `III = I + I + I = 3`, `XXX = X + X + X = 30`, `CCC = = C + C + C = 300`, `MMM = M + M + M + 3000`.

Совет 1 (общий)

Заведем два массива — символьный с наиболее характерными сочетаниями римских "цифр" и числовой с соответствующими значениями этих сочетаний. Оба массива разумно расположить в порядке убывания числовых значений (табл. 3.1).

Таблица 3.1. *Расположение массива в порядке убывания числовых названий*

Индекс элемента	Символьный массив	Числовой массив
0	M	1000
1	CM	900
2	D	500
3	CD	400
4	C	100
5	XC	90
6	L	50
7	XL	40
8	X	10
9	IX	9
10	V	5
11	IV	4
12	I	1

Начинаем вычитать из переводимого числа первый элемент числового массива и вычитаем до тех пор, пока результат остается положительным. Каж-

дый раз, когда вычитание возможно, к результирующей строке присоединяем соответствующее символьное значение. И так продолжаем экспериментировать с каждым элементом числового массива. Кстати, сумма всех элементов равна 3999, чем и объясняется ограничение на допустимый диапазон обрабатываемых чисел.

Программа 3_11.bas

```
REM Перевод чисел в римскую систему
DECLARE FUNCTION torome$ (M%)
REM Перевод арабских чисел в римскую систему счисления
DEFINT A-Z
COMMON SHARED ND()
COMMON SHARED SD$()
DATA 1,4,5,9,10,40,50,90,100,400,500,900,1000
DIM ND(13)
FOR J=0 TO 12: READ ND(J): NEXT J
DATA I,IV,V,IX,X,XL,L,XC,C,CD,D,CM,M
DIM SD$(13)
FOR J=0 TO 12: READ SD$(J): NEXT J
INPUT "Введите целое число от 1 до 3999 : ", N
IF N<1 OR N>3999 THEN PRINT "Число вне диапазона": END
PRINT "В римской системе счисления "; N;" = ";torome$(N)
END

FUNCTION torome$ (M)
SHARED ND(), SD$()
S$=""
FOR K=12 TO 0 STEP -1
  WHILE ND(K)<=M
    M=M-ND(K): S$=S$+SD$(K)
    IF M=0 THEN EXIT FOR
  WEND
NEXT K
torome$=S$
END FUNCTION
```

Программа 3_11.c

```
/* Перевод чисел в римскую систему счисления */
#include <stdio.h>
```

```
#include <conio.h>
#include <string.h>
char *to_rome(int n);
  main()
{
  int N;
  printf("\nВведите целое число от 1 до 3999 : ");
  scanf("%d",&N);
  if(N<0 || N>3999)
    {
      printf("\nЧисло вне диапазона");
      getch(); exit(0);
    }
  printf("\nВ римской системе счисления %d = %s",N,to_rome(N));
  getch();
}
/*---------------------------------------------*/
char *to_rome(int n)
{
  int k;
  static char s[20]="";
  int nd[13]={1,4,5,9,10,40,50,90,100,400,500,900,1000};
  char *sd[13]={"I","IV","V","IX","X","XL",
                "L","XC","C","CD","D","CM","M"};
  for(k=12; k>=0; k--)
    {
      while(nd[k]<=n)
        {
          n=n-nd[k];
          strcat(s,sd[k]);
          if(n==0) break;
        }
    }
  return s;
}
```

Программа 3_11.pas

```
program in_rome;
{Перевод чисел в римскую систему счисления}
```

```
var
  N:1..3999;
function to_rome(n:integer):string;
const
  nd:array [1..13] of integer=
    (1,4,5,9,10,40,50,90,100,400,500,900,1000);
  sd:array [1..13] of string=
    ('I','IV','V','IX','X','XL',
     'L','XC','C','CD','D','CM','M');
var
  k:integer;
  s:string;
begin
  s:='';
  for k:=13 downto 1 do
  while nd[k]<=n do begin
    n:=n-nd[k];
    s:=s+sd[k];
    if n=0 then break;
  end;
  to_rome:=s;
end;
begin
  write('Введите целое число от 1 до 3999 : ');
  readln(N);
  writeln('В римской системе счисления ',N,' = ',to_rome(N));
  readln;
end.
```

Задание 3.12. Перевод чисел из римской системы счисления

Составить программу обратного преобразования из строки с записью числа в римской системе счисления в обычное число.

Программа 3_12.bas

```
REM Перевод чисел из римской системы счисления в арабскую
DEFINT A-Z
DATA 1,4,5,9,10,40,50,90,100,400,500,900,1000
DIM ND(13)
```

```
FOR J=0 TO 12: READ ND(J): NEXT J
DATA I,IV,V,IX,X,XL,L,XC,C,CD,D,CM,M
DIM SD$(13)
FOR J=0 TO 12: READ SD$(J): NEXT J
INPUT "Введите число в римской системе счисления : ", R$
J=1: M=0
100 :
FOR K=12 TO 0 STEP -1
  N=LEN(SD$(K))
  IF MID$(R$,J,N)=SD$(K) THEN
    M=M+ND(K): J=J+N: GOTO 100
  END IF
  IF J>LEN(R$) THEN EXIT FOR
NEXT K
PRINT "В арабской системе счисления ";R$;" = ";M
END
```

Программа 3_12a.bas

```
REM Перевод чисел из римской системы счисления в арабскую
DATA 1000,900,500,400,100,90,50,40,10,9,5,4,1
DIM ND(13): FOR J=0 TO 12: READ ND(J): NEXT J
DATA M,CM,D,CD,C,XC,L,XL,X,IX,V,IV,I
DIM SD$(13): FOR J=0 TO 12: READ SD$(J): NEXT J
INPUT "Введите число в римской системе счисления : ",R$
FOR J=1 TO LEN(R$)
  FOR K=0 TO 12
    N=LEN(SD$(K))
    IF MID$(R$,J,N)=SD$(K) THEN
      M=M+ND(K): J=J+1: K=K-1
    END IF
  NEXT K
NEXT J
PRINT "В арабской системе счисления ";R$;" = ";M
END
```

Программа 3_12.c

```
/* Перевод чисел из римской системы счисления в арабскую */
#include <stdio.h>
```

```
#include <conio.h>
#include <string.h>
  main()
{
  int nd[13]={1000,900,500,400,100,90,50,40,10,9,5,4,1};
  char *sd[13]={"M","CM","D","CD","C","XC","L","XL","X",
                "IX","V","IV","I"};
  char r[20],s[3];
  int j,k,n,m=0;
  printf("\nВведите число в римской системе счисления : ");
  scanf("%s", r);
  for(j=0; j<strlen(r); j++)
    for(k=0; k<13; k++)
      {
        n=strlen(sd[k]);
        strncpy(s,&r[j],n); s[n]=0;
        if(strcmp(s,sd[k])==0)
         { m += nd[k]; j++; k--; }
      }
  printf("\nВ арабской системе счисления %s = %d",r,m);
  getch();
}
```

Программа 3_12.pas

```
{ Перевод чисел из римской системы счисления в арабскую }
program from_rome;
const
  nd:array [1..13] of integer=(1000,900,500,400,100,
                               90,50,40,10,9,5,4,1);
  sd:array [1..13] of string=('M','CM','D','CD','C',
                 'XC','L','XL','X','IX','V','IV','I');
var
  r,s:string;
  j,k,n,m: integer;
begin
  m:=0;
  write('Введите число в римской системе счисления : ');
  readln(r);
```

```
  j:=1;
  while j<=length(r) do
  begin
    k:=1;
    while k<=13 do
    begin
      n:=length(sd[k]);
      s:=copy(r,j,n);
      if s=sd[k] then begin
        inc(m,nd[k]); inc(j); dec(k);
      end;
      inc(k);
    end;
    inc(j);
  end;
  writeln('В арабской системе счисления ',r,' = ',m);
  readln;
end.
```

Задание 3.13. Вхождение строки с разрядкой

Строка `S2` может входить в более длинную строку `S1` в общепринятом смысле, когда непрерывная подпоследовательность символов строки `S1` совпадает с цепочкой символов `S2`. Однако возможна и другая ситуация, когда строка `S2` входит в строку `S1` с разрядкой. При этом символы `S2` с равномерным шагом через 1, 2, 3 или более символов встречаются в строке `S1`.

Например, для строк `S1="ABCBEEC"` и `S2="ABC"` имеют место обычное (с нулевым шагом разрядки) вхождение (символы 1, 2 и 3 в строке `S1`) и вхождение с разрядкой в 2 символа (символы 1, 4 и 7 в строке `S1`).

Составить программу, которая:

- запрашивает и вводит анализируемые строки `S1` и `S2`;
- определяет, входит ли строка `S2` в `S1`, выдает шаг разрядки и номера соответствующих позиций в строке `S1`.

Задача программы — определить все возможные варианты вхождения `S2` в `S1`.

Совет 1 (общий)

Идея поиска довольно проста. Обозначим через `h` шаг по индексу в строке `S1`, с которым выбираются символы, сравниваемые с символами `S2`. Для нулевой разрядки `h=1`, для выборки через символ `h=2` и т. д. Очевидно, что начинать

сравнение надо с символа `S1[I]`, совпадающего с первым символом `S2`. Однако реализация этой идеи может быть разной. Например, из строки `S1` можно извлечь с заданной разрядкой нужное число символов и результат сравнить с `S2`. Более эффективно совместить эти две операции и сравнивать извлекаемый из `S1` символ с очередным символом `S2`, прекращая выборку при первом несовпадении. Указанный подход был реализован победителем областной студенческой олимпиады В. Мартьяновым (апрель 2000 г.), на базе программы которого и построены приведенные ниже примеры. Наиболее оригинальной, на наш взгляд, является проверка условия (`h=1`) или (`L2>1`). Здесь `L2` — длина второй строки.

Программа 3_13.bas

```
REM  Поиск вхождения S2 в S1 с разрядкой
CLS
INPUT "Введите S1: ", S1$: L1=LEN(S1$)
INPUT "Введите S2: ", S2$: L2=LEN(S2$)
FOR H=1 TO L1
  IF (H=1) OR (L2>1) THEN
     FOR I=1 TO L1-(L2-1)*H
       IF MID$(S1$,I,1)=MID$(S2$,1,1) THEN
         FOR J=2 TO L2
           IF MID$(S1$,I+(J-1)*H,1)<>MID$(S2$,J,1) THEN GOTO m1
         NEXT J
         PRINT "Шаг: ";H-1,"Позиции: ";I;
          FOR K = 1 TO L2 - 1: PRINT "-"; I + K * H; : NEXT K: PRINT
          K = 1
m1:   END IF
     NEXT I
   END IF
NEXT H
IF K = 0 THEN PRINT "Строка S2 не входит в S1"
END
```

Программа 3_13.c

```
/* Поиск вхождения S2 в S1 с разрядкой */
#include <stdio.h>
#include <conio.h>
#include <string.h>
  main()
```

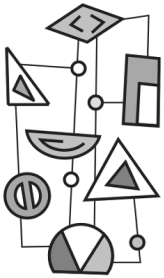
```
{
  char s1[80],s2[80];
  int i,j,k=0,h,L1,L2;
  clrscr();
  printf("Введите S1: "); scanf("%s",s1); L1=strlen(s1);
  printf("Введите S2: "); scanf("%s",s2); L2=strlen(s2);
  for(h=1; h<=L1; h++)
    {if(h==1 || L2>1)
       {for(i=0; i<L1-(L2-1)*h; i++)
          {if(s1[i]==s2[0])
             {for(j=1; j<L2; j++)
                if(s1[i+j*h]!=s2[j]) goto m1;
                printf("\nШаг: %2d  Позиции: %d",h-1,i+1);
                for(k=1; k<L2; k++) printf("-%d",i+1+k*h);
        m1:;}
          }
       }
   }
  if(k==0) printf("Строка S2 не входит в S1");
  getch();
}
```

Программа 3_13.pas

```
program red_str;
{Поиск вхождения S2 в S1 с разрядкой}
uses crt;
var
  s1,s2:string;
  i,j,k,h,L1,L2:integer;
label
  1;
begin
  clrscr;
  write('Введите S1: '); readln (s1); L1:=length(s1);
  write('Введите S2: '); readln (s2); L2:=length(s2);
  k:=0;
  for h:=1 to L1 do
    if (h=1)or(L2>1) then
```

```
      for i:=1 to L1-(L2-1)*h do
        if s1[i]=s2[1] then
        begin
          for j:=2 to L2 do
            if s1[i+(j-1)*h]<>s2[j] then goto 1;
          write ('Шаг: ',h-1:2,'  Позиции: ',i);
          for k:=1 to L2-1 do write('-',i+k*h);
          writeln;
          k:=1;
1:      end;
  if k=0 then writeln('Строка S2 не входит в S1');
  readln;
end.
```

Глава 4



Работа с массивами

Как правило, массив представляет собой набор однотипных данных, расположенных в оперативной памяти таким образом, чтобы по индексам элементов можно было легко вычислить адрес соответствующего значения. Например, пусть одномерный массив `A` состоит из элементов, расположенных в памяти подряд по возрастанию индексов и каждый элемент занимает по `k` байт. Тогда адрес `i`-го элемента вычисляется по формуле:

```
адрес(A[i]) = адрес(A[0]) + i * k
```

Если мы имеем дело с двумерным массивом `B` размерности `M×N`, расположенным в памяти по строкам, то адрес элемента `B[i,j]` вычисляется по формуле:

```
адрес(B[i,j]) = адрес(B[0,0]) + (i * N + j) * k
```

Приведенные выше формулы незначительно усложняются в тех случаях, когда начальные индексы отсчитываются не от нуля, что характерно для Паскаля.

Использование массивов позволяет заменить большое количество индивидуальных имен каждого объекта одним групповым именем набора данных, вслед за которым в круглых (QBasic) или квадратных (Си, Паскаль) скобках задаются один или несколько индексов, определяющих местоположение требуемого значения. Естественно, что такая возможность упрощает и массовую обработку данных в соответствующих циклах программы.

В большинстве задач приходится иметь дело с массивами, элементами которых являются числа того или иного типа, символы и строки фиксированной длины. Однако приходится обрабатывать и битовые массивы, каждый элемент которых представлен одним или несколькими двоичными разрядами, соответствующими кодам цветности пикселов графических изображений. Некоторые системы программирования позволяют работать с массивами, состоящими из неоднородных элементов. Например, с одномерными массивами, каждый элемент которых представлен строкой переменной длины.

Объявление массивов

В QBasic для объявления массивов и одновременного отведения памяти под хранение их элементов используется оператор `DIM`:

```
DIM A(10),B(2 TO 8,3 TO 5),C(3,2,6)
```

В простом объявлении указывается максимальный индекс и, поскольку минимальный индекс по умолчанию равен 0, то в массиве `A`, например, содержится не 10, а 11 элементов. Конструкция "`qq TO kk`" позволяет одновременно задать и минимальный, и максимальный индексы.

Интерпретатор QBasic существенно отличается от многих других систем программирования, где операторы объявления относятся к разряду невыполняемых и обрабатываются только на стадии компиляции программы. Здесь вполне возможны фрагменты следующего типа:

```
INPUT "Введите размерность массива F", N
DIM F(N)
```

После работы с этим массивом в программе может встретиться следующая пара операторов:

```
INPUT "Введите новую размерность массива F",M
REDIM F(M)
```

И новое значение верхней границы может быть как меньше предыдущего, так и больше. Массивы подобного рода в QBasic принято называть динамическими. В отличие от этого оператор `DIM` с конкретными числовыми границами формирует статические массивы, значения элементов которых сохраняются и после завершения работы программы.

Ни Си, ни Паскаль таких "вольностей" не позволяют. В них, конечно, можно определить массив достаточно большого размера и использовать только его часть. Но память будет выделена под весь массив, включая и неиспользуемый в конкретном варианте остаток.

Объявление массивов в Си чем-то особым не отличается от других алгоритмических языков. Разве что в задании многомерных массивов каждый индекс записывается в отдельных квадратных скобках:

```
#define N_max 50
  char a1[20], a2[5][80];
  int b1[25],b2[N_max];
```

Индексы в Си всегда отсчитываются от 0, так что, например, в массиве `b1` можно манипулировать с элементами `b1[0]`, `b1[1]`, ..., `b1[24]`. Элемент `b1[25]` массиву `b1` уже не принадлежит и попытка записи в него может привести к непредсказуемым последствиям.

В Паскале существует несколько вариантов для объявления многомерных массивов. Например, целочисленная матрица `f`, содержащая `q` строк и `k` столбцов (значения `q` и `k` предварительно должны быть объявлены константами), может быть включена в текст программы одним из следующих описаний:

```
type
  mat_q_k=array [1..q,1..k] of integer;
```

```
var
  f:mat_q_k;
```

или

```
var
  f:array [1..q,1..k] of integer;
```

или

```
var
  f:array [1..q][1..k] of integer;
```

или

```
var
  f:array [1..q] of array [1..k] of integer;
```

Еще одна дополнительная особенность Паскаля заключается в том, что в качестве индексов могут использоваться не только числа, но и любые данные интервального или перечислимого типов. Например, буквы:

```
var
  ch:array ['A'..'Z'] of integer;
  str:string;
.................
begin
..............
  inc(ch[str[j]]);
...............
```

Приведенный фрагмент наиболее простым способом позволяет подсчитать частоту появления тех или иных букв в обрабатываемом тексте.

Инициализация массивов

Термином "инициализация" обозначают возможность задать начальные значения элементов массива без программирования соответствующих действий. Например, не прибегая к программным средствам типа присвоения значений в цикле или считывания данных из внешнего источника (файл, клавиатура, блок данных).

Средства такого рода имеются только в Си и Паскале. Здесь одновременно с объявлением массива можно задать начальные значения всех элементов массива или только нескольких первых его компонент:

```
  char a[7]="Привет";
  char b[7]={'П','р','и','в','е','т',0x00};
  char c[]="Привет";
```

```
int d[10]={1,2,3,4};
int f[2][3]={{1,2,3},
             {4,5,6}};
```

В символьном массиве `a` формируются значения семи компонент, первые шесть из которых соответствуют кодам отображаемых символов заданной строки. Признак конца строки — байт с нулевым кодом — заносится в элемент `a[6]` автоматически. В отличие от этого в массиве `b`, где значения элементов формируются посимвольно, такой признак конца программист должен задавать сам. Обратите внимание на третью строку, в которой отсутствует указание о размере массива. Компилятор Си сам сформирует нужное значение по количеству инициализирующих данных. В нашем случае под массив `c` будет отведено 7 байт, включая последний байт с нулевым кодом, завершающий каждую строку.

В Паскале инициализация производится в разделе `const`:

```
const
  a:string[7]='Привет';
  d:array [1..10] of integer=(1,2,3,4);
  f:array [1..2,1..3] of integer=((1,2,3),
                                  (4,5,6));
```

Инициализация глобальных и статических массивов в Си производится один раз при загрузке программы. Локальные массивы в функциях языка Си инициализируются при каждом обращении к функции.

Статические и динамические массивы

Статическим массивом называют набор данных, для хранения которого перед началом функционирования программы выделяется фиксированное место в памяти, освобождаемое после завершения работы программы.

QBasic трактует это понятие несколько шире, сохраняя значения статических массивов и после останова или полного завершения работы программы. Не выходя из интегрированной среды, можно посмотреть значения элементов статических массивов. В отличие от этого, место для хранения динамических массивов выделяется и освобождается в процессе выполнения программы. В одних случаях эти операции осуществляются системой автоматически. Например, когда отводится память для хранения локальных массивов в процедурах и функциях. В других случаях пользователю предоставляется возможность запросить участок памяти нужного размера и в дальнейшем — освободить его. Только таким способом в программах на Си и Паскале можно завести массив переменного размера.

В системе QBasic по умолчанию массивы с конкретными числовыми границами считаются статическими, а с переменными границами —динамическими. Однако, если первой строкой программы задана метакоманда вида `REM $STATIC` или `REM $DYNAMIC`, то все массивы в программе будут заводиться либо как статические, либо как динамические. По оператору `ERASE v1,v2,...` статические массивы "чистятся", а динамические массивы освобождают занимаемую память. Чистка массивов подразумевает засылку нулевых значений во все элементы числовых массивов и засылку "пустых" (нулевых) строк в элементы массива символьного типа. Оператор `REDIM` позволяет переопределить размеры ранее объявленных динамических массивов, не изменяя количество приписанных им индексов. Одновременно с этим происходит чистка переобъявляемых массивов.

В Си для запроса и освобождения памяти используются следующие системные функции:

```
q=(тип_q *)calloc(n_el,s_el);      //запрос памяти с очисткой;
q=(тип_q *)farcalloc(n_el,s_el);   //запрос памяти с очисткой;
q=(тип_q *)malloc(n_byte);         //запрос памяти в ближней "куче"
q=(тип_q *)farmalloc(n_byte);      //запрос памяти в дальней "куче"
q_new=realloc(q_old,n_byte);       //изменение размера блока
q_new=farrealloc(q_old,n_byte);    //изменение размера блока
free(q);                           //освобождение памяти
farfree(q);                        //освобождение памяти
```

В приведенных выше обращениях `q` обозначает указатель на тип данных элементов массива, заменяющий имя массива. Параметры `n_el` и `s_el` задают соответственно количество элементов в массиве и длину каждого элемента в байтах. Параметр `n_byte` определяет количество запрашиваемых байтов.

Максимальный размер сегмента памяти, предоставляемого в ближней "куче", равен 65 521 байт. Добавка `far` означает, что программа использует дальние указатели типа `far` или `huge`, которые позволяют адресоваться к дальней "куче" и использовать сегменты размером более 64 Кбайт. Любая функция выделения памяти возвращает начальный адрес или "нулевой" указатель (`NULL`) в случае отсутствия свободной памяти запрашиваемого размера. Для того чтобы нормально работать с предоставленным фрагментом памяти, возвращаемый адрес обязательно должен быть приведен к типу указателя `q`.

Функция `realloc` (`farrealloc`) позволяет перераспределить ранее выделенную память. При этом новый размер массива может быть как меньше предыдущего, так и больше. Если система выделит память в новом месте, то все предыдущие значения, к которым программа обращалась по указателю `q_old`, будут переписаны на новое место автоматически.

В новых версиях Borland C++ появились две более удобные процедуры для запроса и освобождения памяти, не нуждающиеся в дополнительном указании о приведении типа возвращаемого адреса:

```
q=new тип[n_el];     //запрос памяти под массив из n_el элементов;
q=new тип;           //запрос памяти под скалярную переменную;
delete q[n_el];      //освобождение памяти, занятой массивом;
delete q;            //освобождение памяти, занятой массивом или
                     //скалярной переменной;
```

Для формирования в Паскале динамического массива с элементами определенного типа с таким массивом надо связать соответствующий указатель и обратиться к процедуре `New`:

```
type
  mas=array [1..200] of integer;
  point=^mas;
var
  p:point;
  j:integer;
........
begin
  New(p);
  for j:=1 to 200 do
    readln(p^[j]);
................
```

Запрошенная таким образом память освобождается процедурой `Dispose(p)`. Для запроса и освобождения памяти под разнородные данные в Паскале можно использовать процедуры `GetMem(q,n_byte)` и `FreeMem(q,n_byte)`. Указатель `q`, которому присваивается начальный адрес выделенного сегмента, в данном случае представлен нетипизированным указателем (`q:pointer;`).

Как в Си, так и в Паскале, процедура освобождения памяти не чистит указатель, "смотревший" на начало возвращаемого сегмента. Запись по такому указателю после возврата памяти приводит к трудно обнаруживаемым ошибкам. Поэтому к правилам "хорошего тона" в программировании относится и сброс указателей после возврата динамически запрашивавшейся памяти:

```
q=NULL;          //так это делается в Си
p:=nil;          //так это делается на Паскале
```

Не менее хорошее правило заключается и в проверке, выделена ли запрашиваемая память после обращения к соответствующей процедуре. Например, в Си, не контролирующем запись по нулевому адресу, после стирания

нескольких первых элементов несуществующего массива происходит зависание операционной системы.

Массивы в качестве параметров процедур и функций

Программисты, работавшие на Фортране, с грустью вспоминают времена, когда передача массива любой размерности в подпрограмму или функцию никаких забот не вызывала. Достаточно было написать нечто похожее на:

```
SUBROUTINE MATMULT(A,B,C,N)
  REAL A(N,N),B(N,N),C(N,N),D
  DO 1 J=1,N
  DO 1 K=1,N
    D=0
    DO 2 L=1,N
2   D=D+A(J,L)*B(L,K)
1   C(J,K)=D
END
```

И дальше работать с элементами массивов, ни о чем не задумываясь. Подпрограмма такого вида позволяет перемножать квадратные матрицы любого размера.

Массивы-параметры в подпрограммах и функциях QBasic

Ближе всего к идеологии Фортрана оказался QBasic. В нем довольно похожим образом можно передать массив в подпрограмму или функцию. Продемонстрируем эту возможность на примере подпрограммы сложения целочисленных квадратных матриц:

Программа 4_01.bas

```
DECLARE SUB ADDMAT(A%(),B%(),C%(),N%)
DEFINT A-Z
CLS
DIM A1(2,2),A2(2,2),A3(2,2)
DIM B1(3,3),B2(3,3),B3(3,3)
FOR J=0 TO 2 : FOR K=0 TO 2
  A1(J,K)=J+K : A2(J,K)=J*K
```

```
NEXT K : NEXT J
CALL ADDMAT(A1(),A2(),A3(),2) 'Так можно обратиться к подпрограмме
FOR J=0 TO 3 : FOR K=0 TO 3
  B1(J,K)=J+K : B2(J,K)=J*K
NEXT K : NEXT J
ADDMAT B1(),B2(),B3(),3 'И так можно обратиться к подпрограмме
END

SUB ADDMAT (A%(),B%(),C%(),N%)
  DEFINT A-Z
  FOR Q=0 TO N : FOR S=0 TO N
    C(Q,S)=A(Q,S)+B(Q,S)
  NEXT S : NEXT Q
END SUB
```

Как в заголовке подпрограммы, так и в операторах обращения к ней массив-параметр сопровождается пустыми круглыми скобками, независимо от числа индексов, приписанных фактически передаваемым массивам. Никаких переобъявлений типа `DIM A(N,N)` в теле подпрограммы не нужно. Более того, попытка вставить такой оператор приведет к фиксации ошибки, сопровождаемой сообщением о повторном объявлении массива. Обратите внимание на то, что в заголовке подпрограммы использованы имена `A%`, `B%`, `C%`, `N%`, а в тексте подпрограммы вместо них выступают имена `A`, `B`, `C`, `N`. Своей эквивалентностью они обязаны оператору `DEFINT A-Z`, объявляющему все переменные внутри подпрограммы целочисленными.

Немного сложнее приходится поступать в программах на Си и Паскале, особенно если речь идет о передаче в качестве параметров двумерных массивов переменного размера.

Массивы-параметры в процедурах и функциях Паскаля

Первое ограничение, с которым сталкиваются программисты в Паскале, звучит следующим образом — если формальным параметром является массив, то его нельзя описать явным образом. Например, ошибочным является заголовок:

```
function Max(a:array [1..100] of integer):integer;
```

Правильным считается предварительное объявление типа в головной программе и его использование в заголовке функции:

```
type
  mas100=array [1..100] of integer;
```

```
.................................
function Max(a:mas100):integer;
```

Однако такая функция обрекает нас на определение максимального элемента только в массиве типа `mas100`. А если в нашей программе понадобится обрабатывать массивы и с другим количеством элементов? Конечно, можно добавить еще один параметр — количество обрабатываемых элементов:

```
function Max(a:mas100; n:integer):integer;
```

Это позволит ограничиться первыми `n` элементами массива, но размерность фактического параметра может быть только `mas100`.

Первый выход из столь неприятного положения заключается в том, чтобы передать массив как адрес параметра без типа:

```
function Max(var a; n:integer):integer;
type
  b=array [1.. MaxInt] of integer;
var
  m,k:integer;
begin
  m:=b(a)[1];
  for k:=2 to n do
    if m<b(a)[k] then m:=b(a)[k];
  Max:=m;
end;
```

Несколько необычная запись `b(a)` означает, что нетипизированный параметр `a` приводится к типу целочисленного массива, каковым он на самом деле и является. Только функция `Max` об этом "не знает". Может показаться немного странным выбор верхней границы индекса в типе "`b`". Однако он позволяет избежать неприятностей, когда фактическое значение параметра `n` достаточно велико (при этом может возникнуть аварийная ситуация, вызванная выходом индекса за границы диапазона). Каким бы большим не было значение `n`, оно не может превысить величину `MaxInt`, ибо массивы в Паскале не могут быть более 64 Кбайт.

Второй подход является модернизацией предыдущего с той лишь разницей, что не потребуются никакие преобразования типов:

```
function Max(var a; n:integer):integer;
var
  c:array[1.. MaxInt] of integer absolute a;
  m,k:integer;
begin
  m:=c[1];
```

```
  for k:=2 to n do
    if m<c[k] then m:=c[k];
  Max:=m;
end;
```

Конструкция с использованием служебного слова `absolute` означает, что массив `c`, под который дополнительное место не выделяется, расположен в памяти, начиная с того же адреса, что и массив `a`.

Третья идея заключается в том, чтобы передать нетипизированный указатель на массив `a`:

```
function Max(pa:pointer; n:integer):integer;
var
  m,k:integer;
  pb:^integer;
begin
  pb:=ptr(seg(pa^),ofs(pa^));
  m:=pb^;
  for k:=2 to n do
  begin
    pb:=ptr(seg(pa^),ofs(pa^)+k*2);
    if m < pb^ then m:=pb^;
  end;
  Max:=m;
end;
```

Для вызова такой функции при определении максимального элемента в массиве `q` необходимо задавать адрес аргумента:

```
max_q:=Max(&q,100);
```

Этот пример демонстрирует работу с адресами и указателями в Паскале. В нем использована функция `ptr`, вычисляющая адрес элемента данных по сегменту (`seg`) и смещению (`ofs`). Адрес сегмента, задающий начало массива, у локального указателя `b` совпадает с адресом массива `a` (`seg(pb^)=seg(pa^)`), а смещение очередного элемента вычисляется в цикле путем прибавления к смещению начального элемента `ofs(pa^)` приращения `2*k`.

Наверное, все эти три подхода блекнут перед новинкой в лице *открытых массивов*, появившихся в последних версиях Паскаля. В качестве открытого массива может выступать любой одномерный массив, индексы в котором отсчитываются от 0. Признаком открытого массива является его описание в укороченном виде — без задания индексных границ:

```
function Max(a:array of integer):integer;
var
  m,k:integer;
```

```
begin
  m:=a[0];
  for k:=1 to High(a) do
    if m < a[k] then m:=a[k];
  Max:=m;
end;
```

Для открытых массивов появилась системная функция `High`, позволяющая узнать максимальный индекс. При использовании открытых массивов вы должны включить соответствующее указание компилятору — `{$P+}`.

Несмотря на всю привлекательность последней версии функции поиска максимума, предыдущие варианты сбрасывать со счетов не стоит. Во-первых, открытые массивы могут быть только одномерными. Во-вторых, их элементы передаются через стек, что замедляет работу программы и может послужить причиной переполнения стека.

Распространим некоторые идеи передачи параметров на двумерные массивы в задаче суммирования квадратных матриц. Первый пример использует нетипизированные переменные:

```
procedure sum_mat(var a,b,c; n:integer);
var
  x:array [0..0] of integer absolute a;
  y:array [0..0] of integer absolute b;
  z:array [0..0] of integer absolute c;
  j,k:integer;
begin
{$R-}
  for j:=0 to n-1 do
  for k:=0 to n-1 do
    z[j*n+k]:=x[j*n+k]+y[j*n+k];
{$R+}
end;
```

В этом примере в качестве элементов суммируемых массивов в теле процедуры выступают одномерные массивы, начало которых совпадает с началом соответствующих двумерных массивов, а индексы пересчитываются по ранее приводившимся формулам. Несколько непривычный диапазон индексов от 0 до 0 компенсируется отключением контроля за выходом индексов из установленного диапазона на время работы циклов по суммированию.

Второй пример использует вариант с указателями:

```
procedure sum_mat(pa,pb,pc:pointer; n:integer);
var
  a,b,c:^integer;
```

```
  j,k:integer;
begin
  for j:=0 to n-1 do
  for k:=0 to n-1 do
  begin
    a:=ptr(seg(pa^),ofs(pa^)+(j*n+k)*2);
    b:=ptr(seg(pb^),ofs(pb^)+(j*n+k)*2);
    c:=ptr(seg(pc^),ofs(pc^)+(j*n+k)*2);
    c^:=a^+b^;
  end;
end;
```

Массивы-параметры в функциях Си

В отличие от Паскаля язык Си располагает более широкими возможностями по части адресной арифметики. Здесь можно прибавлять смещение к указателям, не заботясь о длине адресуемых операндов. Однако двумерные массивы все равно доставляют хлопоты в тех случаях, когда мы хотим построить универсальную функцию для обработки массивов разного размера.

Продемонстрируем технику программирования на тех же задачах, что и в предыдущем разделе.

```
int Max(int *a, int n)
{
  int m,k;
  m=a[0];
  for(k=1; k<n; k++)
    if (m < a[k]) m=a[k];
  return m;
}
```

Предлагаемый вариант очень напоминает работу с открытым массивом с той лишь разницей, что функции типа `High` в Си нет. Поэтому список параметров дополнен еще и количеством элементов в массиве `a`.

Абсолютно эквивалентным является вариант с использованием арифметики указателей:

```
int Max(int *a, int n)
{
  int m,k;
  m=*a;
  for(k=1; k<n; k++)
```

```
    if (m < *(a+k) m=*(a+k);
  return m;
}
```

Следующий пример демонстрирует сложение квадратных матриц с приведением адресов элементов двумерного массива к их одномерным аналогам:

```
void sum_mat(int *a,int *b, int *c, int n)
{
  int j,k,m;
  for(j=0; j < n; j++)
  for(k=0; k < n; k++)
    {
     m=j*n+k;
     *(c+m)=*(a+m) + *(b+m);
    }
}
```

Сортировка больших массивов

Говорят, и это подтверждается многочисленными примерами, что 90% времени работы программы приходится на так называемые узкие места, занимающие в общем объеме программы не более 10% команд. Поиск таких мест и улучшение их временных характеристик — один из основных методов совершенствования программ.

К числу таких узких мест относятся фрагменты программ, занимающиеся упорядочением достаточно объемной информации, поиском данных в больших массивах и т. п. Ниже приводится описание нескольких довольно популярных методов упорядочения числовых массивов и тексты реализующих их процедур. Схемы программ на Си заимствованы из [12], однако в их тексты внесены небольшие изменения. Желающим познакомиться более подробно с другими методами сортировки и возникающими при этом серьезными математическими проблемами мы рекомендуем книгу Д. Кнута "Искусство программирования для ЭВМ", т. 3.

Пузырьковая сортировка (bubble)

Идея этого метода заключается в сравнении двух соседних элементов массива, в результате которого меньшее число (более легкий пузырек) перемещается на одну позицию влево. Обычно такой просмотр организуют с конца массива и после первого прохода самое маленькое число перемещается на первое место. Затем все повторяется от конца массива до второго элемента и т. д.

Известна и другая разновидность обменной сортировки (`bubble1`), при которой также сравниваются два соседа и, если хотя бы одна из смежных пар была переставлена, просмотр начинают с самого начала. Так продолжают до тех пор, пока очередной просмотр не закончится без перестановок.

С целью повышения быстродействия программ на Си наиболее активно используемые переменные `i` и `j` распределяются в регистровой памяти.

Подпрограмма bubble.bas

```
SUB BUBBLE(X(),N)
  FOR I=1 TO N-1
  FOR J=N-1 TO I STEP -1
    IF X(J-1) > X(J) THEN
      TMP=X(J-1): X(J-1) = X(J): X(J)= TMP
    END IF
  NEXT J
  NEXT I
END SUB
```

Подпрограмма bubble1.bas

```
SUB BUBBLE1(X(),N)
M:Q=0
  FOR I=1 TO N-1
    IF X(I-1) > X(I) THEN
      TMP=X(I-1): X(I-1)=X(I): X(I)=TMP: Q=1
    END IF
  NEXT I
  IF Q=1 THEN GOTO M
END SUB
```

Функция bubble.c

```
void bubble(int *x, int n)
{
  register int i,j;
  int tmp;
  for(i=1; i<n; i++)
    for(j=n-1; j>=i; j--)
      if(x[j-1] > x[j])
```

```
        {
          tmp=x[j-1];
          x[j-1] = x[j];
          x[j]= tmp;
        }
}
```

Функция bubble1.c

```
void bubble1(int *x, int n)
{
  register int i,j;
  int tmp,q;
m:q=0;
  for(i=0; i<n-1; i++)
    if(x[i] > x[i+1])
      {
        tmp=x[i];
        x[i]=x[i+1];
        x[i+1]=tmp;
        q=1;
      }
  if(q!=0) goto m;
}
```

Процедура bubble.pas

```
procedure bubble(var x:array of integer;n:integer);
var
  i,j,tmp:integer;
begin
  for i:=1 to n-1 do
    for j:=n-1 downto i do
      if x[j]<x[j-1] then
      begin
        tmp:=x[j-1];
        x[j-1]:=x[j];
        x[j]:=tmp;
      end;
end;
```

Процедура bubble1.pas

```
procedure bubble1(var x:array of integer;n:integer);
label m;
var
  i,tmp,q:integer;
begin
m:q:=0
  for i:=0 to n-2 do
    if x[i] > x[i+1] then
    begin
      tmp:=x[i];
      x[i]:=x[i+1];
      x[i+1]:=tmp;
      q:=1;
    end;
  if q=1 then goto m;
end;
```

Сортировка методом отбора (select)

Идея метода отбора заключается в следующем. Находится элемент с наименьшим значением и меняется местами с первым элементом. Среди оставшихся ищется наименьший, который меняется со вторым, и т. д.

Подпрограмма select.bas

```
SUB SELECT(X(),N)
  FOR I=0 TO N-2
    Q=0: K=I: TMP=X(I)
    FOR J=I+1 TO N-1
      IF X(J)<TMP THEN  K=J: TMP=X(J): Q=1
    NEXT J
    IF Q=1 THEN X(K)=X(I): X(I)=TMP
  NEXT I
END SUB
```

Функция select.c

```
void select(int *x, int n)
{
```

```
  register int i,j,k;
  int q,tmp;
  for(i=0; i<n-1; i++)
    {
      q=0; k=i; tmp=x[i];
      for(j=i+1; j<n; j++)
        {
          if(x[j] < tmp)
            {
              k=j;
              tmp=x[j];
              q=1;
            }
        }
      if(q)
        {x[k]=x[i]; x[i]=tmp;}
    }
}
```

Процедура select.pas

```
procedure select(var x:array of integer;n:integer);
var
  i,j,k,q,tmp:integer;
begin
  for i:=0 to n-2 do
  begin
    q:=0;  k:=i; tmp:=x[i];
    for j:=i+1 to n-1 do
      if x[j]<tmp then
      begin
        k:=j; tmp:=x[j]; q:=1;
      end;
      if q=1 then
      begin
        x[k]:=x[i]; x[i]:=tmp;
      end;
  end;
end;
```

Сортировка методом вставки (insert)

Идея этого метода базируется на последовательном пополнении ранее упорядоченных элементов. На первом шаге сортируются первые два элемента. Затем на свое место среди них вставляется третий элемент. К трем упорядоченным элементам добавляется четвертый, который занимает свое место в новой четверке. И так продолжается до тех пор, пока к `n-1` ранее упорядоченным элементам не присоединится последний. Примерно таким способом игроки упорядочивают свои карты при сдаче их по одной.

Подпрограмма insert.bas

```
SUB INSERT(X%(),N%)
  DIM I,J,TMP
  FOR I=1 TO N-1
    TMP=X(I)
    FOR J=I-1 TO 0 STEP -1
      IF TMP > X(J) THEN EXIT FOR
      X(J+1)=X(J)
    NEXT B
    X(J+1)=TMP
  NEXT I
END SUB
```

Функция insert.c

```
void insert(int *x,int n)
{
  register int i,j;
  int tmp;
  for(i=1; i<n; i++)
    {
      tmp=x[i];
      for(j=i-1; j>=0 && tmp < x[j]; j--)
        x[j+1]=x[j];
      x[j+1]=tmp;
    }
}
```

Процедура insert.pas

```
procedure insert(var x:array of integer;n:integer);
var
  i,j,tmp:integer;
begin
  for i:=1 to n-1 do
  begin
    tmp:=x[i];
    j:=i-1;
    while (j>=0) and (tmp<x[j]) do
    begin
      x[j+1]:=x[j];
      j:=j-1;
    end;
    x[j+1]:=tmp;
  end;
end;
```

Сортировка методом Шелла (shell)

Метод Д. Л. Шелла, опубликованный в 1959 г., предлагает сортировать массив в несколько этапов. На первом этапе в сортировке участвуют достаточно далекие элементы, например, отстоящие друг от друга на восемь позиций. На втором проходе сортируются элементы, расстояние между которыми уменьшено, например до четырех. Затем упорядочиваются каждые вторые элементы и, наконец, на последнем этапе сравниваются смежные элементы. Выбор последовательно уменьшающихся шагов в методе Шелла представляет довольно сложную математическую задачу. На практике чаще всего применяют пятизвенную схему 9→5→3→2→1.

Подпрограмма shell.bas

```
SUB SHELLSORT(X%(),N%)
  DIM I,J,GAP,K,XX,A(5)
  A(0)=9: A(1)=5: A(2)=3: A(3)=2: A(4)=1
  FOR K=0 TO 4
    GAP=A(K)
    FOR I=GAP TO N-1
      XX=X(I)
      FOR J=I-GAP TO 0 STEP -GAP
```

```
        IF XX >= X(J) THEN EXIT FOR
        X(J+GAP)=X(J)
      NEXT J
      X(J+GAP)=XX
    NEXT I
  NEXT K
END SUB
```

Функция shell.c

```
void shell(int *x,int n)
{
  register int i,j,gap,k;
  int xx;
  char a[5]={9,5,3,2,1};
  for(k=0; k<5; k++)
    {
      gap=a[k];
      for(i=gap; i<n; ++i)
        {
          xx=x[i];
          for(j=i-gap; xx < x[j] && j >= 0; j=j-gap)
            x[j+gap]=x[j];
          x[j+gap]=xx;
        }
    }
}
```

Процедура shell.pas

```
procedure shell(var x:array of integer;n:integer);
var
  i,j,k,gap,xx:integer;
const
  a:array [0..4] of byte=(9,5,3,2,1);
begin
  for k:=0 to 4 do
  begin
    gap:=a[k];
```

```
    for i:=gap to n-1 do
    begin
      xx:=x[i];
      j:=i-gap;
      while (j >= 0) and (xx < x[j])  do
      begin
        x[j+gap]:=x[j];
        j:=j-gap;
      end;
      x[j+gap]:=xx;
    end;
  end;
end;
```

Сортировка методом Хоара (quicksort)

Метод Ч. Э. Р. Хоара, опубликованный в 1962 г., издалека напоминает способ ловли льва в пустыне "методом Вейерштрасса". Сначала пустыню делят пополам и выбирают ту половину, в которой находится лев. Затем эту половину снова делят пополам и так продолжают до тех пор, пока область, содержащая льва, не помещается в клетку. В математике подобный метод применяют для нахождения корня функции, принимающей на концах отрезка разные знаки. Это и есть метод Вейерштрасса, более известный под названием "метода деления пополам".

Хоар применил подобный подход к сортировке и предложил следующий алгоритм. Выберем какой-то средний элемент последовательности `xx` (обычно в качестве такового выбирают средний элемент). Если он стоит на своем месте (имеется в виду ситуация, когда последовательность упорядочена), то все элементы, расположенные левее, не больше `xx`, а элементы, находящиеся правее, — не меньше `xx`. Поэтому первый шаг быстрой сортировки заключается в том, чтобы перенести в левую часть массива те элементы правой половины, которые меньше `xx`, а в правую часть — те элементы из левой половины, которые больше `xx`. Обычно при этом поиск ведут в обе стороны от `xx` и как только обнаруживают пару, нарушающую порядок, производят обмен.

Затем аналогичный прием применяют к каждой из полученных половин. В каждой из них выбирается свой средний элемент и относительно него осуществляются необходимые перестановки. Как следует из описания, алгоритм Хоара очень хорошо укладывается в понятие рекурсивной процедуры. Для единообразия в обращении процедура быстрой сортировки представлена в виде двух процедур — внешней `hoar`, к которой обращается пользова-

тель, и внутренней — `quick`. Последняя выполняет рекурсивную обработку подмассива, заданного граничными значениями индексов — `left` и `right`.

Подпрограмма hoare.bas

```
SUB HOARE(X%(),N%)
  QUICK X%(),0,N%-1
END SUB

SUB QUICK(X%(),LEFT%,RIGHT%)
  DIM I,J,XX,YY
  I=LEFT%: J=RIGHT%
  XX=X((LEFT%+RIGHT%)\2)
  DO
    WHILE X%(I) < XX AND I < RIGHT%: I=I+1: WEND
    WHILE XX < X%(J) AND J > LEFT%: J=J-1: WEND
    IF I <= J THEN
      YY=X%(I): X%(I)=X%(J): X%(J)=YY: I=I+1: J=J-1
    END IF
  LOOP WHILE I <= J
  IF LEFT% < J THEN QUICK X%(),LEFT%,J
  IF I < RIGHT% THEN QUICK X%(),I,RIGHT%
END SUB
```

Функция hoare.c

```
void hoare(int *x,int n)
{
  quick(x,0,n-1);
  return;
}
/*-------------------------------------*/
void quick(int *x, int left, int right)
{
  register int i,j;
  int xx,tmp;
  i=left;
  j=right;
  xx=x[(left+right)/2];
```

```
  do
    {
      while (x[i] < xx && i < right)       i++;
      while (xx < x[j] && j > left)        j--;
      if(i<=j)
        {
          tmp=x[i];
          x[i]=x[j];
          x[j]=tmp;
          i++; j--;
        }
    }
  while(i<=j);
  if(left < j)   quick(x,left,j);
  if(i < right)  quick(x,i,right);
}
```

Процедура hoare.pas

```
procedure quick(var x:array of integer;left,right:integer);
var
  i,j,xx,tmp:integer;
begin
  i:=left; j:=right;
  xx:=x[(left+right) div 2];
  repeat
    while (x[i] < xx) and (i < right) do i:=i+1;
    while (xx < x[j]) and (j > left) do j:=j-1;
    if i<=j then
    begin
      tmp:=x[i];
      x[i]:=x[j];
      x[j]:=tmp;
      i:=i+1;
      j:=j-1;
    end;
  until i>j;
  if left < j  then quick(x,left,j);
  if i < right then quick(x,i,right);
end;
```

```
procedure hoare(var x:array of integer;n:integer);
begin
  quick(x,0,n-1);
end;
```

Поиск

Алгоритмические и математические аспекты поиска достаточно сложны и их исследованию посвящены многочисленные работы. Наиболее полно эти вопросы рассматриваются в ранее упоминавшейся трилогии Д. Кнута. Мы же ограничимся самыми простыми алгоритмами и программами, позволяющими понять суть проблемы и познакомиться с некоторыми подходами к повышению эффективности поиска.

В достаточно общих чертах задача поиска формулируется следующим образом. Имеется массив `a`, содержащий `n` однородных объектов (чисел, строк, записей), и нужно установить, содержится ли в нем заданный объект `q`. При положительном ответе следует дополнительно сообщить порядковый номер (индекс) `j` найденного объекта (`a[j] = q`).

Последовательный поиск

Если исходный массив `a` не упорядочен, то единственно разумным способом является последовательный перебор всех элементов массива и сравнение их с заданным значением. В лучшем случае мы можем получить ответ на первом же шаге, если `q = a[1]`. В худшем случае придется перебрать все `n` элементов и только после этого дать положительный или отрицательный ответ. В среднем количество проверок может оказаться порядка `n/2`.

Классический алгоритм последовательного поиска включает следующие шаги:

- S1:Установить начальный индекс равным 1 (`j = 1`).
- S2:Проверить условие `q = a[j]`. Если оно выполняется, то сообщить, что искомое значение находится в массиве а на `j`-ом месте и прервать работу. В противном случае продолжить работу.
- S3:Увеличить индекс `j` на 1.
- S4:Проверить условие `j < n + 1`. Если оно выполняется, то вернуться к шагу S2. В противном случае сообщить, что значение `q` в массиве `a` не содержится.

Большинству программистов кажется, что приведенный алгоритм является оптимальным и ничего сократить в нем нельзя. Однако это — очень распространенное заблуждение. Д. Кнут приводит модификацию алгоритма последовательного поиска, в которой цикл содержит не две логические проверки

(шаги S2 и S4), а всего одну. В нашем случае это довольно существенно, т. к. описанный выше цикл реализуется пятью-шестью машинными командами и исключение из него даже одной команды эквивалентно повышению производительности на 15—20%.

Модификация состоит в том, что к массиву `a` добавляется еще один элемент, в который до начала цикла заносится `q`. Теперь цикл повторяется до тех пор, пока не будет встречен элемент `a[j]`, равный `q`, и необходимость в проверке `j < n + 1` отпадает. Конечно, перед возвратом из процедуры надо удостовериться в том, что найденный индекс `j` не равен `n + 1`. Но такая проверка выполняется за пределами цикла.

Для программистов, имеющих дело с языком ассемблера, известен и более простой прием, не требующий расширения исходного массива. Очевидно, цикл поиска можно организовать как в прямом (`j` меняется от `1` до `n`), так и в обратном (`j` меняется от `n` до `1`) направлении. Обратный цикл на ассемблере реализуется с помощью команды `LOOP`, организующей возврат в начало цикла с одновременным вычитанием 1 из счетчика `CX`. Цикл повторяется до тех пор, пока содержимое регистра `CX` не станет равным 0. Таким образом, дополнительного сравнения (`j < n + 1`) здесь не требуется.

В приводимых ниже программах последовательного поиска возвращаемое соответствующей функцией значение либо равно индексу найденного элемента, либо равно −1, если искомая величина в массиве не обнаружена.

Функция ssearch.bas

```
FUNCTION SSEARCH(Q,A(),N)
  FOR J=0 TO N-1
    IF Q=A(J) THEN SSEARCH=J: EXIT FUNCTION
  NEXT J
  SSEARCH=-1
END FUNCTION
```

Функция ssearch.c

```
int ssearch(int q, int *a, int n)
{
  register int j;
  for(j=0; j<n; j++)
    if(q==a[j]) return j;
  return -1;
}
```

Функция ssearch.pas

```
function ssearch(q:integer;a:array of integer;n:integer):integer;
var
  j:integer;
begin
  for j:=0 to n-1 do
    if q=a[j] then
    begin
      ssearch:=j; exit;
    end;
  ssearch:=-1;
end;
```

Бинарный поиск

Бинарный поиск, суть которого была раскрыта выше на примере поимки льва в пустыне, применяется только для предварительно упорядоченных массивов. Несмотря на абсолютную прозрачность идеи деления пополам, ее программная реализация требует большой аккуратности как в использовании сравнений, так и в выборе очередного интервала поиска.

Исключим тривиальный случай, когда искомое значение `q` выходит за пределы интервала [`a[1]`,`a[n]`]. Обозначим через `left` и `right` индексы элементов массива, определяющие текущий диапазон поиска. В начальный момент `left = 1` и `right = n`. Сравним значение `q` с величиной среднего элемента `a[mid]` в диапазоне поиска `(mid = (left + right)/2)`. Если значение `q` строго меньше `a[mid]`, то заменим правую границу диапазона поиска на `mid - 1`. Если значение `q` строго больше `a[mid]`, то заменим левую границу диапазона поиска на `mid + 1`. Если оба строгие неравенства не выполнены, то имеет место равенство `q = a[mid]`, и в этом случае процедура поиска завершена успешно.

Поиск следует продолжать до тех пор, пока значение индекса `left` не превосходит значения индекса `right`. А нарушиться это условие может только в случае, когда диапазон поиска сведен к минимальному (`right = left + 1`) и имеют место неравенства:

```
a[left] < q < a[right]
```

Это означает, что значение `q` в массиве `a` не содержится.

Функция bsearch.bas

```
FUNCTION BSEARCH(Q,A(),N)
  LEFT=0: RIGHT=N-1
```

```
  WHILE LEFT <= RIGHT
    MIDDLE=(LEFT+RIGHT)\2
    IF Q < A(MIDDLE) THEN RIGHT=MIDDLE-1: GOTO M
    IF Q > A(MIDDLE) THEN LEFT=MIDDLE+1: GOTO M
    BSERACH=MIDDLE : EXIT FUNCTION
M:WEND
  BSEARCH=-1
END FUNCTION
```

Функция bsearch.c

```
int bsearch(int q,int *a,int n)
{
  register int left,right,mid;
  left=0;  right=n-1;
  for(;left <= right;)
    {
      mid=(left+right)/2;
      if(q < a[mid]) right=mid-1;
      else if(q > a[mid]) left=mid+1;
      else return mid;
    }
  return -1;
}
```

Функция bsearch.pas

```
function bsearch(q:integer;a:array of integer;n:integer):integer;
var
  left,right,mid:integer;
begin
  left:=0;  right:=n-1;
  until left <= right do
  begin
    mid:=(left+right) div 2;
    if q < a[mid] then right=mid-1
    else if q > a[mid]  then left=mid+1
    else
    begin
      bsearch:=mid;
```

```
      exit;
    end;
  end;
  bsearch:=-1;
end;
```

Идея бинарного поиска может быть с успехом реализована в игре с отгадыванием задуманного целого числа из заранее установленного диапазона [0,N]. В ответ на число, предложенное угадывающим, его партнер может дать один из трех ответов:

- это число меньше загаданного;
- это число больше загаданного;
- это число равно загаданному.

Оптимальное поведение отгадывающего в точности повторяет схему бинарного поиска. Сначала надо предложить число, расположенное в середине интервала, т. е. N/2. Затем, сузив интервал поиска вдвое, повторить аналогичный маневр. Попадание в цель произойдет не более чем через $\log_2 N$ шагов.

Приведенные ниже тексты программ реализуют оптимальную тактику отгадывания чисел из диапазона [0,100], затрачивая на это не более 7 шагов. На вопросы программы загадавший должен нажимать клавишу Y или y (в случае положительного ответа) или любую другую клавишу, если вопрос программы не соответствует загаданному числу.

Программа 4_02.bas

```
'Программа угадывания задуманного числа в интервале [0,100]
DEFINT A-Z
CLS
LEFT=0: RIGHT=100:
DO
  MIDDLE=(LEFT+RIGHT)\2
  PRINT "Задуманное Вами число больше, чем";MIDDLE;" (Y/N) - ";
  INPUT "",A$
  IF RIGHT-LEFT <= 1 THEN
     IF A$="Y" OR A$="y" THEN
        PRINT "Вы задумали ";RIGHT: END
     ELSE
        PRINT "Вы задумали ";LEFT: END
     END IF
  END IF
```

```
  IF A$="Y" OR A$="y" THEN LEFT=MIDDLE+1 ELSE RIGHT=MIDDLE
LOOP
END
```

Программа 4_02.c

```
/* Программа угадывания задуманного числа в интервале [0,100]*/
#include <stdio.h>
#include <conio.h>
  main()
{
  int left=0, right=100, middle, ok;
  char ch;
  clrscr();
  for (;;)
  {
    middle=(left+right)/2;
    printf("\nЗадуманное Вами число больше, чем % d (Y/N) - ",
              middle);
    ch=getche();
    if(right-left <= 1)
    if (ch=='Y' || ch == 'y') { ok=right;  break; }
    else { ok=left;  break;}
    if(ch=='Y' || ch== 'y') left=middle+1;
    else right=middle;
  }
  printf("\nВы задумали %d",ok);
  getch();
}
```

Программа 4_02.pas

```
{ Программа угадывания задуманного числа в интервале [0,100] }
uses Crt;
var
  ch: char;
  left, right, middle, ok: byte;
begin
  left:=0;
```

```
  right:=100;
  clrscr;
  repeat
    middle := (left + right) div 2;
    write('Задуманное Вами число больше, чем ',middle,' (Y/N) - ');
    ch:=readkey; writeln(ch);
    if (right-left <= 1) then
       if (ch='Y') or (ch = 'y') then
            begin  ok:=right; break;  end
       else begin  ok:=left;  break;  end;
    if(ch='Y') or (ch='y') then left := middle + 1
                           else right := middle;
  until false;
  writeln('Вы задумали ',ok);
  readln;
end.
```

Задачи, советы и ответы

Задание 4.03. Перестановка элементов одномерного массива в обратном порядке

Составить процедуру инвертирования целочисленного массива, которая меняет местами первый элемент с последним, второй — с предпоследним и т. д. Процедура должна обрабатывать массивы любой длины.

> ***Совет 1 (общий)***
>
> В этой задаче особых проблем не возникает, и она включена как подготовительный этап к выполнению последующего задания. Цикл по перестановке элементов следует продолжать до целой части `n/2` (`n` — число элементов в массиве), не обращая внимания на четность или нечетность `n`. Возможно, что при этом средний элемент, который должен оставаться на месте, переставится сам с собой.

Программа 4_03.bas

```
DECLARE SUB INVERT (A%(), N%)
DEFINT A-Z
CLS
N = 20
DIM A(N)
```

```
PRINT "Массив до перестановки :"
FOR I=0 TO N-1: A(I)=I+1: PRINT A(I); : NEXT I
PRINT
INVERT A(), N
PRINT "Массив после перестановки :"
FOR I=0 TO N-1: PRINT A(I); : NEXT I
END

SUB INVERT (A%(), N%)
  DEFINT A-Z
  FOR I = 0 TO (N-1)\2
    TMP = A(I): A(I) = A(N-I-1): A(N-I-1) = TMP
  NEXT I
END SUB
```

Программа 4_03.c

```
#include <stdio.h>
#include <conio.h>
void invert(int *a, int n);
  void main(void)
{
#define N 20
  int i,a[N];
  clrscr();
  printf("Массив до перестановки :\n");
  for(i=0; i < N; i++)
    {a[i]=i+1;  printf("%3d",a[i]); }
  invert(a,N);
  printf("\nМассив после перестановки :\n");
  for(i=0; i < N; i++) printf("%3d",a[i]);
  getch();
}
void invert(int *a,int n)
{
  int j,tmp;
  for(j=0; j < n/2; j++)
    { tmp=a[j]; a[j]=a[n-j-1]; a[n-j-1]=tmp; }
}
```

Программа 4_03.pas

```
program perestanovka;
uses crt;
const
  N=20;
var
  a:array [1..N] of integer;
  j:integer;
procedure invert(var a; n:integer);
var
  j,tmp:integer;
  aa:array [1..1] of integer absolute a;
begin
  {$R-}
  for j:=1 to n div 2 do
  begin
    tmp:=aa[j];
    aa[j]:=aa[n-j+1];
    aa[n-j+1]:=tmp;
  end;
  {$R+}
end;
begin
  clrscr;
  writeln('Массив до перестановки :');
  for j:=1 to N do
  begin a[j]:=j;  write(a[j]:3); end;
  invert(a,N);
  writeln;
  writeln('Массив после перестановки :');
  for j:=1 to N do  write(a[j]:3);
  readln;
end.
```

Задание 4.04. Перестановка головы и хвоста массива

Составить процедуру, перемещающую `k` начальных элементов массива в его конец, сохраняя взаимное расположение между соседними элементами в

головной и хвостовой частях. Основным препятствием для реализации процедуры является запрет на использование вспомогательного массива.

Совет 1 (общий)

Это довольно известная в мире задача из набора программистских "жемчужин" (Бентли Д. Жемчужины творчества программистов / Пер. с англ. М.: Радио и связь, 1990). Она имеет изящное трехшаговое решение. На первом шаге производится обратная перестановка элементов только головной части массива. На втором шаге такой же перестановке подвергаются элементы хвостовой части. И заключительный этап — еще одна перестановка, но уже всех элементов массива.

Конечно, можно принять предложенный алгоритм на веру и убедиться на нескольких примерах, что он работает правильно. Но это не может дать абсолютной уверенности в том, что также произойдет на любых сочетаниях исходных данных. Поэтому можно попытаться доказать правильность алгоритма, проследив за местоположением (индексом) каждого элемента после указанных перестановок.

Если элемент с исходным индексом `i` принадлежал голове массива ($1 \le i \le k$), то инвертирование головы перемещает:

1-й элемент в позицию с индексом `k`,

2-й элемент в позицию с индексом `k-1`,

.....

`k`-й элемент в позицию с индексом 1.

Сумма индексов элементов, меняющихся своими местами, при этом равна `k+1`. Поэтому `i`-й элемент из головной части после инвертирования головы массива окажется на месте с индексом `k-i+1`. При последующем инвертировании всего массива сумма индексов взаимных пар меняющихся элементов равна `n+1`. Следовательно, элемент с первоначальным индексом `i` из головной части окажется на месте элемента с индексом `(n+1)-(k-i+1)=n-k+i`. Таким образом:

1-й элемент окончательно окажется в позиции `n-k+1`,

2-й элемент окончательно окажется в позиции `n-k+2`,

.....

`k`-й элемент окончательно окажется в позиции `n-k+k`.

Это означает, что прежняя голова массива переместится в его хвост, сохранив взаимное расположение элементов.

Теперь рассмотрим поведение `i`-го элемента из хвостовой части ($k + 1 \le i \le n$). После инвертирования хвоста:

элемент с индексом `k+1` окажется в позиции `n`,

элемент с индексом `k+2` окажется в позиции `n-1`,

.....

элемент с индексом `n` окажется в позиции `k+1`.

Сумма индексов пар, меняющихся при этом местами, равна `n+k+1`. Поэтому `i`-й элемент из хвостовой части после инвертирования хвоста перемещается в по-

зицию n+k+1-i. При последующем инвертировании полного массива, когда сумма индексов соответствующих пар равна n+1, произойдет следующее:

элемент с индексом k+1, предварительно перемещенный в позицию n, займет место элемента с номером (n+1)-n=1;

элемент с индексом k+2, предварительно перемещенный в позицию n-1, займет место элемента с номером (n+1)-(n-1)=2;

.....

элемент с индексом n, предварительно перемещенный в позицию k+1, займет место элемента с номером (n+1)-(k+1)=n-k.

Таким образом, хвостовые элементы, сохраняя взаимное расположение, перейдут в голову нового массива и будут находиться вплотную перед его прежней головой.

Совет 2 (общий)

Процедуру invert, приведенную в предыдущем задании, целесообразно модифицировать, задавая в ней дополнительные параметры. Например — индекс j, начиная с которого производится инвертирование, и количество len элементов, участвующих в перестановке.

Программа 4_04.bas

```
DECLARE SUB INVERT1 (A%(), J%, N%)
DEFINT A-Z
CLS
N=20: K=15
DIM A(N)
PRINT "Массив до перестановки :"
FOR I=0 TO N-1: A(I)=I+1: PRINT A(I); : NEXT I
PRINT
INVERT1 A(),0,K
PRINT "После перестановки в головной части :"
FOR I=0 TO N-1: PRINT A(I); : NEXT I: PRINT
INVERT1 A(),K,N-K
PRINT "После перестановки в хвостовой части :"
FOR I=0 TO N-1: PRINT A(I); : NEXT I: PRINT
INVERT1 A(),0,N
PRINT "После полной перестановки :"
FOR I=0 TO N-1: PRINT A(I); : NEXT I
END
```

```
SUB INVERT1 (A%(), J%, N%)
  DEFINT A-Z
  FOR I = J TO J+(N-1)\2
    TMP=A(I): A(I)=A(2*J+N-1-I): A(2*J+N-1-I)=TMP
  NEXT I
END SUB
```

Программа 4_04.c

```
#include <stdio.h>
#include <conio.h>
void invert1(int *a, int j,int k);
  void main(void)
{
#define N 20
#define K 15
  int i,a[N];
  clrscr();
  printf("Массив до перестановки :\n");
  for(i=0; i < N; i++)
    { a[i] = i + 1;  printf("%3d",a[i]); }
  invert1(a,0,K);
  printf("\nПосле перестановки в головной части :\n");
  for(i=0; i < N; i++) printf("%3d",a[i]);
  invert1(a,K,N-K);
  printf("\nПосле перестановки в хвостовой части :\n");
  for(i=0; i < N; i++) printf("%3d",a[i]);
  invert1(a,0,K);
  printf("\nПосле полной перестановки :\n");
  for(i=0; i < N; i++) printf("%3d",a[i]);
  getch();
}
void invert1(int *a, int j, int k)
{
  int m,tmp;
  for(m=j; m < j+k/2; m++)
    { tmp=a[m]; a[m] = a[2*j+k-m-1];  a[2*j+k-m-1]=tmp; }
```

```
}
```

Программа 4_04.pas

```
program pearl;
uses crt;
const
  N=20; K=15;
var
  a:array [1..N] of integer;
  i:integer;
procedure invert1(var a; j,k:integer);
var
  m,tmp:integer;
  aa:array [1..1] of integer absolute a;
  begin   {$R-}
    for m:=j to j + k div 2 do
    begin
      tmp:=aa[m]; aa[m]:=aa[2*j+k-m-1];  aa[2*j+k-m-1]:=tmp;
    end;  {$R+}
  end;
begin
  clrscr;
  writeln('Массив до перестановки :');
  for i:=1 to N do
  begin a[i]:=i;  write(a[i]:3); end;
  writeln;
  invert1(a,1,K);
  writeln('После перестановки в головной части :');
  for i:=1 to N do write(a[i]:3); writeln;
  invert1(a,K+1,N-K);
  writeln('После перестановки в хвостовой части :');
  for i:=1 to N do write(a[i]:3); writeln;
  invert1(a,1,N);
  writeln('После полной перестановки :');
  for i:=1 to N do write(a[i]:3);
  readln;
end.
```

Задание 4.05. Вывод массивов

Составить процедуру (функцию) вывода небольших целочисленных матриц размерности m×n с управлением по ширине колонок (w) и по местоположению на экране (row — строка, col — столбец, определяющие левый верхний угол матрицы). Предполагается, что матрица целиком помещается на экране и что ширина ее колонок не превосходит девяти позиций.

Совет 1 (общий)

В каждом алгоритмическом языке предусмотрена процедура типа window, позволяющая определить прямоугольную область вывода на экране. Однако пользоваться ей в данном случае нецелесообразно, т. к. придется запоминать параметры окна, существовавшие до обращения к процедуре вывода, затем устанавливать новые, а перед выходом из процедуры восстанавливать запомненные. В Си и Паскале, в принципе, такая возможность имеется (см. функцию gettextinfo), а в QBasic такие действия перекладываются на программиста. Все это сильно усложнит организацию процедуры. Будем считать, что в нашем распоряжении находится полный экран и никаких дополнительных окон вызывающая программа не создавала. Поэтому можно воспользоваться прямыми средствами управления позицией курсора на экране — LOCATE (QBasic) или gotoxy (Си, Паскаль).

Совет 2 (QBasic)

Для вывода числа, прижатого к правой границе поля фиксированной ширины, целесообразно использовать оператор PRINT USING. Так как ширина поля в нашем случае — величина переменная, то шаблон вывода придется сформировать в виде значения строки, содержащей w символов "#".

Совет 3 (Си)

Так же, как и в Бейсике, здесь придется сформировать символьную строку вида "%wd". Чтобы вписать в нее символьное представление ширины поля w, можно воспользоваться суммой вида w+48, где добавка 48 соответствует коду символа "0".

Совет 4 (Паскаль)

Оператор write позволяет задавать ширину поля вывода не только в виде конкретного числа, но и как значение арифметического выражения (в данном случае простейшего — w).

Программа 4_05.bas

```
DECLARE SUB PRINTA (ROW%, COL%, W%, C%(), N%, M%)
DEFINT A-Z
CLS
DIM A(2, 3), B(3, 3)
```

```
FOR J = 0 TO 2: FOR K = 0 TO 3
  A(J, K) = J + K * K
NEXT K: NEXT J
PRINTA 5, 5, 3, A(), 3, 4
FOR J = 0 TO 3: FOR K = 0 TO 3
  B(J, K) = J * 10 + K * 25
NEXT K: NEXT J
PRINTA 5, 40, 5, B(), 4, 4
END

SUB PRINTA (ROW%, COL%, W%, C%(), N%, M%)
  DEFINT A-Z
  F$ = LEFT$("##########", W)
  FOR J = 0 TO N - 1: FOR K = 0 TO M - 1
    LOCATE ROW + J, COL + K*W
    PRINT USING F$; C(J, K)
  NEXT K: NEXT J
END SUB
```

Программа 4_05.c

```
#include <stdio.h>
#include <conio.h>
void printa(int row,int col,int w,int *c,int n,int m);
  void main(void)
{
  int a[3][4]={{1,2,3,4},{10,20,30,40},{100,200,300,400}};
  int b[4][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
  clrscr();
  printa(5,5,4,(int *)a,3,4);
  printa(5,40,5,(int *)b,4,4);
  getch();
}
void printa(int row,int col,int w,int *c,int n,int m)
{
  int j,k;
  char f[4]="%0d";
  f[1] += w;
  for(j=0; j<n; j++)
```

```
    for(k=0; k<m; k++)
      {
        gotoxy(col+k*w,row+j);
        printf(f,c[k+j*m]);
      }
}
```

Программа 4_05.pas

```
program mat_print;
uses crt;
const
  a:array [1..3,1..4] of integer = ((1,2,3,4), (10,20,30,40),
(100,200,300,400));
  b:array [1..4,1..4] of integer = ((1,2,3,4), (5,6,7,8), (9,10,11,12),
(13,14,15,16));
procedure printa(row,col,w:integer;var c;n,m:integer);
var
  j,k:integer;
  d:array [0..MaxInt-1] of integer absolute c;
begin
  for j:=0 to n-1 do
    for k:=0 to m-1 do
      begin
        gotoxy(col+k*w,row+j);
        write(d[k+m*j]:w);
      end;
end;
begin
  clrscr;
  printa(5,5,4,a,3,4);
  printa(5,40,5,b,4,4);
  readln;
end.
```

Задание 4.06. Ход конем

Существует довольно много комбинаторных задач, связанных с шахматным полем. Это и задача о расстановке восьми ферзей, которые бы не угрожали друг другу, т. е. чтобы ни одна из пар не находилась на общей вертикали,

горизонтали и диагонали. Это и различные путешествия коня по шахматному полю. Среди них — построение траектории ходов, при которых конь, начиная из заданной позиции, обходит все клетки поля, не заходя повторно в уже пройденные. Или — заданы начальная и конечная позиции, и нужно построить одну или все траектории между указанными клетками, содержащие минимальное число ходов. Самая простая среди "лошадиных" задач заключается в определении минимального количества ходов, за которое конь может добраться из начальной позиции в конечную. Именно ей и посвящено очередное задание.

Говорят, что конь ходит буквой "Г". Это не совсем точно, т. к. допустимые ходы этой фигуры образуются перемещениями на две клетки по горизонтали (вертикали) с последующим поворотом на 90 градусов по или против часовой стрелки и сдвигом по вертикали (горизонтали) еще на одну клетку. Если сопоставить с шахматной доской матрицу 8×8, то из позиции `a[i,j]` возможны максимум 8 ходов в клетки со следующими индексами:

1. `a[i-2,j-1]`	5. `a[i+1,j-2]`
2. `a[i-2,j+1]`	6. `a[i+1,j+2]`
3. `a[i-1,j-2]`	7. `a[i+2,j-1]`
4. `a[i-1,j+2]`	8. `a[i+2,j+1]`

По мере приближения к границам поля какие-то из этих перемещений становятся недопустимыми из-за выхода некоторых индексов за пределы установленного диапазона. Минимальным числом ходов, равным 2, обладают угловые клетки шахматной доски.

Для определения минимального количества ходов, соединяющих клетки `a[i1,j1]` и `a[i2,j2]`, предлагается следующий алгоритм. Сначала все элементы массива расписываются каким-то кодом, например числом −1, означающим, что все клетки шахматного поля свободны. Затем в начальную позицию `a[i1,j1]` заносится 0 и делаются попытки определить все клетки, достижимые за один ход. В каждую из них заносим 1. Для каждой клетки, достигаемой после первого хода, делается попытка совершить следующий ход, и все вновь достигаемые позиции метятся кодом 2. Естественно, что среди них исключаются ходы, возвращающие нас в начальную позицию, т. е. ходить надо только по свободным клеткам. Из каждой клетки, достигаемой за два хода, делается попытка совершить следующий ход в еще не занятые позиции и пометить все допустимые элементы массива числом 3. Числовую метку, которую мы заносим на очередном шаге охвата клеток шахматного поля, можно назвать уровнем досягаемости.

Расстановка уровней досягаемости продолжается до тех пор, пока мы не попадем в конечную позицию `a[i2,j2]`. А еще лучше — до тех пор, пока не будет заполнена вся матрица, и тогда мы получим ответ, за сколько ходов можно переместиться из позиции `a[i1,j1]` в любую другую клетку шахматного поля.

Наряду с матрицей `a` можно ввести еще один массив `b` и заполнять его аналогичным образом, начиная из конечной позиции `b[i2,j2]`. Если `n` — минимальное число ходов, переводящих коня из клетки `a[i1,j1]` в клетку `a[i2,j2]`, то обратное путешествие по минимальной траектории займет тоже `n` шагов. Кроме того, можно заметить, что в каждой клетке любой минимальной траектории имеет место равенство

```
a[i,j] + b[i,j] = n.
```

Оно означает, что число промежуточных ходов из начальной позиции в клетку `a[i,j]` и число встречных ходов по этой же траектории из конечной точки в сумме составляют длину минимальной траектории. Этот факт позволяет отсечь все недопустимые позиции и, тем самым, выделить клетки, принадлежащие только минимальным траекториям.

В качестве программы, доведенной до конца, мы ограничимся формированием всех уровней досягаемости из заданной позиции и по результатам ее работы узнаем, что любая клетка шахматного поля может быть достигнута не более чем за шесть ходов.

Совет 1 (общий)

Целесообразно выделить в отдельную процедуру поиск и заполнение клеток, достигаемых за один ход из каждой клетки `k`-го уровня. Назовем эту процедуру `newlevel` и условимся, что, помимо маркировки клеток `(k+1)`-го уровня, она сообщает, удалось ли сделать хотя бы один ход.

Программа 4_06.bas

```
  DIM A(8,8)
  FOR I=0 TO 7 : FOR J=0 TO 7
    A(I,J)=-1
  NEXT J: NEXT I
  CLS : INPUT "Задайте начальную позицию :",I,J
  A(I,J)=0 : K=0
M:NEWLEVEL
  IF XOD = 1 THEN GOTO M
  FOR I=0 TO 7 : FOR J=0 TO 7
    PRINT A(I,J)
  NEXT J: PRINT : NEXT I
  END

SUB NEWLEVEL
  XOD=0
```

```
  FOR I=0 TO 7 : FOR J=0 TO 7
    IF A(I,J)=K THEN
       TRY(I-2,J-1): TRY(I-2,J+1)
       TRY(I-1,J-2): TRY(I-1,J+2)
       TRY(I+1,J-2): TRY(I+1,J+2)
       TRY(I+2,J-1): TRY(I+2,J+1)
    END IF
  NEXT J: NEXT I
  K=K+1
END SUB

SUB TRY(P,Q)
  IF P>=0 AND P<8 AND Q>=0 AND Q<8 AND A(P,Q)<0 THEN
    A(P,Q)=K+1 : XOD=1
  END IF
END SUB
```

Программа 4_06.c

```
#include <stdio.h>
#include <conio.h>
char newlevel(void);
void try(int p,int q);
char a[8][8],i,j,k=0,xod;
  void main(void)
{
  clrscr();
  for(i=0; i<8; i++)
    for(j=0; j<8; j++)
      a[i][j]=-1;
  puts("Задайте начальную позицию :");
  scanf("%d %d",&i,&j);
  a[i][j]=0;
m:newlevel();
  if(xod==1) goto m;
  for(i=0; i<8; i++)
    {
      for(j=0; j<8; j++)  printf("%3d",a[i][j])=—1;
      printf("\n");
```

```
    }
  getch();
}
char newlevel(void)
{
  char di[8]={-2,-2,-1,-1, 1,1, 2,2};
  char dj[8]={-1, 1,-2, 2,-2,2,-1,1};
  char f;
  xod=0;
  for(i=0; i<8; i++)
  for(j=0; j<8; j++)
    if(a[i][j]==k)
      for(f=0; f<8; f++) try(i+di[f],j+dj[f]);
  k++;
  return xod;
}
void try(int p,int q)
{
  if(p>=0 && p<8 && q>=0 &&  q<8 && a[p][q]<0)
    { a[p][q]=k+1; xod=1;}
}
```

Программа 4_06.pas

```
program horst;
{=================================================
Построение полей досягаемости ходом коня из заданной
позиции. Индексы позиций задаются от 0 до 7. Поле, из
которого конь начинает, помечается уровнем 0. Поля,
достигаемые после первого хода, помечаются уровнем 1,
после второго хода - уровнем 2 и т. д.
=================================================}
uses Crt;
label m;
var
  i,j,k,xod:byte;
  a:array [0..7,0..7] of shortint;

procedure try(p,q:integer);
```

```
{=================================================
Попытка совершить ход уровня k+1 в позицию (p,q).
При возможности хода в a[p,q] заносится номер
уровня, а в переменную xod - значение 1. В противном
случае переменная xod остается равной 0.
=================================================}
begin
  if (p>=0) and (p<8) and (q>=0) and (q<8) and (a[p,q]<0) then
     begin  a[p,q]:=k+1; xod:=1; end;
end;

procedure newlevel;
{=============================
Поиск всех ходов уровня k+1 из
клеток, достигнутых за k ходов
=============================}
const
  {смещения по индексам i,j для 8 возможных ходов коня}
  di:array [0..7] of integer = (-2,-2,-1,-1, 1,1, 2,2);
  dj:array [0..7] of integer = (-1, 1,-2, 2,-2,2,-1,1);
var
  f:byte;
begin
  xod:=0;  {Гашение признака возможности совершить ход}
  {Поиск полей, достигаемых на k-том уровне}
  for i:=0 to 7 do
    for j:=0 to 7 do
      if a[i,j]=k then
  {Если поле найдено, из него делаются
   попытки сходить по 8 возможным направлениям}
        for f:=0 to 7 do try(i+di[f],j+dj[f]);
  k:=k+1;  {Повышение уровня хода}
end;

begin
  k:=0;   {Уровень хода}
  clrscr;
  for i:=0 to 7 do
    for j:=0 to 7 do
```

```
      a[i,j]:=-1; {Начальная роспись матрицы позиций}
  write('Задайте начальную позицию : ');
  readln(i,j);
  a[i,j]:=0; {Отметка начальной позиции}
m: newlevel; {На поиск ходов следующего уровня}
  if xod=1 then goto m; {Если поиск был завершен успешно}
  {Цикл вывода заполненной матрицы с уровнями ходов}
  for i:=0 to 7 do
    begin
      for j:=0 to 7 do
        write(a[i,j]:3);
      writeln;
    end;
  readln;
end.
```

Задание 4.07. Сравнение методов сортировки

Составить программу тестирования методов сортировки, приведенных выше. Предполагается, что тестовая программа будет генерировать случайным образом целочисленный массив достаточно большого размера, например содержащий 15 000 элементов. Для чистоты эксперимента каждому методу должен задаваться один и тот же исходный массив, т. е. не надо прибегать к процедуре `randomize`. В начале и конце работы программы сортировки необходимо зафиксировать показания системных часов, по разности которых можно судить о быстродействии того или иного алгоритма на данном компьютере.

Совет 1 (общий)

Программу тестирования целесообразно написать в двух вариантах — в укороченном (`MAX = 20`) и в полном (`MAX = 15 000`, для интерпретатора QBasic достаточно `MAX = 5000`). Их можно объединить, закомментировав ту или иную группу операторов. В укороченном варианте следует предусмотреть вывод элементов массива до и после упорядочения с целью проверки работоспособности программы. В полном варианте вывод массива следует закомментировать, чтобы это время не включалось в хронометраж алгоритма.

Совет 2 (общий)

Можно собрать в рамках одной программы все алгоритмы сортировки, но оставить незакомментированным вызов только одного из них. А затем по очереди подключать тот или иной алгоритм.

Программа 4_07.bas

```
DECLARE SUB BUBBLE(X%(),N%)
DECLARE SUB INSERT(X%(),N%)
DECLARE SUB SELECT1(X%(),N%)
DECLARE SUB SHELLSORT(X%(),N%)
DECLARE SUB HOARE(X%(),N%)
DECLARE SUB QUICK(X%(),LEFT%,RIGHT%)

DEFINT A-Z
CLS
CONST N=5000
DIM A(N)
FOR J=0 TO N
  A(J)=INT(N*RND)
'PRINT USING "#### "; A(J);
NEXT J
PRINT
T1#=TIMER
BUBBLE A(),N
'INSERT A(),N
'SELECT1 A(),N
'SHELLSORT A(),N
'HOARE A(),N
T2#=TIMER
PRINT T2#-T1#; "сек"
'FOR J=0 TO N: PRINT USING "#### ";A(J); : NEXT J
END

REM Тексты подпрограмм сортировки
..................
```

Программа 4_07.c

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#define MAX 15000
```

```
void bubble(int *x, int n);
void select(int *x, int n);
void insert(int *x, int n);
void shell(int *x, int n);
void hoare(int *x, int n);
void quick(int *x, int left, int right);
  main()
{
  int num[MAX];
  int i;
  struct time w;
  clrscr();
  //cout << "До сортировки" << endl;
  for(i=0; i < MAX; i++)
//  {
      num[i]=random(MAX);
//    cout << num[i] << " ";}
//    cout << endl;
//    getch();
//    cout << "После сортировки" << endl;
  gettime(&w);
  cout << (unsigned int)w.ti_sec <<".";
  cout << (unsigned int)w.ti_hund << endl;
  bubble(num,MAX);
//select(num,MAX);
//insert(num,MAX);
//shell(num,MAX);
//hoare(num,MAX);
  gettime(&w);
  cout << (unsigned int)w.ti_sec <<".";
  cout << (unsigned int)w.ti_hund << endl;
//cout << "Кончили";
  getch();
//for(i=0; i<MAX; i++)
//cout << num[i] << " ";
//getch();
  return 0;
}
/*===== тестируемые процедуры ==========*/
```

Программа 4_07.pas

```
program all_sort;
uses crt,WinDos;
const
  MAX=15000;
type
  xarr=array [0..MAX-1] of integer;
var
  x:xarr;
  i:integer;
  hour,minute,second,sec100:word;
{===== тестируемые процедуры ======}
begin
  clrscr;
  for i:=0 to MAX-1 do
  begin
    x[i]:=random(MAX);
    {write(x[i],' ');}
  end;
  writeln;
  write('До сортировки    - ');
  gettime(hour,minute,second,sec100);
  write(minute:2,' мин ',second:2,'.',sec100:2,' сек');
  writeln;
{Вызов метода}
{ bubble(x,MAX);}
{ select(x,MAX);}
{ insert(x,MAX);}
  shell(x,MAX);
{ hoare(x,MAX);}
  gettime(hour,minute,second,sec100);
  write('После сортировки - ');
  write(minute:2,' мин ',second:2,'.',sec100:2,' сек');
  writeln;
{ for i:=0 to MAX-1 do  write(x[i],' ');}
  readln;
end.
```

Задание 4.08. Счастливый билет

Билет с шестизначным цифровым номером считается "счастливым", если сумма трех старших цифр совпадает с суммой трех младших цифр. В предположении, что в билетной кассе находится миллион билетов с номерами от 000000 до 999999, надо определить количество потенциально осчастливленных пассажиров.

Совет 1 (общий)

Алгоритм "в лоб" заключается в том, чтобы организовать шесть вложенных циклов, в каждом из которых перебирается очередная цифра номера. В самом внутреннем цикле организуется проверка суммы старших и младших цифр номера и подсчет счастливых билетов. Однако эту проверку придется повторять миллион раз, и даже достаточно мощные процессоры затратят на такой алгоритм заметное время. Решение задачи можно найти примерно в 1000 раз быстрее, если вместо лобового перебора подсчитать, сколько раз сумма трех цифр равна 0, 1, 2, ... , 27. Очевидно, что нулевую сумму дает единственная комбинация цифр 000, представляющая единственный счастливый билет с номером 000000. Сумму, равную 1, дают три комбинации — 001, 010 и 100. Соответственно, счастливых билетов с такой суммой уже 9 (каждая из перечисленных выше комбинаций в старших цифрах с тремя аналогичными сочетаниями в младших разрядах). Таким образом, если обозначить через `S[0], S[1], ... , S[27]` количество комбинаций цифр, сумма которых равна индексу элемента в массиве `S`, то общее количество счастливых билетов `N` будет равно сумме:

```
N = S[0] * S[0] + S[1] * S[1] + ... + S[27] * S[27].
```

Программа 4_08.bas

```
DIM S(28)
FOR A1=0 TO 9: FOR A2=0 TO 9: FOR A3=0 TO 9
  S(A1+A2+A3)=S(A1+A2+A3)+1
NEXT A3: NEXT A2: NEXT A1
FOR K=0 TO 27: N=N+S(K)*S(K) : NEXT K
PRINT "Количество счастливых билетов = ";N
END
```

Программа 4_08.c

```
#include <stdio.h>
#include <conio.h>
  main()
{
```

```
  int s[28],a1,a2,a3,k;
  long N;
  clrscr();
  for(k=0; k<28; k++)s[k]=0;
  for(a1=0; a1<10; a1++)
    for(a2=0; a2<10; a2++)
      for(a3=0; a3<10; a3++)
        s[a1+a2+a3]++;
  for(k=0,N=0; k<28; k++)
    N += s[k]*s[k];
  printf("\nКоличество счастливых билетов = %ld",N);
  getch();
}
```

Программа 4_08.pas

```
program lucky_ticket;
var
  a1,a2,a3,k:integer;
  N:longint;
  s:array [0..27] of integer;
begin
  for a1:=0 to 9 do
    for a2:=0 to 9 do
      for a3:=0 to 9 do
        inc(s[a1+a2+a3]);
  for k:=0 to 27
    N:=N+s[k]*s[k];
  writeln('Количество счастливых билетов = ',N);
  readln;
end.
```

Задание 4.09. Количество разных элементов в целочисленном массиве

Составить подпрограмму подсчета количества разных чисел в массиве, элементами которого являются неотрицательные двухбайтовые значения. Эта задача интересна не своей практической ценностью, а возможностью продемонстрировать несколько разных подходов к ее решению.

Совет 1 (общий)

Первый алгоритм основан на предварительном упорядочении элементов массива по убыванию (или возрастанию) с последующим сравнением соседних элементов. В качестве подпрограммы сортировки может быть использована любая из приводившихся выше процедур. Тестовые примеры подобраны таким образом, что тело процедуры `sort` может быть пустым, чтобы не загромождать программу.

Совет 2 (общий)

Второй алгоритм состоит из трех шагов. На первом из них выясняется, содержится ли в исходном массиве хотя бы один элемент с нулевым значением. Если таковой не обнаружен, то признак `k0 = 0`, в противном случае `k0 = 1`. На втором шаге организуются два вложенных цикла, в которых каждый ненулевой `i`-й элемент сравнивается со всеми последующими. Если при этом будет обнаружен двойник, то `i`-му элементу присваивается нулевое значение. На третьем шаге подсчитывается количество оставшихся ненулевых значений, которое корректируется на величину `k0`.

Совет 3 (общий)

Третий алгоритм базируется на использовании массива индикаторов. Так как значения элементов исходного массива принадлежат интервалу [0,32767], то заводится 32 768 битовых индикаторов (4096 байт). В начале все индикаторы сбрасываются в нуль, а затем выполняется цикл одноразового просмотра элементов исходного массива. Для значения каждого элемента `a[i]` проверяется соответствующий бит в массиве индикаторов и, если он равен 0, в него заносится 1 и к счетчику разных чисел добавляется единица. Если проверяемый бит индикаторов отличен от нуля, то соответствующее значение уже встречалось.

Совет 4 (Си)

Массив индикаторов (`b`) можно явным образом запрашивать при входе в подпрограмму и освобождать при выходе, хотя локальные данные в процедурах и функциях все равно выделяются и освобождаются автоматически. Но в библиотеке Си есть полезная функция `calloc`, которая одновременно выделяет память и чистит ее. Переменные `byte` и `bit` использованы для определения байта и бита в массиве индикаторов, соответствующих анализируемому значению `a[i]`.

Программа 4_09a.bas

```
DECLARE SUB SORT (A() AS INTEGER, N%)
DECLARE FUNCTION DIFFERENCE% (A() AS INTEGER, N%)
DEFINT A-Z
DIM A(5)
DATA 0,0,0,0,0
DATA 1,1,1,1,1
DATA 0,1,1,1,1
```

```
DATA 0,0,1,1,2
DATA 0,1,2,3,4
DATA 1,2,3,4,5
CLS
FOR k=1 TO 5
 FOR I=0 TO 4: READ A(I): NEXT I
 PRINT "Количество разных чисел в массиве ";k;" = ";
 PRINT DIFFERENCE(A(), 5)
NEXT k
END

FUNCTION DIFFERENCE (A() AS INTEGER, N%)
  SORT A(),N%
  M=1
  FOR I=0 TO N%-2
    IF A(I)<>A(I+1) THEN M=M+1
  NEXT I
  DIFFERENCE=M
END FUNCTION

SUB SORT(A() AS INTEGER,N%)
  REM тело любой процедуры сортировки
END SUB
```

Программа 4_09b.bas

```
DECLARE SUB SORT (A() AS INTEGER, N%)
DECLARE FUNCTION DIFFERENCE% (A() AS INTEGER, N%)
DEFINT A-Z
DIM A0(5),A1(5),A2(5),A3(5),A4(5),A5(5)
DATA 0,0,0,0,0
FOR I=0 TO 4: READ A0(I): NEXT I
DATA 1,1,1,1,1
FOR I=0 TO 4: READ A1(I): NEXT I
DATA 0,1,1,1,1
FOR I=0 TO 4: READ A2(I): NEXT I
DATA 0,0,1,1,2
FOR I=0 TO 4: READ A3(I): NEXT I
DATA 0,1,2,3,4
```

```
FOR I=0 TO 4: READ A4(I): NEXT I
DATA 1,2,3,4,5
FOR I=0 TO 4: READ A5(I): NEXT I
PRINT "Количество разных чисел в массиве A0 = ";
PRINT DIFFERENCE(A0(),5)
PRINT "Количество разных чисел в массиве A1 = ";
PRINT DIFFERENCE(A1(),5)
PRINT "Количество разных чисел в массиве A2 = ";
PRINT DIFFERENCE(A2(),5)
PRINT "Количество разных чисел в массиве A3 = ";
PRINT DIFFERENCE(A3(),5)
PRINT "Количество разных чисел в массиве A4 = ";
PRINT DIFFERENCE(A4(),5)
PRINT "Количество разных чисел в массиве A5 = ";
PRINT DIFFERENCE(A5(), 5)
END

FUNCTION DIFFERENCE (A() AS INTEGER,N%)
DEFINT A-Z
  FOR I=0 TO N%-1
    IF A(I)=0 THEN K0=1: EXIT FOR
  NEXT I
  FOR I=0 TO N%-1
    IF A(I)<>0 THEN
       FOR J=I+1 TO N%-1
         IF A(I)=A(J) THEN A(I)=0: EXIT FOR
       NEXT J
    END IF
  NEXT I
  FOR I=0 TO N%-1
    IF A(I)<>0 THEN M=M+1
  NEXT I
  DIFFERENCE=M+K0
END FUNCTION
```

Программа 4_09a.c

```
#include <stdio.h>
#include <conio.h>
```

```
void sort(int *a,int n);
int difference(int *a,int n);
  main()
{
  int a0[5]={0,0,0,0,0};
/*
  int a1[5]={1,1,1,1,1};
  int a2[5]={0,1,1,1,1};
  int a3[5]={0,0,1,1,2};
  int a4[5]={0,1,2,3,4};
  int a5[5]={1,2,3,4,5};
*/
  printf("\nКоличество разных чисел в этом массиве равно ");
  printf("%d",difference(a0,5));
  getch();
}
void sort(int *a,int n)
{
  /* тело любой процедуры сортировки */
  return;
}
int difference(int *a, int n)
{
  int i,m;
  sort(a,n);
  for(i=0,m=1; i<n-1; i++)
    if(a[i] != a[i+1]) m++;
  return m;
}
```

Программа 4_09b.c

```
#include <stdio.h>
#include <conio.h>
int difference(int *a,int n);
  main()
{
  int a0[5]={0,0,0,0,0};
  int a1[5]={1,1,1,1,1};
```

```
  int a2[5]={0,1,1,1,1};
  int a3[5]={0,0,1,1,2};
  int a4[5]={0,1,2,3,4};
  int a5[5]={1,2,3,4,5};

  printf("\nКоличество разных чисел в этом массиве равно ");
  printf("%d",difference(a0,5));
  getch();
}
int difference(int *a,int n)
{
  int i,j,k0,m;
  for(i=k0=0; i<n; i++)
    if(a[i] == 0) { k0=1;  break; }
  for(i=0; i<n-1; i++)
    { if(a[i]==0) continue;
      for(j=i+1; j<n; j++)
        if(a[i]==a[j]) { a[i]=0; break; }
    }
   for(i=m=0; i<n; i++)
     if(a[i]!=0)m++;
   return m+k0;
}
```

Программа 4_09c.c

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int difference(int *a,int n);
  main()
{
  int a0[5]={0,0,0,0,0};
  int a1[5]={1,1,1,1,1};
  int a2[5]={0,1,1,1,1};
  int a3[5]={0,0,1,1,2};
  int a4[5]={0,1,2,3,4};
  int a5[5]={1,2,3,4,5};

  printf("\nКоличество разных чисел в этом массиве равно ");
```

```
  printf("%d",difference(a5,5));
  getch();
}
int difference(int *a, int n)
{
  char *b;
  char mask[8]={128,64,32,16,8,4,2,1};
  int bit,byte,i,m;
  b=calloc(4096,1);
  for(i=m=0; i<n; i++)
    {
      byte = a[i]/8;
      bit  = a[i]%8;
      if((b[byte] & mask[bit])==0)
        { m++;  b[byte] |= mask[bit]; }
    }
  free(b);
  return m;
}
```

Программа 4_09a.pas

```
program dif;
const
  a0:array [0..4] of integer=(0,0,0,0,0);
  a1:array [0..4] of integer=(1,1,1,1,1);
  a2:array [0..4] of integer=(0,1,1,1,1);
  a3:array [0..4] of integer=(0,0,1,1,2);
  a4:array [0..4] of integer=(0,1,2,3,4);
  a5:array [0..4] of integer=(1,2,3,4,5);
procedure sort(var a:array of integer);
  begin
  end;
function difference(a:array of integer):integer;
  var  i,m:integer;
  begin
    sort(a); m:=1;
    for i:=0 to High(a)-1 do
      if a[i] <> a[i+1] then inc(m);
```

```
    difference:=m;
  end;
begin
  write('Количество разных чисел в этом массиве равно ');
  write(difference(a5));
  readln;
end.
```

Программа 4_09b.pas

```
program dif1;
const
  a0:array [0..4] of integer=(0,0,0,0,0);
  a1:array [0..4] of integer=(1,1,1,1,1);
  a2:array [0..4] of integer=(0,1,1,1,1);
  a3:array [0..4] of integer=(0,0,1,1,2);
  a4:array [0..4] of integer=(0,1,2,3,4);
  a5:array [0..4] of integer=(1,2,3,4,5);

function difference(a:array of integer):integer;
  var
    i,j,k0,m,n:integer;
  begin
    k0:=0; n:=High(a);
    for i:=0 to n do
      if a[i]=0 then begin k0:=1;  break; end;
    for i:=0 to n-1 do
      begin
        if a[i]=0 then continue;
        for j:=i+1 to n do
          if a[i]=a[j] then begin a[i]:=0; break; end;
      end;
    m:=0;
    for i:=0 to n do
      if a[i]<>0 then inc(m);
   difference:= m+k0;
end;
begin
  write(' Количество разных чисел в этом массиве равно ');
```

```
  write(difference(a5));
  readln;
end.
```

Задание 4.10. Перемешивание "колоды карт"

Практически все карточные игры нуждаются в процедуре тасования колоды карт.

Совет 1 (общий)

Для кодировки колоды карт в памяти ЭВМ можно предложить, как минимум, два варианта. Например, можно связать с каждой картой числовой номер из диапазона [0,35] (или [0,51]). Предварительно надо договориться о порядке следования мастей (например, пики, трефы, бубны, черви) и карт внутри каждой масти (например, по силе карты: 6, 7, 8, 9, 10, валет, дама, король, туз). Вместо числового массива можно использовать строковый массив, в котором каждая карта представлена двух-, трехсимвольным обозначением (например, "6Т" — шестерка треф, "10Ч" — десятка червей, "ТП" — туз пик).

Совет 2 (общий)

Одна из наиболее простых идей перемешивания элементов массива заключается в использовании датчика случайных чисел. По двум очередным случайным числам (`i`, `j`) `i`-й и `j`-й элементы массива меняются местами. Если эту процедуру повторять достаточно долго, например 10 000 раз, то "колода карт" окажется хорошо перетасованной.

Совет 3 (общий)

Процедуру перемешивания можно ускорить, если в теле цикла обращаться к датчику случайных чисел не два раза, а один, и менять выпавшую `i`-ю "карту", например с первой или последней.

Совет 4 (общий)

Чтобы избежать повторения расклада при повторных играх, процедура перемешивания должна прибегать к возмущению датчика случайных чисел (оператор `randomize`).

Программа 4_10.bas

```
DECLARE SUB MIXER(B() AS INTEGER)
DEFINT A-Z
DIM A(36)
CLS
PRINT "Упорядоченная колода:"
FOR K=0 TO 35: A(K)=K: PRINT USING "####";A(K); : NEXT K
```

```
FOR J=0 TO 4
  MIXER A()
  PRINT "Перетасованная колода:"
  FOR K=0 TO 35: PRINT USING "####";A(K); : NEXT K
NEXT J
END

SUB MIXER(B() AS INTEGER)
  DEFINT A-Z
  RANDOMIZE INT(32767*RND)
  FOR J=0 TO 10000
    I=INT(35*RND+.5)
    TMP=B(0)
    B(0)=B(I)
    B(I)=TMP
  NEXT J
END SUB
```

Программа 4_10.c

```
#include <stdlib.h>
#include <conio.h>
void mixer(char *b);
  main()
{
  char j,k, a[36];
  clrscr();
  printf("\nУпорядоченная колода:\n");
  for (k=0; k<36; k++)
    {
      a[k]=k;
      printf("%4d",a[k]);
    }
  for(j=0; j<5; j++)
    {
       mixer(a);
       printf("\nПеретасованная колода:\n");
       for(k=0; k<36; k++)
       printf("%4d",a[k]);
```

```
    }
  getch();
}
void mixer(char *b)
{
  int j;
  char i,tmp;
  randomize();
  for (j=0; j<10000; j++)
    {
      i=random(36);
      tmp=b[0];
      b[0]=b[i];
      b[i]=tmp;
    }
  return;
}
```

Программа 4_10.pas

```
program mix_card;
uses Crt;
var
  j,k:byte;
  a:array [0..35] of byte;
procedure mixer(var b:array of byte);
  var
    i,j:integer;
    tmp:byte;
  begin
    randomize;
    for j:=0 to 10000 do
    begin
      i:=random(36);
      tmp:=b[0];
      b[0]:=b[i];
      b[i]:=tmp;
    end;
  end;
```

```
begin
  clrscr;
  writeln('Упорядоченная колода:');
  for k:=0 to 35 do
    begin
      a[k]:=k;
      write(a[k]:4);
    end;
  writeln;
  for j:=0 to 4 do
  begin
    mixer(a);
    writeln('Перетасованная колода:');
    for k:=0 to 35 do
      write(a[k]:4);
    writeln;
  end;
  readln;
end.
```

Задание 4.11. Игра в НИМ

Правила игры:

- в начале игры в каждой из `n` кучек находится отличное от 0 количество каких-то предметов (например, спичек);
- два игрока ходят по очереди. За один ход разрешается взять любое, отличное от 0, количество предметов из любой кучки;
- выигрывает тот, кто заберет содержимое последней оставшейся кучки.

Совет 1 (общий)

Имеет место выигрышная ситуация, если поразрядная сумма по модулю 2 двоичных чисел, представляющих предметы в кучках, отлична от 0. Выигрышным ходом считается тот, после которого указанная сумма станет равной 0. При ненулевой поразрядной сумме такой ход всегда существует и их может оказаться даже несколько. Если ситуация такова, что выигрышный ход сделать нельзя (это имеет место при нулевой поразрядной сумме), то можно потянуть время, взяв, например, один предмет из кучки с наибольшим количеством предметов.

Совет 2 (общий)

Программу игры целесообразно разбить на четыре процедуры:

- `start`, осуществляющую выбор количества кучек и начальный расклад;

- `user`, обеспечивающую диалоговое взаимодействие с пользователем и ввод его хода;
- `computer`, реализующую ход программы (для подсчета поразрядной суммы использовать операцию `xor`);
- `print_xod`, выводящую текущее состояние предметов в кучках.

Совет 3 (общий)

В игровых программах очень важно оформление, но мы ограничимся самыми скромными средствами протоколирования игры. Количество кучек будем выбирать случайным образом в диапазоне от 3 до 5, число предметов в каждой кучке тоже ограничим интервалом [1,12]. Каждый ход будем фиксировать в виде строки красного (после хода компьютера) или зеленого (после хода игрока) цвета, отражающей текущее содержимое каждой кучки.

Программа 4_11.bas

```
DECLARE SUB USER()
DECLARE SUB START()
DECLARE SUB COMPUTER()
DECLARE SUB PRINTXOD(col%,msg$)
  DEFINT A-Z
  DIM SHARED N
  DIM SHARED B AS INTEGER,S AS INTEGER,M AS INTEGER,Q AS INTEGER
  N=5: Q=12
  DIM SHARED A(N) AS INTEGER,I AS INTEGER,J AS INTEGER,K AS INTEGER
  M=1
  START
m1:
  USER
  COMPUTER
  GOTO m1
END

SUB COMPUTER
S=0
  FOR I=0 TO K-1: S=S XOR A(I): NEXT I
  IF S=0 THEN
    J=0: S=A(0)
    FOR I=1 TO K-1
      IF S<A(I) THEN S=A(I): J=I
      B=1
```

```
    NEXT I
  ELSE
    FOR J=0 TO K-1
      B=A(J)-(A(J) XOR S)
      IF B>=0 THEN EXIT FOR
     NEXT J
  END IF
  A(J)=A(J)-B
  PRINTXOD 4, "Победил компьютер"
END SUB

SUB PRINTXOD(col%,msg$)
COLOR col%, 0
  FOR I=0 TO K-1
    LOCATE M+1,3*I+1: PRINT USING "###";A(I)
  NEXT I
  M=M+1: IF M>23 THEN M=2: CLS: PRINTXOD col%, msg$
  S=0
  FOR J=0 TO K-1: S=S+A(J): NEXT J
  IF S<>0 THEN EXIT SUB
  LOCATE M,2: PRINT msg$
  SLEEP
  STOP
END SUB

SUB START
CLS : RANDOMIZE (VAL(RIGHT$(TIME$, 2)))
  K=INT(RND*(N-3))+3: ' число кучек
  FOR I=0 TO K-1
    A(I)=INT(RND*Q)+1
  NEXT I
  LOCATE 1,2 : PRINT "Начало игры"
  PRINTXOD 4,""
END SUB

SUB USER
COLOR 2,0
  LOCATE M,20
  PRINT "Ваш ход (кучка - сколько берем):"
```

```
M2:
  LOCATE M,33:    PRINT "       "
  LOCATE M,33:    INPUT J
  IF (J<1) OR (J>K) OR (A(J-1)=0) THEN GOTO M2
m3:
  LOCATE M,35:   PRINT "-    "
  LOCATE M,37:   INPUT B
  IF (B<1) OR (B>A(J-1)) THEN GOTO m3
  A(J-1)=A(J-1)-B
  SLEEP
  PRINTXOD 2, "Вы победили"
END SUB
```

Программа 4_11.c

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void print_xod(int color,char *msg);
void start(void);
void user(void);
void computer(void);
#define N 5
#define Q 12
  int i,j,k,b,s,m=1;
  int a[N];

  main()
{
  start();
m1:
  user();
  computer();
  goto m1;
}
void start(void)
{
  clrscr();  randomize();
  k=random(N-3)+3;   /* число кучек */
```

```
  for(i=0; i<k; i++)  a[i]=random(Q)+1;
  print_xod(RED,"Начало игры");
}
void user(void)
{
  textcolor(GREEN);
  gotoxy(20,m-1);  cprintf("Ваш ход (кучка - сколько берем):");
m2:
  gotoxy(33,m);  cprintf("       ");
  gotoxy(33,m);  scanf("%d",&j);
  if((j<1) || (j>k) || (a[j-1]==0)) goto m2;
m3:
  gotoxy(35,m);  cprintf("-    ");
  gotoxy(37,m);  scanf("%d",&b);
  if((b<1) || (b>a[j-1])) goto m3;
  a[j-1]-=b;
  print_xod(GREEN,"Вы победили");
  return;
}
void computer(void)
{
  for(s=0, i=0; i<k; i++) s ^= a[i];
  if(s==0)
    {
      for(i=1,j=0,s=a[0]; i<k; i++)
      if(s<a[i]) {s=a[i];  j=i;}
      b=1;
    }
  else
    for(j=0; j<k; j++)  {b=a[j]-(a[j]^s);  if(b>=0) break;}
  a[j]-=b;
  print_xod(RED,"Победил компьютер");
  return;
}
void print_xod(int color,char *msg)
{
  textcolor(color);
  for(i=0; i<k; i++)
    {gotoxy(3*i+1,m);  cprintf("%3d",a[i]);}
```

```
  m++;
  if(m>23) { m=2; clrscr(); print_xod(color, msg);}
  for(j=0,s=0; j<k; j++) s+=a[j];
  if(s!=0) return;
  gotoxy(1,m);  cprintf("%s",msg);
  getch();
  exit(0);
}
```

Программа 4_11.pas

```
program nim;
uses Crt;
const
  n=5;  q=12;
var
  i,j,k,b,s,m:integer;
  a:array [1..n] of byte;
label m1;
procedure print_xod(color:integer);
  begin
    textcolor(color);
    for i:=1 to k do  begin
      gotoxy(3*i,m);  write(a[i]:2);  end;
    m:=m+1;
if m > 23 then begin clrscr; m:= 2; print_xod(color); end;
  end;
procedure start;
  begin
    clrscr;     randomize;
    k:=random(n-3)+3;     {число кучек}
    m:=1; if m>23 then begin
    clrscr; m:=2;
    print_xod(color);
    end;
    for i:=1 to k do a[i]:=random(q)+1;
    print_xod(RED);
  end;
procedure user;
  label m2,m3;
```

```
  begin
    textcolor(GREEN);
    gotoxy(20,m-1);    write('Ваш ход (кучка - сколько берем):');
m2: gotoxy(33,m);  write('       ');
    gotoxy(33,m);  read(j);
    if(j<1) or (j>k) or (a[j]=0) then goto m2;
m3: gotoxy(35,m);  write('-    ');
    gotoxy(37,m);  read(b);
    if(b<1) or (b>a[j]) then goto m3;
    a[j]:=a[j]-b;
    print_xod(GREEN);
  end;
procedure computer;
  begin
    s:=0;
    for i:=1 to k do s:=s xor a[i];
    if s=0 then begin
      s:=a[1]; j:=1;
      for i:=2 to k do
        if s<a[i] then begin
          s:=a[i];  j:=i;  end;
        b:=1;
    end
    else
      for j:=1 to k do begin
        b:=a[j]-(a[j] xor s);
        if b>=0 then break;
      end;
    a[j]:=a[j]-b;
    print_xod(RED);
  end;
begin
  start;
m1 : user;
  s:=0;
  for j:=1 to k do s:=s+a[j];
  if s=0 then
  begin
    gotoxy(1,m);
```

```
    write('Вы победили');
    readln;    readln;
    exit;
  end;
  computer;
  s:=0;
  for j:=1 to k do s:=s+a[j];
  if s<>0 then goto m1;
  gotoxy(1,m);
  write('Победил компьютер');
  readln;    readln;
end.
```

Задание 4.12. Игра "крестики-нолики"

Составить программу для игры в крестики-нолики на поле размером 3×3. Правила игры: играют двое, делая последовательные ходы в клетках игрового поля. Первый игрок ставит крестики, второй — нолики. Выигрывает тот, кому первому удастся разместить три своих символа на одной из восьми игровых линий (три горизонтали, три вертикали, две диагонали). Если на поле после девятого хода выигрышная ситуация не выстроена, то считается, что игра закончилась вничью. Многими докомпьютерными поколениями подтверждено, что при "правильном" поведении обоих игроков, каждому из них обеспечена, как минимум, ничья. И только при явной ошибке партнера создается выигрышная ситуация для его оппонента.

Беспроигрышная тактика для первого игрока заключается в соблюдении трех следующих правил:

- первым ходом захватить центр, т. к. центральная клетка контролирует максимальное число линий (4 из 8);
- вторым ходом поставить крестик в один из углов, расположенных на вертикальной или горизонтальной линии, занятой первым ходом противника (этот ход контролирует 3 линии и создает возможность выигрыша по диагонали);
- третий и дальнейшие ходы делаются по следующей схеме:
  - a) если обнаружена линия, на которой расположены два крестика и есть свободное поле (выигрышная ситуация), заполните его;
  - b) если ситуация a) не обнаружена, но имеет место выигрышная позиция для противника, воспрепятствуйте этому;
  - c) если не обнаружена ни ситуация a), ни ситуация b), то поставьте крестик на любое свободное поле.

Для второго игрока беспроигрышная тактика заключается в соблюдении следующих правил:

- если центр не был занят первым ходом, то займите его;
- если центр занят, то нужно поставить нолик в **любую** из четырех угловых клеток (другой ход приведет к проигрышу);
- второй и последующие ходы выполняются в точном соответствии с приведенной ранее схемой:
  - a) если обнаружена линия, ведущая к победе, сделайте решающий ход;
  - b) если своего выигрышного хода нет, воспрепятствуйте выигрышу первого игрока;
  - c) если не обнаружена ни ситуация a), ни ситуация b), то ставьте нолик на любое свободное поле.

Совет 1 (общий)

Несмотря на относительную простоту тактики игры программа "крестики-нолики" довольно сложна, и пройдет немало времени, прежде чем заработает придуманный вами вариант. Остановимся на некоторых деталях приводимых ниже программ.

Выбор конфигурации рабочего поля. Не будем усложнять программу графическими процедурами, остановимся лишь на работе дисплея в текстовом режиме. Для того чтобы приблизить конфигурацию рабочего поля к квадрату, разумно выбрать соотношение сторон клетки 5 (по горизонтали) к 3 (по вертикали) и размещать символ в центральной позиции такого прямоугольника.

Нумерация позиций игрового поля, привязка к экрану и хранение информации о ходах. Пронумеруем клетки рабочего поля слева направо и сверху вниз от 1 до 9 и сопоставим содержимое центра каждой клетки с элементом целочисленного массива `a`:

- `a[i] = 0`, если клетка свободна;
- `a[i] = 2`, если клетка занята "крестиком";
- `a[i] =-2`, если клетка занята "ноликом".

Центры клеток в относительных номерах столбцов (`x`) и строк (`y`) имеют следующие координаты:

- клетка 1 (x=3, y=2) клетка 4 (x=3, y=4) клетка 7 (x=3, y=6)
- клетка 2 (x=7, y=2) клетка 5 (x=7, y=4) клетка 8 (x=7, y=6)
- клетка 3 (x=11,y=2) клетка 6 (x=11,y=4) клетка 7 (x=11,y=6)

В программе массив `pos` заполнен элементами, соответствующими экранным координатам позиций игрового поля.

Нумерация игровых линий. Пронумеруем линии числами от 1 до 8 следующим образом:

- линия 1 (верхняя горизонталь) проходит через клетки (1,2,3);
- линия 2 (средняя горизонталь) проходит через клетки (4,5,6);

- линия 3 (нижняя горизонталь) проходит через клетки (7,8,9);
- линия 4 (левая вертикаль) проходит через клетки (1,4,7);
- линия 5 (средняя вертикаль) проходит через клетки (2,5,8);
- линия 6 (правая вертикаль) проходит через клетки (3,6,9);
- линия 7 (1-я диагональ) проходит через клетки (1,5,9);
- линия 8 (2-я диагональ) проходит через клетки (3,5,7).

Обратите внимание на массив `lines`, каждая тройка элементов которого соответствует одной из игровых линий.

Отображение игрового поля. Воспроизведение контуров игрового поля осуществляется выводом семи строковых констант, состоящих из нужной цепочки символов псевдографики. Для отображения сделанных ходов организуем цикл перебора элементов массива `a` и выводим в центрах каждой клетки символ, соответствующий значению элемента `a[j]`. В программе для этой цели использована процедура `show(j,k)`, которая совмещает описанные выше действия с предварительной засылкой числа `k` в элемент `a[j]` (параметр `k` может принимать значения +2 или –2, в зависимости от того, кем был сделан текущий ход). После отображения игрового поля курсор переводится в центр, т. к. из него можно добраться до любой игровой клетки самым быстрым способом.

Ввод хода игрока. Для индикации хода человека используется процедура `input` (в программе на QBasic — `USER`, т. к. слово `INPUT` является служебным). Она предоставляет возможность перемещать курсор по центральным позициям игрового поля с помощью стрелок на клавиатуре. В выбранной позиции человек должен нажать клавишу <Enter>. Перемещение за пределы игрового поля пресекается соответствующими проверками. Нажатие непредусмотренных клавиш игнорируется с выдачей звукового сигнала (вывод символа `beep` с кодом 7).

Выполнение первого хода программы (в приводимых ниже программах первый ход делает компьютер, реализация второго варианта вносит очень небольшие изменения в текст головной программы). Реализуется непосредственным обращением к процедуре `show(5,2)`.

Выполнение второго хода компьютера. На любой из возможных восьми ходов человека предусмотрен фиксированный ответ — если человек выбрал клетку с номером `j` (`j = 1, 2, 3, 4, 6, 7, 8` или `9`), то второй ход программы определяется значением элемента массива `b[j]`.

Выполнение последующих ходов компьютера. Осуществляется процедурой `step345`, действия которой сводятся к следующим акциям:

- попытаться совершить свой выигрышный ход, если он возможен;
- попытаться воспрепятствовать ходу соперника, если обнаружена угрожающая ситуация;
- занять любое свободное поле, если ничего другого не остается.

Определение выигрышной ситуации. Организуем цикл перебора игровых линий с суммированием элементов `a[j]`, соответствующих клеткам, через которые проходит линия. Если полученная сумма равна 4, то на данной линии находятся два крестика и имеется свободное поле. В него и надо поставить

очередной крестик, чтобы победить. Этот анализ выполняется с помощью функции `xod(k1 = 2,k2 = 2)`. Она пытается найти линию, в которой сумма элементов `a[j]` равна `2*k1`, и, если таковая находится, заносит в "свободный" элемент линии число `k2`. Функция `xod` принимает значение "истина", если победный ход был сделан.

Определение угрожающей ситуации со стороны второго игрока. Организуем аналогичный цикл и, если получена сумма, равная –4, то найдена линия, в которой находится два нолика и имеется свободная позиция. В нее и нужно поставить очередной крестик, чтобы воспрепятствовать выигрышу соперника. Этот анализ тоже выполняет функция `xod`, но теперь ее аргументы `k1 = -2` и `k2 = 2`. Если угрожающую линию удалось закрыть, то функция `xod` принимает значение "истина".

Определение свободного поля. Если речь идет о линии, то номера перебираемых элементов определяются соответствующей тройкой из массива `lines`. Если нужно сделать несущественный ход, то можно перебирать элементы массива `a` подряд.

Определение результата игры. В случае обнаружения выигрышной позиции надо зафиксировать победный ход и сообщить о победе компьютера. Для обнаружения ничейной ситуации можно ограничиться подсчетом абсолютных значений элементов `a[j]`. После девятого хода, заполняющего рабочее поле, там находится пять величин, соответствующих ходам программы (+2), и четыре величины, фиксировавших ходы человека (–2). Так что общая сумма элементов (по модулю) будет равна 18. Конечно, о ничейном результате можно было бы догадаться и раньше, но такой анализ несколько утяжелил бы программу.

Совет 2 (QBasic)

Обратите внимание на оператор `LOCATE`. Во-первых, в нем, в отличие от процедуры `gotoxy`, координаты `x` и `y` переставлены местами (`y` — строка, `x` — столбец). Во-вторых, кроме местоположения курсора, здесь добавлен третий параметр, определяющий видимость курсора. Если этого не сделать, то игрок не будет видеть позицию, в которой он намеревается поставить нолик.

Совет 3 (Си, Паскаль)

Некоторая экономия в объеме программы отображения игрового поля достигнута за счет использования окна вывода (`window`). Задавая нужные значения констант (`x0`, `y0`), можно переместить игровое поле в другое место экрана.

Программа 4_12.bas

```
DECLARE SUB STEP345()
DECLARE SUB RESULT(s$)
DECLARE SUB CURIND(cur%,porog%,dcur%,dind%)
DECLARE SUB SHOW(k%,c%)
DECLARE SUB USER()
```

```
DECLARE FUNCTION XOD!(k%,k1%)
DEFINT A-Z
DIM SHARED POS1(1 TO 18) AS INTEGER,lines(0 TO 23) AS INTEGER
DIM SHARED A(1 TO 9) AS INTEGER,b(1 TO 9) AS INTEGER
DIM SHARED x0 AS INTEGER,y0 AS INTEGER
DIM SHARED Ind AS INTEGER,CurX AS INTEGER,CurY AS INTEGER

FOR k=1 TO 9: A(k)=0: NEXT k
DATA 3,1,1,1,5,3,1,7,3
FOR k=1 TO 9: READ b(k): NEXT k
x0=1: y0=1

DATA 3,2,7,2,11,2,3,4,7,4,11,4,3,6,7,6,11,6
FOR k=1 TO 18: READ POS1(k): NEXT k

DATA 1,2,3,4,5,6,7,8,9,1,4,7,2,5,8,3,6,9,1,5,9,3,5,7
FOR k=0 TO 23: READ lines(k): NEXT k

CLS
SHOW 5,2        ' Первый ход программы
USER            ' Ввод 1-го хода игрока
FOR J=1 TO 9    ' Второй ход программы
  IF A(J)=-2 THEN SHOW b(J),2: EXIT FOR
NEXT J
m2: USER        ' Ввод последующих ходов игрока
STEP345         ' Последующие ходы программы
GOTO m2
END

DEFSNG A-Z
SUB CURIND(cur%,porog%,dcur%,dind%)
  IF cur<>porog THEN cur=cur+dcur: Ind=Ind+dind
END SUB

DEFSNG A-Z
SUB RESULT(s$)
  LOCATE 1,40: PRINT s$: END
END SUB

DEFINT A-Z
```

```
SUB SHOW(k%,c%)
DEFINT A-Z
  CLS
  LOCATE x0,y0:      PRINT "+-----------+"
  LOCATE x0+1,y0:    PRINT "|   |   |   |"
  LOCATE x0+2,y0:    PRINT "|---|---|---|"
  LOCATE x0+3,y0:    PRINT "|   |   |   |"
  LOCATE x0+4,y0:    PRINT "|---|---|---|"
  LOCATE x0+5,y0:    PRINT "|   |   |   |"
  LOCATE x0+6,y0:    PRINT "+---+---+---+"
  A(k%)=c%
  FOR J=1 TO 9
    LOCATE y0-1+POS1(J*2),x0-1+POS1(J*2-1)
    IF A(J)=2 THEN PRINT "X"
    IF A(J)=-2 THEN PRINT "0"
  NEXT J
  CurX=x0+6: CurY=y0+3: Ind=5
  LOCATE CurY,CurX,1
END SUB

SUB STEP345
DEFINT A-Z
  IF XOD(2,2)=1 THEN RESULT "Победа компьютера": END
  IF XOD(-2,2)=1 THEN EXIT SUB
  FOR J=1 TO 9
    IF A(J)=0 THEN SHOW J,2: EXIT FOR
  NEXT J
END SUB

DEFSNG A-Z
SUB USER
DEFINT A-Z
  Left=75: Right=77: Up=72: Down=80: Enter=13
  k=0: FOR J=1 TO 9: k=k+ABS(A(J)): NEXT J
  IF k=18 THEN RESULT "Боевая ничья": END
m: ch$=INKEY$: IF LEN(ch$)=0 THEN GOTO m
  SELECT CASE ASC(RIGHT$(ch$,1))
    CASE Left:   CURIND CurX,3,-4,-1
    CASE Right:  CURIND CurX,11,4,1
```

```
    CASE Up:     CURIND CurY,2,-2,-3
    CASE Down:   CURIND CurY,6,2,3
    CASE Enter:  IF A(Ind)=0 THEN SHOW Ind,-2: EXIT SUB
    CASE ELSE:   BEEP
  END SELECT
  LOCATE y0-1+CurY,x0-1+CurX,1
  USER
END SUB

DEFINT A-Z
FUNCTION XOD(k%,k1%)
DIM J AS INTEGER,m AS INTEGER,p AS INTEGER
  XOD=0
  FOR J=0 TO 7
    m=J*3
    IF A(lines(m))+A(lines(m+1))+A(lines(m+2))=2*k% THEN
       XOD=1
       FOR p=m TO m+2
         IF A(lines(p))=0 THEN
            SHOW lines(p),k1%
            EXIT FUNCTION
         END IF
       NEXT p
    END IF
  NEXT J
END FUNCTION
```

Программа 4_12.c

```
#include <stdio.h>
#include <conio.h>
void input(void);
void cur_ind(int *cur,int porog,int dcur,int dind);
void step345(void);
int xod(int k, int k1);
void show(int k, int c);
void result(char *s);
  int a[9]={0,0,0,0,0,0,0,0,0},
      b[9]={3,1,1,1,5,3,1,7,3},
```

```
      x0=1, y0=1, j, CurX, CurY, Ind;
  main()
{
  clrscr();
  window(x0,y0,x0+13,y0+7);
  show(5,2);                /* Первый ход программы */
  input();                  /* Ввод 1-го хода игрока */
  for(j=0; j<9; j++)        /* Второй ход программы */
    if (a[j]==-2) { show(b[j],2); break; }
m2:
  input();        /* Ввод последующих ходов игрока */
  step345();      /* Последующие ходы программы */
  goto m2;
}
/*--------------------------------------*/
void result(char *s)
{
  window(40,1,60,2);
  puts(s);
  getch();
  exit(0);
}
/*--------------------------------------*/
void show(int k,int c)
{
  char pos[18]={3,2,7,2,11,2,3,4,7,4,11,4,3,6,7,6,11,6};
  char j;
  clrscr();
  printf("+-----------+\n");
  printf("|   |   |   |\n");
  printf("|---|---|---|\n");
  printf("|   |   |   |\n");
  printf("|---|---|---|\n");
  printf("|   |   |   |\n");
  printf("+-----------+");
  a[k-1]=c;
  for(j=0; j<9; j++)
    {
      gotoxy(x0-1+pos[j*2],y0-1+pos[j*2+1]);
```

```
      if(a[j]==+2) printf("X");
      if(a[j]==-2) printf("0");
    }
  CurX=x0+6;
  CurY=y0+3;
  Ind=5;
  gotoxy(CurX,CurY);
}
/*-------------------------------------------*/
int xod(int k, int k1)
{
  char line[24]={1,2,3,4,5,6,7,8,9,1,4,7,2,5,8,3,6,9,1,5,9,3,5,7},
       j,m,p;
  for(j=0; j<8; j++)
    {
      m=j*3;
      if(a[line[m]-1]+a[line[m+1]-1]+a[line[m+2]-1]==2*k)
        {
          printf("\nm=%d",m);
          for(p=m; p<m+3; p++)
            if(a[line[p]-1]==0)
              {
                show(line[p],k1);
                return 1;
              }
        }
    }
  return 0;
}
/*-------------------------------------------*/
void step345(void)
{
  if(xod( 2,2)) result("Победа компьютера");
  if(xod(-2,2)) return;
  for(j=0; j<9; j++)
    if(a[j]==0)
      {
        show(j+1,2);
        break;
```

```
      }
}
/*---------------------------------------------*/
void cur_ind(int *cur, int porog, int dcur, int dind)
{
  if( *cur != porog)
    {
      *cur += dcur;
      Ind += dind;
    }
}
/*---------------------------------------------*/
void input(void)
{
  int ch, j, k;
  for(k=0, j=0; j<9; j++) k+=abs(a[j]);
  if(k==18) result("Боевая ничья");
  ch=getch();  if(ch==0) ch=getch();
  switch (ch)
  {
    case 75: cur_ind(&CurX, 3,-4,-1); break; /*Left*/
    case 77: cur_ind(&CurX,11, 4, 1); break; /*Right*/
    case 72: cur_ind(&CurY, 2,-2,-3); break; /*Up*/
    case 80: cur_ind(&CurY, 6, 2, 3); break; /*Down*/
    case 13: printf("\nInd=%d",Ind);         /*Enter*/
             if(a[Ind-1]==0) { show(Ind,-2); return;}
             else  printf("%c",0x7);  break;
    default: printf("%c",0x7);
  }
  gotoxy(x0-1+CurX,y0-1+CurY);
  input();
}
```

Программа 4_12.pas

```
program krestiki;
uses Crt;
label m2;
const
```

```
  a:array [1..9] of integer = (0,0,0,0,0,0,0,0,0);
  b:array [1..9] of byte = (3,1,1,1,5,3,1,7,3);
  x0=5;     y0=5;
var
  j, CurX, CurY, Ind : word;
procedure result(s:string);
begin
  window(40,1,60,2);  write(s);
  readln;  halt;
end;
procedure show(k,c:integer);
const
  pos:array [1..18] of byte = (3,2,7,2,11,2,3,4,7,4,11,4,3,6,7,6,11,6);
var
  j:byte;
  x,y:word;
begin
  clrscr;
  writeln('+---+---+---+');
  writeln('|   |   |   |');
  writeln('|---|---|---|');
  writeln('|   |   |   |');
  writeln('|---|---|---|');
  writeln('|   |   |   |');
    write('+---+---+---+');
  a[k]:=c;
  for j:=1 to 9 do
    begin
      x:=pos[(j-1)*2+1];   y:=pos[(j-1)*2+2];
      gotoxy(x,y);
      if a[j]=+2 then write('X');
      if a[j]=-2 then write('0');
    end;
  CurX:=7;   CurY:=4;    Ind:=5;
  gotoxy(CurX,CurY);
end;
function xod(k,k1:integer):boolean;
const
  line:array [0..23] of byte=
```

```
       (1,2,3,4,5,6,7,8,9,1,4,7,2,5,8,3,6,9,1,5,9,3,5,7);
var
  j,m,p:byte;
begin
  xod:=false;
  for j:=0 to 7 do
    begin  m:=j*3;
      if a[line[m]]+a[line[m+1]]+a[line[m+2]]=2*k then
        begin
          xod:=true;
          for p:=m to m+2 do
            if a[line[p]]=0 then
              begin
                show(line[p],k1);  exit;
              end;
        end;
    end;
end;
procedure step345;
begin
  if xod( 2,2) then result('Победа компьютера');
  if xod(-2,2) then exit;
  for j:=1 to 9 do
    if a[j]=0 then
      begin
        show(j,2);  break;
      end;
end;
procedure input;
const
  Left=#75;  Right=#77;  Up=#72;  Down=#80;  Enter=#13;
var
  ch:char;
  j,k:byte;
procedure cur_ind(var cur:word;porog,dcur,dind:integer);
begin
  if cur <> porog then begin
  cur:=cur+dcur; Ind:=Ind+dind; end;
```

```
end;
begin
  k:=0; for j:=1 to 9 do k:=k+abs(a[j]);
  if k=18 then result('Боевая ничья');
  ch:=readkey;  if ch=#0 then ch:=readkey;
  case ch of
    Left:   cur_ind(CurX, 3,-4,-1);
    Right:  cur_ind(CurX,11, 4, 1);
    Up:     cur_ind(CurY, 2,-2,-3);
    Down:   cur_ind(CurY, 6, 2, 3);
    Enter:  if a[Ind]=0 then begin
               show(Ind,-2); exit; end else write(#7);
    else    write(#7);
  end;
  gotoxy(CurX,Cury);
  input;
end;
begin
  clrscr;   window(x0,y0,x0+13,y0+7);
  show(5,2);                { Первый ход программы }
  input;                    { Ввод 1-го хода игрока }
  for j:=1 to 9 do          { Второй ход программы }
    if a[j]=-2 then
    begin show(b[j],2); break; end;
m2: input;         { Ввод последующих ходов игрока }
    step345;       { Последующие ходы программы }
    goto m2;
end.
```

Задание 4.13. Слияние массивов

Составить процедуру, которая объединяет два предварительно упорядоченных по возрастанию массива, получая в результате массив с возрастающими по величине элементами.

Совет 1 (общий)

Если в объединяемых массивах находится соответственно `m` и `n` элементов, то процедура слияния потребует `m+n` шагов. На первом шаге сравниваются первые элементы сливаемых массивов и меньший из них переписывается в результирующий массив. Перед очередным сравнением необходимо переместить

текущий указатель в массиве, из которого на предыдущем шаге был выбран минимальный элемент. Однако предварительно разумно проверить, не исчерпан ли уже один из массивов.

Программа 4_13.bas

```
DECLARE SUB MERGE (A%(),NA%,B%(),NB%,C%())
DEFINT A-Z
CLS
NA=3:
DIM A(NA)
DATA 0,2,4
FOR K=0 TO NA-1: READ A(K): PRINT A(K); : NEXT K: PRINT
NB=4
DIM B(NB)
DATA 1,3,5,7
FOR K=0 TO NB-1: READ B(K): PRINT B(K); : NEXT K: PRINT
DIM C(NA+NB)
MERGE A(),NA,B(),NB,C()
FOR K=0 TO NA+NB-1: PRINT C(K); : NEXT K
END

SUB MERGE(A(),NA,B(),NB,C())
JA=0: JB=0
FOR JC=0 TO NA+NB-1
  IF JA=NA THEN GOTO MB
  IF JB=NB THEN GOTO MA
  IF A(JA)<B(JB) THEN GOTO MA
MB: C(JC)=B(JB): JB=JB+1: GOTO M1
MA: C(JC)=A(JA): JA=JA+1:
M1:
NEXT JC
END SUB
```

Программа 4_13.c

```
#include <stdio.h>
#include <conio.h>
void merge(int *a,int ka,int *b,int kb,int *c);
```

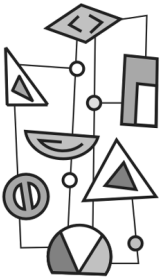
```
  main()
{
#define na 3
#define nb 4
  int j,a[na]={0,2,4},b[nb]={1,3,5,7},c[na+nb];
  clrscr();
  for(j=0; j<na; j++) printf("%4d",a[j]);
  printf("\n");
  for(j=0; j<nb; j++) printf("%4d",b[j]);
  printf("\n");
  merge(a,na,b,nb,c);
  for(j=0; j<na+nb; j++) printf("%4d",c[j]);
  getch();
}

void merge(int *a,int ka,int *b,int kb,int *c)
{
  int ja=0,jb=0,jc;
  for(jc=0; jc<ka+kb; jc++)
    {
      if (ja==ka) goto mb;
      if (jb==kb) goto ma;
      if (a[ja]<b[jb])goto ma;
mb:   c[jc]=b[jb]; jb++; continue;
ma:   c[jc]=a[ja]; ja++;
    }
}
```

Программа 4_13.pas

```
program merge2;
uses Crt;
const
  na=3;
  nb=2;
  a:array [0..na] of integer = (0,2,4,6);
  b:array [0..nb] of integer = (1,3,5);
var
  c:array [0..na+nb+1] of integer;
```

```
  j:integer;
procedure merge(a,b:array of integer;var c:array of integer);
var
  ja,jb,jc,na,nb,nc:integer;
label
  ma,mb;
begin
  na:=High(a); ja:=0;
  nb:=High(b); jb:=0;
  nc:=High(c); if nc < na+nb+1 then
  begin
    writeln('Массив с слишком мал');
    exit;
  end;
  for jc:=0 to na+nb+1 do
    begin
      if ja > na then goto mb;
      if jb > nb then goto ma;
      if a[ja] < b[jb] then goto ma;
mb:   c[jc]:=b[jb]; inc(jb); continue;
ma:   c[jc]:=a[ja]; inc(ja);
    end;
end;
begin
  clrscr;
  for j:=0 to na do write(a[j]:4);
  writeln;
  for j:=0 to nb do write(b[j]:4);
  writeln;
  merge(a,b,c);
  for j:=0 to na+nb+1 do write(c[j]:4);
  readln;
end.
```

Глава 5



Рекурсивные функции и процедуры

В простейшем варианте процедура (функция) считается рекурсивной, если она пытается вызвать сама себя. Математики нередко прибегают к рекурсивному определению функций:

```
n! = n*(n-1)!
```

Естественно, что при таком хождении "по кругу" должно быть предусмотрено условие выхода, иначе вычислительный процесс может продолжаться бесконечно долго. В примере с факториалом тело процедуры на Паскале может выглядеть следующим образом:

```
if n < 2 then fact:=1
else fact:=n*fact(n-1);
```

В более сложных вариантах рекурсивная процедура входит в состав двух или более процедур, последняя из которых вновь обращается к начальной:

```
A -> B -> A -> B -> A ...
```

В язык Паскаль, требующий описать любой объект до его использования, пришлось включить специальное опережающее объявление рекурсивных процедур, сопровождаемое служебным словом `forward`:

```
{ Опережающее объявление первой в цепочке процедуры A }
  procedure A(<список параметров>); forward;
{ Описание последней в цепочке рекурсивной процедуры B }
  procedure B(<список параметров>);
  begin
  ...........
    A(...);  { Вызов рекурсивной процедуры A }
     ...........
  end;
{ Описание первой в цепочке рекурсивной процедуры A }
  procedure A; { Здесь можно не повторять список аргументов }
  begin
  ............
```

```
    B(...);  { Вызов рекурсивной процедуры B }
     ............
  end;
```

Несмотря на все изящество рекурсивных программ, их работа сопряжена с повышенными затратами машинного времени и ресурсов по памяти. При каждом новом вызове рекурсивной процедуры приходится сохранять значения всех ее локальных переменных и выделять новые участки памяти для очередной порции локальных данных. Как правило, рекуррентный алгоритм с большими или меньшими усилиями можно превратить в обычный циклический процесс. Например, фрагмент программы, вычисляющей `n!`, может выглядеть следующим образом:

```
p:=1;
for i:=1 to n do p:=p*i;
fact:=p;
```

Однако, как бы ни относились разные программисты к рекурсии, следует признать, что некоторые рекурсивные программы выглядят очень эффектно и их алгоритмы оказываются намного более прозрачными (см., например, процедуру быстрой сортировки `quicksort` или игровую программу "`Ханойские башни`").

Задачи, советы и ответы

Задание 5.01. Числа Фибоначчи

История чисел Фибоначчи восходит к началу XIII в., когда итальянский математик Леонардо Пизанский нашел изящное решение задачи о размножении кроликов. Пусть в начале эксперимента имеется единственная пара — самец и самка, которые приносят каждый месяц приплод, состоящий также из самца и самки. Дети включаются в цикл продолжения рода через два месяца. Требуется определить количество пар, спустя `k` месяцев после начала эксперимента. Ситуация не так уж и сложна:

- в начале эксперимента — 1 родительская пара;
- через 1 месяц — 2 пары (родители и дети первого поколения);
- через 2 месяца — 3 пары (родители и дети двух поколений);
- через 3 месяца — 5 пар (родители, дети трех поколений, внуки от первого поколения);
- через 4 месяца — 8 пар (родители, дети четырех поколений, 2 пары внуков от первого поколения, 1 пара внуков от второго поколения);
- .....

Леонардо догадался, что числа указанной последовательности связаны рекуррентным соотношением:

$f_k = f_{k-2} + f_{k-1}$

В разных учебниках последовательность чисел Фибоначчи дополняют либо еще одной единицей, либо нулем и единицей:

1,1,2,3,5,8,13,21,34,55,....

0,1,1,2,3,5,8,13,21,34,.....

Последний вариант принят в приводимых ниже программах, отсчет порядковых номеров чисел Фибоначчи ведется от 0. Диапазон действия этих программ не так уж велик: `fib(22)=17711`, `fib(23)=28657` и уже 24-е число выходит за пределы предусмотренного диапазона. Если вас заинтересуют числа с большими номерами, то можно изменить тип функции `fib` или воспользоваться формулой Бинэ:

```
fib(k)=(((1+sqrt(5))/2)^k-((1-sqrt(5))/2)^k)/sqrt(5)
```

Наиболее известное применение чисел Фибоначчи — оптимальный выбор точек при поиске экстремума унимодальной функции на заданном отрезке. Рекурсивная реализация нахождения чисел Фибоначчи абсолютно неэффективна, т. к. в теле процедуры при каждой итерации происходит два обращения к этой же процедуре и объем вычислений с ростом `k` возрастает почти в геометрической прогрессии.

Программа 5_01.bas

```
DECLARE FUNCTION FIB%(N%)
INPUT "Задайте порядковый номер числа Фибоначчи - ",N%
PRINT "Число Фибоначчи с номером ";N%;" равно ";FIB%(N%)
END

FUNCTION FIB%(N%)
  IF N% < 2 THEN
     FIB%=N%
  ELSE
     FIB%=FIB%(N%-2)+FIB%(N%-1)
  END IF
END FUNCTION
```

Программа 5_01.c

```
#include <stdio.h>
#include <conio.h>
```

```
int fib(int k);
  main()
{
  int n;
  printf("\nЗадайте порядковый номер числа Фибоначчи - ");
  scanf("%d",&n);
  printf("\nЧисло Фибоначчи с номером %d равно %d",n,fib(n));
  getch();
}
int fib(int k)
{
  if(k<2) return k;
  return fib(k-2)+fib(k-1);
}
```

Программа 5_01.pas

```
program fibonacci;
var
  n:integer;
function fib(k:integer):integer;
begin
  if k<2 then fib:=k
  else fib:=fib(k-2)+fib(k-1);
end;
begin
  write('Задайте порядковый номер числа Фибоначчи - ');
  readln(n);
  writeln('Число Фибоначчи с номером ',n,' равно ',fib(n));
  readln;
end.
```

Задание 5.02. Наибольший общий делитель

Составить подпрограмму нахождения наибольшего общего делителя двух натуральных чисел.

Совет 1 (общий)

Алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел приписывают Евклиду. Он базируется на трех сравнительно тривиальных утверждениях:

```
НОД(k1,k2) = НОД(k2,k1)     // поэтому можно считать, что k1 < k2
НОД(0,k) = k                // неудивительно, т.к. 0 делится на все
НОД(k1,k2) = НОД(k3,k1)     // k3 — остаток от деления k2 на k1
```

В истинности последнего несложно убедиться, если записать:

```
k1*m + k3 = k2        // m - частное от деления k2 на k1
```

Если `k1` и `k2` делятся на свой НОД, то и `k3` делится на это же число без остатка. Поэтому алгоритм Евклида состоит из следующих шагов:

1. Если первое число `k1` больше второго, то поменять их местами.
2. Если `k1=0`, то вычисления прекращаются.
3. Находим остаток от деления `k2` на `k1` и заменяем им большее число.
4. Повторяем шаги, начиная с первого.

Совет 2 (общий)

За счет небольшой потери эффективности от перестановки большего и меньшего чисел можно отказаться. При этом подпрограмма лишний раз обратится к себе:

```
НОД(120,84)=НОД(84,120)=НОД(36,84)=НОД(12,36)=НОД(0,12)=12
```

Поэтому тело функции можно построить из единственного условного оператора.

Программа 5_02.bas

```
DECLARE FUNCTION NOD&(N1&,N2&)
PRINT "Введите 2 натуральных числа, разделяя их запятой"
INPUT "",M&,N&
PRINT "Их наибольший общий делитель равен ";NOD&(M&,N&)
END

FUNCTION NOD&(N1&,N2&)
  IF N1&=0 THEN
     NOD&=N2&
  ELSE
     NOD&=NOD&(N2& MOD N1&,N1&)
  END IF
END FUNCTION
```

Программа 5_02.c

```
#include <stdio.h>
#include <conio.h>
long nod(long n1,long n2);
  main()
{
  long m,n;
  printf("\nВведите 2 натуральных числа, разделяя их пробелом\n");
  scanf("%ld %ld",&n,&m);
  printf("Их наибольший общий делитель равен %ld",nod(m,n));
  getch();
}
long nod(long n1,long n2)
{
  if(n1==0) return n2;
  return nod(n2%n1,n1);
}
```

Программа 5_02.pas

```
program Euclid;
var
  n,m:longint;
function nod(n1,n2:longint):longint;
begin
  if n1=0 then nod:=n2
  else nod:=nod(n2 mod n1,n1);
end;
begin
  writeln('Введите два натуральных числа');
  readln(n,m);
  writeln('Их наибольший общий делитель равен ',nod(n,m));
  readln;
end.
```

Задание 5.03. Ханойские башни

Игра "Ханойские башни" была придумана французским математиком Э. Люка в конце XIX в. На подставке установлены три шпильки, которые

мы условно назовем A, B и C. На шпильку A надето k дисков разного диаметра таким образом, что они образуют "правильную" пирамиду (каждый диск, расположенный выше, имеет диаметр меньший, чем у всех, лежащих ниже). Цель игры заключается в переносе пирамиды на шпильку C. При этом за один ход разрешается переложить один диск, надевая его на одну из шпилек таким образом, чтобы верхний диск всегда был меньшего диаметра.

Можно показать, что число ходов, необходимое для решения этой задачи, равно 2^k-1. Для k=1 и k=2 это легко проверяется. Дальнейшие рассуждения выполняются по индукции: если это справедливо для любого k-1, то докажем справедливость утверждения о числе шагов и для любого k. Предположим, что мы умеем переносить пирамиду из k-1 дисков за $2^{k-1}-1$ ходов. Тогда для переноса башни из k дисков можно поступить следующим образом:

- переносим верхние k-1 диски со шпильки A на шпильку B;
- переносим оставшийся самый большой диск со шпильки A на шпильку C (это еще один ход);
- используя освободившуюся шпильку A в качестве рабочей, переносим пирамиду из k-1 диска со шпильки B на шпильку C.

Таким образом, число ходов, которое мы затратили для переноса пирамиды из k дисков, равно:

$$2^{k-1} - 1 + 1 + 2^{k-1} - 1 = 2^k-1$$

Совет 1 (общий)

Для построения рекурсивной программы необходимо последовательно уменьшать размерность задачи, сводя ее на каждой итерации к последовательности трех описанных выше перемещений. Для этой цели нам понадобятся две процедуры:

```
MoveAll(k,from,to,tmp)
MoveOne(from,to)
```

Первая из них выполняет шаги по переносу пирамиды из k дисков со шпильки from на шпильку to, используя в качестве рабочей шпильку tmp. Вторая переносит один диск со шпильки from на шпильку to.

Совет 2 (QBasic)

Переменная m% введена для нумерации ходов. Для того чтобы ее значение сохранялось после очередного перемещения одного диска в заголовке подпрограммы MoveOne использован указатель STATIC.

Совет 3 (Паскаль)

В программе нельзя использовать идентификатор to, т. к. он занят под служебное слово в операторе цикла for.

Программа 5_03.bas

```
DECLARE SUB MoveAll(k%,from$,to$,tmp$)
DECLARE SUB MoveOne(from$,to$)
CLS
INPUT "Введите количество дисков ",k%
MoveAll k%,"A","C","B"
END

SUB MoveAll (k%,from$,to$,tmp$)
  IF k%=0 THEN EXIT SUB
  MoveAll k%-1,from$,tmp$,to$
  MoveOne from$,to$
  MoveAll k%-1,tmp$,to$,from$
END SUB

SUB MoveOne(from$,to$) STATIC
  m%=m%+1
  PRINT USING "#### : &--> & ";m%;from$;to$
END SUB
```

Программа 5_03.c

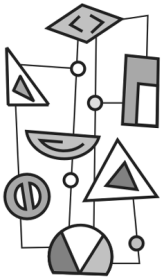
```
#include <stdio.h>
#include <conio.h>
void MoveAll(int k,char from,char to,char tmp);
void MoveOne(char from,char to);
  int m=0;
  main()
{
  int k;
  clrscr();
  printf("\nВведите количество дисков - ");
  scanf("%d",&k);
  MoveAll(k,'A','C','B');
  getch();
}

void MoveAll(int k,char from,char to,char tmp)
```

```
{
  if(k==0) return;
  MoveAll(k-1,from,tmp,to);
  MoveOne(from,to);
  MoveAll(k-1,tmp,to,from);
}

void MoveOne(char from,char to)
{
  m++;
  printf("\n%4d : %c --> %c",m,from,to);
}
```

Программа 5_03.pas

```
program Hanoi;
uses Crt;
var
  k:integer;
const
  m:integer=0;
procedure MoveOne(from,to1:char);
begin
  inc(m);
  writeln(m:4,from:2,' --> ',to1);
end;
procedure MoveAll(k:integer;from,to1,tmp:char);
begin
  if k = 0 then exit;
  MoveAll(k-1,from,tmp,to1);
  MoveOne(from,to1);
  MoveAll(k-1,tmp,to1,from);
end;
begin
  clrscr;
  write('Введите количество дисков - ');
  readln(k);
  MoveAll(k, 'A', 'C', 'B');
  readln;
end.
```

Глава 6



Подпрограммы (процедуры) и функции

Примеры программ в предыдущих разделах должны были подготовить вас к восприятию мысли о том, что в каждой задаче полезно вычленять логически завершенные фрагменты и оформлять их как отдельные программные единицы — подпрограммы (процедуры) или функции. Что это дает?

Во-первых, упрощается логика программ за счет вынесения многочисленных деталей за рамки основного алгоритма. Появляется возможность конструировать программу "сверху — вниз", составляя ее по началу из пустых блоков (заглушек), о которых известно только их функциональное назначение. Потом каждый из блоков детализируется, разбиваясь на более мелкие программные единицы и т. д. Подпрограммы, текст которых обычно не превышает одну-две страницы, гораздо легче анализировать и отлаживать. Во-вторых, выделение подпрограмм позволяет уменьшить общий объем программы особенно в тех случаях, когда к ним приходится многократно обращаться из разных точек. Наконец, многие подпрограммы могут быть использованы повторно при решении других задач.

Цель настоящей главы — вкратце систематизировать основные сведения, связанные с организацией и использованием подпрограмм (функции можно рассматривать как частный случай подпрограмм, результатом работы которых является единственное значение).

Любая программа, готовая к самостоятельному исполнению, состоит из единственной *головной* (или *главной*) программы, с которой начинается работа, и некоторого количества подчиненных ей *подпрограмм*. Головная программа непосредственно или косвенно может обратиться к любой подпрограмме, которые точно таким же образом могут вызвать любую подпрограмму, включая и себя. К головной программе никакая другая программная единица обратиться не может.

Подпрограммы, поставляемые в составе системы программирования, обычно называют *стандартными* или *системными*. Все остальные подпрограммы, которые мы пишем сами или заимствуем из каких-либо других источников, называют *нестандартными* или *пользовательскими*.

Текст программы на алгоритмическом языке, оформленный в виде дискового файла с соответствующим расширением (bas, c, cpp, pas), принято называть *исходным модулем*. В решении задачи может принимать участие один или

несколько исходных модулей. В системе QBasic такие модули сменяют друг друга в оперативной памяти с помощью оператора `CHAIN`. Система Turbo C (Borland C++) позволяет разбить текст исходной программы на несколько программных файлов, которые могут быть протранслированы автономно и объединены между собой с помощью специального сборочного файла — проекта. В системе Turbo Pascal отдельные подпрограммы могут быть вынесены в специальным образом оформленные модули, начинающиеся с оператора `Unit` и подключаемые к тексту исходной программы по директиве `Uses`.

Результатом трансляции (компиляции) каждого программного файла в системе Turbo C является *объектный модуль* — дисковый файл с расширением obj. В составе одного объектного модуля может находиться как головная функция, так и любое количество подчиненных ей пользовательских функций. С помощью сервисной программы `tlib.exe` объектные модули могут быть помещены в *библиотеку* — дисковый файл с расширением lib. Любая компонента библиотеки может быть использована при сборке готовой к исполнению программы.

Модули Паскаля, начинающиеся с оператора `Unit`, после компиляции превращаются в дисковые файлы с расширением tpu (от Turbo Pascal Unit) и играют роль, аналогичную библиотекам объектных модулей на языке Си.

На последней стадии обработки исходной программы в системах компилирующего типа (TC, TP) появляются готовые к исполнению программы в виде *загрузочных модулей* — дисковых файлов с расширением exe. Очень большие программы на языке Паскаль, целиком не помещающиеся в памяти и работающие по частям, сменяющим друг друга, состоят из exe-файла и дополнительного *оверлейного файла* с расширением ovl. Для создания последнего программист должен сам разбить задачу на части и оформить их соответствующим образом.

Оформление и вызов программных единиц в системе QBasic

Во входном языке системы QBasic нет специальных средств для выделения головной программы. Поэтому все, что не принадлежит специальным образом оформленным подпрограммам или функциям, относится к тексту головной программы. Дополнительной особенностью Бейсика является возможность использования как *внешних*, так и *внутренних* программных единиц. Тексты последних непосредственно входят в состав головной программы или тела любой другой программной единицы.

Описанию внутренней функции с именем, обязательно начинающимся с символов "`FN`", предшествует заголовок вида:

```
DEF FNs1s2...[(аргументы)]
```

Вслед за ним располагаются обычные операторы, участвующие в вычислении значения функции, завершающиеся обязательным присвоением вычисленного результата:

```
FNs1s2... = выражение
```

Признаком конца описания внутренней функции является оператор `END DEF`. В теле внутренней функции может находиться оператор `EXIT DEF`, обеспечивающий досрочное завершение вычислений при выполнении определенных условий.

Внутренняя функция, алгоритм вычисления которой сводится к единственному выражению, описывается одним оператором, после которого завершение `END DEF` не требуется:

```
DEF FNs1s2...[(аргументы)]=выражение
```

В операторах вычисления значения внутренней функции наряду с ее аргументами могут использоваться любые переменные той программной единицы, которой принадлежит внутренняя функция. Прибегать к ним, как правило, не рекомендуется, т. к. нет никакой гарантии, что два значения функции при одних и тех же аргументах будут совпадать. Между последовательными обращениями к функции значения использованных переменных головной программы могли измениться. Однако и исключать их использование вообще тоже нельзя. Примером полезного применения внутренней функции без аргументов может служить вычисление расстояния между двумя точками, которое может потребоваться в разных местах программы после модификации координат одной или обеих точек:

```
DEF FNd=SQR((X1-X2)^2+(Y1-Y2)^2+(Z1-Z2)^2)
```

Эффект от использования такой функции заключается в сокращении длины текста программы — 37 символов приведенного выражения в точках вызова функции заменяются всего на 3 символа.

Операторы вычисления значения внутренней функции срабатывают только при вызове функции из головной программы. Если Бейсик-интерпретатор попадает на них другим способом, то тело внутренней функции обходится.

Внешняя функция начинается с заголовка:

```
FUNCTION имя_функции[(аргументы)][STATIC]
```

Добавка `STATIC` требует, чтобы значения всех локальных переменных после очередного обращения к функции были сохранены и моглиут быть использованы при последующем обращении к функции. Если по условиям задачи необходимо сохранять значения только некоторых локальных переменных, то слово `STATIC` может быть использовано в теле функции при объявлении соответствующих переменных:

```
STATIC Q1 AS INTEGER, F AS DOUBLE
```

В теле функции, состоящем из обычных операторов, обязательно должно присутствовать присвоение:

```
имя_функции = выражение
```

Если имя функции используется в правой части оператора присвоения или в условном выражении, то вслед за ним в круглых скобках должны быть указаны аргументы и это означает рекурсивный вызов самой себя.

Признаком конца описания внешней функции является оператор `END FUNCTION`. Для досрочного выхода из подпрограммы вычисления функции используется оператор `EXIT FUNCTION`.

Тип внутренней или внешней функции определяется специальным символом, завершающим имя функции (`%` — короткий целый, `&` — длинный целый, `!` — короткий вещественный, `#` — длинный вещественный, `$` — строковый). Если имя функции завершается любым другим символом, то по умолчанию функции присваивается короткий вещественный тип.

Для вычисления значения внутренней или внешней функции достаточно использовать ее имя с набором фактических аргументов в качестве операнда выражения соответствующего типа.

Внутренняя подпрограмма, входящая в текст головной программы, обязательно должна начинаться с метки, на которую передается управление оператором `GOSUB`. Работа внутренней подпрограммы завершается оператором `RETURN`, после чего продолжается работа основной программы с оператора, следующего за `GOSUB`. В отличие от обычных подпрограмм у внутренней подпрограммы отсутствуют параметры, и она может оперировать только с переменными основной программы. Логически внутренние подпрограммы повторяют структуру подпрограмм в машинных кодах. К внутренней подпрограмме может обратиться только ее владелец.

Внешняя подпрограмма начинается с заголовка:

```
SUB имя-подпрограммы[(параметры)][STATIC]
```

Тело внешней подпрограммы завершается оператором `END SUB`. Для досрочного прекращения работы программы используется оператор `EXIT SUB`.

К внешней подпрограмме можно обратиться из любой программной единицы двумя способами, особой разницы между которыми нет:

```
имя_подпрограммы [фактические параметры]
```

или

```
CALL имя_подпрограммы[(фактические параметры)]
```

Оформление и вызов программных единиц в системе Turbo C

Все программные единицы в Си носят название функций и разницу в оформлении настоящих подпрограмм и настоящих функций можно обнаружить по типу возвращаемого значения. Если в качестве указания типа использовано служебное слово `void`, то перед нами типичная подпрограмма (в терминах Си — функция, не возвращающая значение). Перед заголовком объявляемой функции всегда присутствует либо стандартный тип, либо описатель типа с последующей звездочкой. Последнее означает, что функция возвращает указатель на данное соответствующего типа. В частности, допускается, что таковым указателем может быть "`void *`" В самом общем виде заголовок функции выглядит следующим образом:

```
void имя_функции([параметры])
```

или

```
тип имя_функции([параметры])
```

или

```
тип * имя_функции([параметры])
```

Тело функции, расположенное после заголовка, заключается в фигурные скобки. Если функция возвращает значение, то в ее теле должен присутствовать хотя бы один оператор `return` с указанием возвращаемого значения. Например:

```
int sign(int x)
{
/* Определение знака целого числа */
  if(x<0) return -1;
  if(x>0) return 1;
  return 0;
}
```

В отличие от Бейсика и Паскаля функции Си, не имеющие параметров, всегда сопровождаются пустыми круглыми скобками. Например — `clrscr()`.

Головная программа на Си представлена функцией `main`, которая может иметь до трех параметров, связанных с извлечением аргументов из командной строки. Чаще всего эта функция не имеет параметров и не возвращает значение:

```
void main(void)
```

К функции-подпрограмме в Си обращаются, указав ее имя со списком фактических параметров:

```
имя_функции(фактические параметры);
```

Функция, возвращающая значение, может использоваться как операнд в выражении соответствующего типа:

```
int qq;
......
qq=getch(); /*ожидание ввода кода символа с клавиатуры*/
```

Однако в большинстве примеров предыдущих глав вы могли заметить, что к функции `getch` обращаются и как к обычной подпрограмме, игнорируя возвращаемое значение. Это правило распространяется на любые функции Си. Конечно, не каждая из них в этом варианте может оказаться полезной. Например, допустимо, но нелепо встретить строку:

```
sin(0.5);
```

Системная функция в данном случае проработает, но эффекта от такого вызова никто не заметит. Разве что немного увеличится время работы программы. А в случае с функцией `getch` игнорирование результата позволяет зафиксировать момент, когда пользователь нажал какую-то клавишу.

Оформление и вызов программных единиц в системе Turbo Pascal

Как дань своему прототипу, — алгоритмическому языку АЛГОЛ, — Паскаль называет свои подпрограммы процедурами и начинает их описание со строки вида:

```
procedure имя_процедуры[(параметры)];
```

Тело процедуры на Паскале в точности повторяет структуру головной программы. В нем могут присутствовать разделы описания меток (`label`), констант (`const`), типов данных (`type`), переменных (`var`) и других процедур и функций, входящих в состав данной процедуры. Наличие вложенных процедур и функций отличает Паскаль и от Си, и от Бейсика. Собственно вычислительная часть тела процедуры начинается со служебного слова `begin` и заканчивается соответствующим `end`, после которого, в отличие от головной программы, следует не точка, а точка с запятой.

Фрагмент программы от заголовка процедуры до завершающей операторной скобки `end` принято называть блоком. Для Паскаля характерна возможность использования вложенных блоков, с каждым из которых можно связать натуральное число, соответствующее уровню вложения. Так, например, в блок `A` могут быть вложены два последовательных блока первого уровня — блоки

`B` и `C`. Если же блок `C` вложен в блок `B`, входящий в свою очередь в блок `A`, то блок `C` уже имеет уровень два по отношению к блоку `A`. В теле любого блока могут содержаться обращения к вложенным блокам своего первого уровня или к последовательным блокам такого же уровня. Блок не имеет права напрямую обратиться к своим процедурам второго или более высокого уровня вложенности.

Если процедура `A` обращается к процедуре `B`, то описание процедуры `B` должно предшествовать описанию процедуры `A`. Однако, если программа включает рекурсивную цепочку из двух или более процедур, одна из них должна объявляться (но не описываться) первой с добавкой `forward`. Соответствующее объявление носит название опережающего (`forward` — впереди бегущий футболист, нападающий):

```
procedure имя_процедуры[(параметры)]; forward;
```

Описание опережающей процедуры располагается после других процедур рекурсивной цепочки, но в ее заголовке список параметров может уже не указываться.

Для обращения к процедуре достаточно задать ее имя с последующим набором фактических параметров.

Описание функции отличается только тем, что ее заголовок начинается со служебного слова `function` и заканчивается указанием типа функции.

```
function имя_функции[(параметры)]:тип;
```

Тело функции отличается от тела подпрограммы только наличием оператора присваивания:

```
имя_функции:=выражение;
```

Досрочный выход из тела функции или тела процедуры осуществляется по оператору `exit`.

Головная программа на Паскале выступает в роли контейнера, содержащего все свои функции и процедуры. Первым оператором головной программы может быть не обязательная строка:

```
program имя_программы;
```

Собственно тело головной программы начинается с операторной скобки `begin`, расположенной вслед за последним объявлением, и завершается закрывающей операторной скобкой `end` с последующей точкой.

Оформление модулей на Паскале

В отличие от головной программы с ее первым необязательным оператором, модуль обязан начинаться со следующего заголовка:

```
Unit имя_модуля;
```

Если в качестве имени паскалевской программы может выступать любой идентификатор, в том числе и достаточно длинный, то именем модуля может быть только имя файла, допустимое в операционной системе MS-DOS. Это связано с тем, что при некоторых обстоятельствах (например, если пользователь вместо режима `Compile` выбрал режим `Make` или `Build`) система программирования Turbo Pascal вынуждена заново оттранслировать пользовательские модули. И тогда она пытается открыть дисковые файлы, имена которых совпадают с именами обновляемых модулей.

Второе отличие модуля от головной программы заключается в структуре модуля, содержащего от одной до трех частей, начинающихся служебными словами `interface`, `implementation` и `begin`. Интерфейсная часть, следующая за словом `interface`, состоит из описания типов данных, констант и переменных, а также заголовков функций и процедур, доступных любой программе, использующей данный модуль.

Вслед за служебным словом `implementation` начинается раздел описания всех процедур и функций, включенных в состав модуля. Среди них, естественно, должны присутствовать все процедуры и функции, упомянутые в интерфейсной части. Однако, кроме них, в состав модуля могут входить любые процедуры и функции внутреннего пользования, доступные только внутри данного модуля. Об их существовании внешний пользователь не обязан знать. Если после завершения исполнительного раздела (`implementation`) встречается операторная скобка `begin`, то за ней следует так называемая инициирующая часть модуля. Эта часть содержит обычные операторы, выполняемые непосредственно перед стартом головной программы. Как и программа, модуль завершается операторной скобкой `end` с последующей точкой.

Любая из описанных выше частей в модуле может быть пустой. Например, модуль, содержащий только интерфейсную часть с описанием структур данных типа календарных дат, позволит не повторять эти описания в основной программе:

```
Unit Calendar
interface
  type
    Days=(Mon,Tue,Wed,Thu,Fri,Sat,Sun);
    WorkingDays=Mon..Fri;
    Months=(Jan,Feb,Mar,Apr,May,June,July,Aug,Sept,Oct,Nov,Decem);
    Summer=June..Aug;
    Autumn=Sep..Nov;
    Spring=Mar..May;
    DayNo=1..31;
    YearNo=1900..2119;
    Date=record
      Day:DayNo;
```

```
      Month:Months;
      Year:YearNo;
  end;
implementation
end.
```

Модуль, содержащий только инициирующую часть, может произвести какие-то первоначальные действия — открыть временный файл, проверить принтер и т. п.

Параметры подпрограмм, локальные и глобальные данные

Для обеспечения максимальной независимости программных единиц во всех алгоритмических языках принято стандартное правило — данные, объявленные внутри программной единицы, являются собственностью этой программной единицы и образуют так называемый набор *локальных данных*. В QBasic и Си область действия локальных данных распространяется только на подпрограмму (функцию), в которой эти данные объявлены. Поэтому, несмотря на совпадение имен переменных, используемых в разных программных единицах, между такими переменными ничего общего нет. Это развязывает руки создателям программ, т. к. позволяет им не вникать в детали чужих программ, которые они собираются использовать. Наличие вложенных блоков в программах на Паскале немного модифицирует этот стандарт. Здесь действуют права наследования — любой внутренний блок имеет право использовать данные, объявленные во внешнем по отношению к нему блоке. Но если во внутреннем блоке объявляется переменная с таким же именем, как и во внешнем блоке, для нее выделяется новая ячейка оперативной памяти при одновременном сохранении значения переменной внешнего блока.

В связи с описанным разграничением данных возникает вопрос о механизмах взаимодействия программных единиц, одна из которых обращается к другой. Таких механизмов, в общем-то, два — передача данных через *параметры* и совместное использование *общих (глобальных)* переменных.

Передача параметров в рассматриваемых системах программирования организуется через стек, в который вызывающая программа может положить либо значение соответствующего фактического параметра, либо его адрес в оперативной памяти. Поэтому принято говорить о передаче параметров либо *по значению*, либо *по адресу*.

Если в стеке находится значение фактического параметра, то вызываемая подпрограмма может его извлечь и поместить в свою собственную локальную переменную. Эта переменная может участвовать в любых вычислениях, ее значение во время работы вызванной подпрограммы может неоднократно изменяться, но вызвавшая программа об этих изменениях ничего не узнает. По-

этому по значению можно передавать только те параметры, которые содержат входную информацию для работы вызванной процедуры или функции.

Если вместо значения фактического параметра в стек был помещен его адрес, то любые изменения значения соответствующей переменной сразу же станут общим достоянием. Поэтому по адресу обычно передают те параметры, значения которых должна сформировать вызванная подпрограмма. Кроме этого, по адресу стараются передавать массивы, т. к. поместить в стек адрес намного эффективнее, чем пересылать туда каждый элемент массива. Иногда в стек желательно поместить адрес параметра, сопроводив его дополнительным указанием, запрещающим вызванной подпрограмме изменять значение параметра.

Как определить или задать тип параметра в подпрограммах и функциях на Бейсике? Обратите внимание на заголовок подпрограммы или функции. Имя каждого формального параметра либо сопровождается специальным знаком (`%` — короткий целый, `&` — длинный целый, `!` — короткий вещественный, `#` — длинный вещественный, `$` — строковый), либо за ним следует описатель типа вида `AS type`. В качестве описателя типа параметра может быть использовано одно из служебных слов `INTEGER`, `LONG`, `SINGLE`, `DOUBLE` или `STRING`.

Если параметром является массив, то, независимо от его размерности, вслед за именем располагаются пустые скобки. Для определения минимального и максимального значения индекса по каждому измерению фактического массива, который подпрограмма получит во время работы, следует использовать функции `LBOUND` и `UBOUND`. Первым аргументом таких функций является имя формального массива (а во время выполнения вместо него будет подставлено имя фактического массива), а вторым — порядковый номер индекса для многомерных массивов. Если формальный параметр `A` представлен одномерным массивом, то обращения `LBOUND(A,1)` и `LBOUND(A)` эквивалентны.

Если при вызове функции или подпрограммы в качестве фактического параметра указано имя простой переменной или массива, то происходит передача по адресу. Поэтому, если вызванная подпрограмма изменяет значение формального параметра, эти изменения дойдут и до вызывавшей программы. Если имя фактического параметра было заключено в круглые скобки, то интерпретатор передает значение и никакие изменения формального параметра не отразятся на значении фактического аргумента. Если фактический параметр задан выражением, то в любом случае будет передано только его значение. Таким образом, QBasic допускает, что вместо одного и того же формального параметра в одном обращении может фигурировать адрес фактического параметра, а в другом — значение фактического параметра. Объясняется это тем, QBasic является интерпретатором, который в зависимости от сложившейся ситуации может изменить ход выполнения подпрограммы.

В отличие от этого в системах компилирующего типа, к которым относятся и Turbo C, и Turbo Pascal, ситуация с формальными параметрами процедур и функций определена более жестко. В качестве любого конкретного пара-

метра можно передать либо адрес соответствующего фактического аргумента, либо его значение. Для параметров, принимающих только адреса, в Си используются указатели, а в Паскале — аргументы, снабженные описателем `var`. Все остальные параметры передаются только по значению. Чтобы предотвратить изменение фактического параметра, переданного по адресу, в списке аргументов перед описанием соответствующего формального параметра должно находиться служебное слово `const`.

Кроме аппарата параметров, программные единицы могут совместно использовать значения общих (глобальных переменных).

В QBasic объявление глобальных переменных, доступных в любой внешней подпрограмме или внешней функции, сопровождается добавкой `SHARED` (дословно — "совместно используемый"):

```
DIM SHARED A(20) AS INTEGER, D AS SINGLE
COMMON SHARED A(20) AS INTEGER, D AS SINGLE
```

Последнее объявление означает, что общие переменные (массив `A` и одиночная переменная `D`) будут доступны не только всем подпрограммам и функциям данного программного файла, но и другим Бейсик-программам, загружаемым в память по оператору `CHAIN`.

В программных файлах Си переменные, объявленные за пределами всех функций, считаются глобальными по отношению к функциям этого файла и могут использоваться в теле любой функции без каких-либо дополнительных указаний. Но если в теле функции объявлена переменная с таким же именем, то зона ее влияния локализована в данной функции и глобальная переменная с аналогичным именем здесь уже не действует. Если же задача собирается из нескольких программных файлов, общие для всех функций переменные целесообразно выделить в отдельный программный файл. В функциях, где предполагается их использование, такие переменные должны объявляться с описателем `extern` (внешний):

```
extern int a[20];
extern float d;
```

В программе на Паскале все переменные головной программы являются глобальными. К таковым же относятся и все переменные, объявленные в модулях, подключаемых к программе с помощью директивы `Uses`. А дальше действует описанный выше стандарт — переменные любого блока могут выступать в качестве глобальных по отношению ко всем вложенным блокам.

Дерево решений

В начале XX в. многие американские фермеры увлекались игрой в "15", за решение которой предлагался весьма солидный приз. На квадратном поле размером 4×4 размещались 15 фишек с числовыми номерами 1, 2, ..., 15.

Наличие одного свободного поля позволяло передвигать фишки по горизонтали и вертикали с целью их сортировки по возрастанию номеров. Позиция, за которую предлагалось вознаграждение в размере $100 000, содержала единственное нарушение порядка — фишки с номерами 14 и 15 были переставлены местами. Позднее было доказано, что из этой позиции невозможно перейти к полностью упорядоченной последовательности фишек. Так что призовой комитет ничем не рисковал.

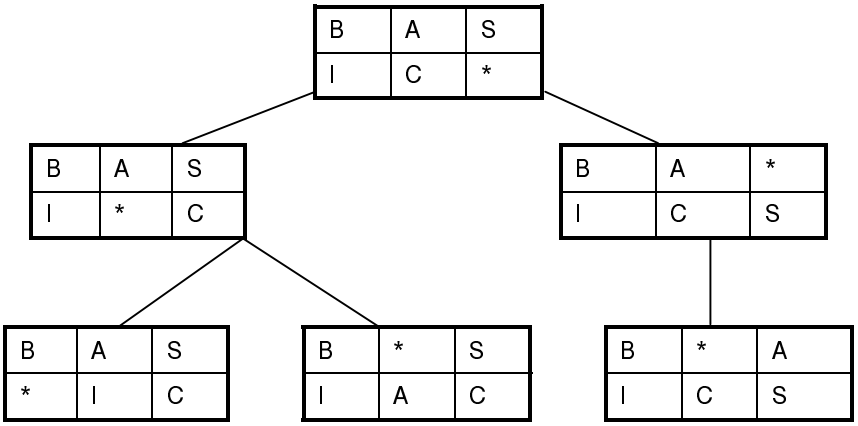
Более того, оказалось, что все позиции фишек делятся на две категории. Если суммарное количество нарушений порядка в исходной позиции было четным, то полная сортировка фишек возможна. И тогда единственный смысл игры (исключая простое времяпрепровождение) заключался в достижении полностью упорядоченной расстановки за минимальное количество ходов. Для определенности под ходом мы будем подразумевать перемещение одной фишки на свободное место (в реальной игре прибегают к одновременному сдвигу двух или трех фишек, расположенных в одной строке или одном столбце).

С целью сокращения количества переборов мы продемонстрируем аналогичную игру на поле размером 2×3, где расположены фишки с буквами A, B, C, I, S. Цель игры — за минимальное число ходов перевести заданную исходную позицию в позицию BASIC:

B	A	S
I	C	

Идея решения этой задачи заключается в построении полного дерева решений, в верхушке которого располагается целевая позиция BASIC. Каждый последующий уровень дерева образуют позиции, полученные из предыдущей вершины за один ход. С целью удобства ввода начальной позиции и последующего отображения ходов пустая клетка у нас будет представлена символом "*".

В нашей игре начало дерева может выглядеть следующим образом:



Естественно, что в очередной вершине дерева должна находиться позиция, не повторяющая предыдущие позиции. Так как общее количество перестановок из шести элементов равно 6!=720, то общее количество вершин в нашем дереве не будет превосходить эту границу. Более того, задача симметрична и из целевой позиции BASIC* можно будет породить ровно 359 новых вершин.

Для хранения дерева допустимых позиций используем два массива — массив строк `tree`, обеспечивающий хранение 360 шестисимвольных значений позиций, и числовой массив `ind[360]`, в элементах которого будут фиксироваться индексы породивших строк. В нашем примере начальное заполнение массивов `tree` и `ind` может выглядеть следующим образом (табл. 6.1):

Таблица 6.1. *Начальное заполнение массивов* `tree` *и* `ind`

Индекс строки	Значение строки	Ссылка на породившую строку
0	BASIC*	–1
1	BAIC*S	0
2	BASI*C	0
3	B*AICS	1
4	B*SIAC	2
5	BAS*IC	2
6	BCAI*S	3
...	.....	...

После того как построение полного дерева приводимых решений будет завершено, поиск оптимальной цепочки перестановок сводится к довольно простой процедуре. Сначала надо проверить, содержится ли исходная позиция среди вершин дерева. Если ее там нет, то заданная позиция не может быть сведена к позиции BASIC*. Если заданная позиция обнаружена в `k`-й вершине, то для восстановления цепочки переходов используем массив индексов породивших строк:

- `k1 = ind[k]` — индекс предыдущей вершины, определяющей 1-й ход;
- `k2 = ind[k1]` — индекс вершины, определяющей 2-й ход;
- `k3 = ind[k2]` — индекс вершины, определяющей 3-й ход;
- .....

И это построение надо продолжать до тех пор, пока мы не встретим целевую вершину, у которой нет продолжения (ее индекс равен –1).

Совет 1 (общий)

Для построения дерева решений можно воспользоваться самым простым перебором: если пустая клетка находится на пересечении `i`-й строки и `j`-го столбца, то теоретически с ней могут поменяться местами символы с индексами (`i-1`, `j`), (`i+1`, `j`), (`i`, `j-1`), (`i`, `j+1`). На самом деле допустимых перемещений не четыре, а три или два, в зависимости от положения пустой клетки. Следовательно, в процедуре `change` придется позаботиться о проверке на принадлежность границам матрицы каждого кандидата на перемещение. И включать в дерево решений мы будем только те позиции, которые еще не встречались.

Совет 2 (общий)

Для удобства манипуляций со значениями вершин их целесообразно рассматривать и как одномерные строки, и как двумерные массивы 2×3, каждый элемент которых представлен соответствующим символом. Первое удобно для организации сравнения позиций, а второе — для перестановки фишек. Связь между одномерным индексом `k` и парой двумерных индексов (`i`,`j`) осуществляется по формулам:

```
k = i * 3 + j.
i = k div 3.
j = k mod 3.
```

Следует не забывать, что приведенные формулы справедливы, если индексы в массивах отсчитываются от 0, например так, как это делается в Си. Однако в строках Бейсика и Паскаля индексы символов отсчитываются от 1. Поэтому в соответствующих фрагментах программ введена соответствующая коррекция значения одномерного индекса `k`.

Совет 3 (общий)

Для более четкой организации структуры программы выделим следующие программные единицы:

- `level` — процедура, порождающая новые вершины в дереве решений из позиции с индексом `from`;
- `change(i1, j1)` — процедура, осуществляющая обмен символа с указанными индексами, и символа "`*`", расположенного в позиции (`i`, `j`). Если новая позиция допустима и она еще не содержится в дереве решения, то процедура `change` присоединяет новую позицию к дереву решений и увеличивает значение счетчика вершин `nmax`;
- `poisk` — процедура, осуществляющая поиск исходной позиции `pos0` в дереве решений. Если исходная позиция найдена среди приводимых вершин, то процедура `poisk` осуществляет раскрутку цепочки ходов до главной вершины дерева;
- `print_tab(s)` — процедура отображения очередного хода, представленного строкой `s`, в форме таблички размером 2×3. Координаты левого верхнего угла таблички на экране представлены глобальными переменными (`x0`, `y0`).

Программа 6_01.bas

```
DECLARE SUB change (i1 AS INTEGER, j1 AS INTEGER)
DECLARE SUB printTAB (s AS STRING)
DECLARE SUB level ()
DECLARE SUB poisk ()
REM Дерево решений BASIC
DEFINT A-Z
DIM SHARED x0 AS INTEGER, y0 AS INTEGER, nmax AS INTEGER, k AS INTEGER
DIM SHARED from AS INTEGER
DIM SHARED tree(360) AS STRING * 6, ind(360) AS INTEGER, pos0 AS STRING *
6
nmax = 1: x0 = 1: y0 = 2
tree(0) = "BASIC*"
ind(0) = -1
CLS
INPUT "Введите строку с исходной позицией - ", pos0
FOR from = 0 TO nmax - 1:
  level
NEXT from
poisk
END

SUB change (i1 AS INTEGER, j1 AS INTEGER)
  DIM n AS INTEGER, k1 AS INTEGER, tmp AS STRING, str2 AS STRING
  IF (i1 < 0) OR (i1 > 1) OR (j1 < 0) OR (j1 > 2) THEN EXIT SUB
  k1 = i1 * 3 + j1 + 1
  str2 = tree(from)
  tmp = MID$(str2, k, 1)
  MID$(str2, k, 1) = MID$(str2, k1, 1)
  MID$(str2, k1, 1) = tmp
  FOR n = 0 TO nmax - 1
      IF str2 = tree(n) THEN EXIT SUB
  NEXT n
  ind(nmax) = from
  tree(nmax) = str2
  nmax = nmax + 1
END SUB
```

```
SUB level
  FOR k = 1 TO 6
    IF MID$(tree(from), k, 1) = "*" THEN EXIT FOR
  NEXT k
  i = (k - 1) \ 3  '  номер строки
  j = (k - 1) MOD 3'  номер столбца
  CALL change(i - 1, j)
  CALL change(i + 1, j)
  CALL change(i, j - 1)
  CALL change(i, j + 1)
END SUB

SUB poisk
  FOR q = 0 TO nmax - 1
    IF pos0 = tree(q) THEN GOTO m
  NEXT q
  PRINT "Эта позиция не сводится к требуемой"
  EXIT SUB
m:
  CALL printTAB(tree(q))
  q = ind(q)
  IF q >= 0 THEN GOTO m
END SUB

SUB printTAB (s AS STRING)
  LOCATE y0, x0:  PRINT "+-+-+-+"
  LOCATE y0 + 1, x0: PRINT "|"; MID$(s, 1, 1);
                  PRINT "|"; MID$(s, 2, 1);
                  PRINT "|"; MID$(s, 3, 1); "|"
  LOCATE y0 + 2, x0: PRINT "+-+-+-+"
  LOCATE y0 + 3, x0: PRINT "|"; MID$(s, 4, 1);
                  PRINT "|"; MID$(s, 5, 1);
                  PRINT "|"; MID$(s, 6, 1); "|"
  LOCATE y0 + 4, x0: PRINT "+-+-+-+"
  x0 = x0 + 10
  IF x0 = 81 THEN y0 = y0 + 5: x0 = 1
END SUB
```

Программа 6_01.c

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void level();
void change(int i1, int j12);
void poisk();
void print_tab(char* s);
char tree[360][7];
char pos0[7];
int ind[360];
int i,j,k,nmax=1;
int x0=1,y0=2;
int from;
  void main()
{
  strcpy(&tree[0][0],"BASIC*");
  ind[0]=-1;
  clrscr();
  printf("Введите строку с исходной позицией - ");
  scanf("%s",pos0);
  for(from=0; from<nmax; from++)
    level();
  poisk();
  getch();
}
//-----------------------------------------
void level()
{
  for(k=0; k<6; k++)
    if(tree[from][k]=='*')break;
  i=k/3;  //номер строки
  j=k%3;  //номер столбца
  change(i-1,j);
  change(i+1,j);
  change(i,j-1);
  change(i,j+1);
}
```

```
//-----------------------------------------
void change(int i1, int j1)
{ char tmp,str2[7];
  int n,k1;
  if(i1<0 || i1>1 || j1<0 || j1>2)
     return;
  k1=i1*3+j1;
  strcpy(str2,(char*)&tree[from][0]);
  tmp=str2[k];
  str2[k]=str2[k1];
  str2[k1]=tmp;
  for(n=0; n<nmax; n++)
      if(strcmp(str2,(char*)&tree[n][0])==0) return;
  ind[nmax]=from;
  strcpy((char*)&tree[n][0],str2);
  nmax++;
}
//-----------------------------------------
void poisk()
{ char *p;
  p=&tree[0][0];
  for(int q=0; q<nmax; q++)
      if(strcmp(pos0,p+q*7)==0) goto m;
  printf("\nЭта позиция не сводится к требуемой");
  return;
m:
  print_tab(p+q*7);
  q=ind[q];
  if(q>=0) goto m;
  return;
}
//-------------------------------------------
void print_tab(char* s)
{ char top[]=   "+-+-+-+";
  char mid[]=   "+-+-+-+";
  char bottom[]="+-+-+-+";
  gotoxy(x0,y0);
  printf("%s",top);
  gotoxy(x0,y0+1);
```

```
  printf("¦%c¦%c¦%c¦",s[0],s[1],s[2]);
  gotoxy(x0,y0+2);
  printf("%s",mid);
  gotoxy(x0,y0+3);
  printf("¦%c¦%c¦%c¦",s[3],s[4],s[5]);
  gotoxy(x0,y0+4);
  printf("%s",bottom);
  x0=x0+10;
  if(x0==81){y0=y0+5;x0=1;}
}
```

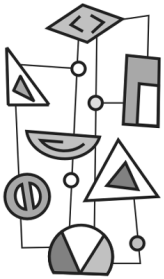
Программа 6_01.pas

```
program basic;
uses Crt;
var
  tree:array [0..359] of string[6];
  pos0:string[6];
  ind:array [0..359] of integer;
  x0,y0,i,j,k,nmax,from:integer;

procedure print_tab(s:string);
begin
  gotoxy(x0,y0);
  write('+-+-+-+');
  gotoxy(x0,y0+1);
  write('|',s[1],'|',s[2],'|',s[3],'|');
  gotoxy(x0,y0+2);
  write('|-+-+-|');
  gotoxy(x0,y0+3);
  write('|',s[4],'|',s[5],'|',s[6],'|');
  gotoxy(x0,y0+4);
  write('+-+-+-+');
  x0:=x0+10;
  if(x0=81)then begin y0:=y0+5;x0:=1; end;
end;

procedure poisk;
var
  q:integer;
```

```
label m;
begin
  for q:=0 to nmax-1 do
    if pos0=tree[q] then goto m;
  write('Эта позиция не сводится к требуемой');
  readln;
  exit;
m:
  print_tab(tree[q]);
  q:=ind[q];
  if q>=0 then goto m;
end;

procedure change(i1,j1:integer);
var
  tmp:char;
  str2:string[6];
  n,k1:integer;
begin
  if (i1<0)or(i1>1)or(j1<0)or(j1>2) then exit;
  k1:=i1*3+j1+1;
  str2:=tree[from];
  tmp:=str2[k];
  str2[k]:=str2[k1];
  str2[k1]:=tmp;
  for n:=0 to nmax-1 do
    if str2=tree[n] then exit;
  ind[nmax]:=from;
  tree[nmax]:=str2;
  inc(nmax);
end;

procedure level;
begin
  for k:=1 to 6 do
    if tree[from][k]='*' then break;
  i:=(k-1) div 3;  {номер строки}
  j:=(k-1) mod 3;  {номер столбца}
```

```
  change(i-1,j);
  change(i+1,j);
  change(i,j-1);
  change(i,j+1);
end;

begin
  nmax:=1;
  x0:=1; y0:=2;
  tree[0]:='BASIC*';
  ind[0]:=-1;
  clrscr;
  write('Введите строку с исходной позицией');
  read(pos0);
  for from:=0 to 359 do level;
  poisk;
  readln;
end.
```

Глава 7



Работа с дисковыми файлами

Наиболее распространенным видом файлов являются именованные области внешней (дисковой) памяти, с которыми программы могут обмениваться информацией. Необходимость в таких обменах возникает, когда объем оперативной памяти недостаточен для хранения нужной информации. В другой ситуации, например, программа может воспользоваться данными, полученными ранее другой программой и предусмотрительно записанными на диск. К числу абонентов, участвующих в обмене данными, относятся и файлы-устройства: дисплей, принтер, графопостроитель, сканер, клавиатура, каналы связи и т. п. Данный раздел посвящен работе с дисковыми файлами, хотя технология обслуживания файлов-устройств принципиально мало отличается.

Оценивая ключевые аспекты процесса обмена данными, можно сказать, что работа с файлами, в основном, ограничивается тремя-четырьмя операциями:

- выделение ресурсов и приведение файла в состояние готовности к обмену (именно это скрывается за термином "открыть файл");
- чтение (ввод из файла) или запись (вывод в файл) очередной порции данных;
- возврат выделенных ресурсов и завершение неоконченных операций (этому соответствует термин "закрыть файл").

Несмотря на кажущуюся простоту процесса обмена данными, файловые операции достаточно сложны в освоении. Ну как, например, не спасовать перед системной библиотекой Си, которая насчитывает более 85 функций, обслуживающих дисковые файлы, и свыше 60 констант, задающих режимы работы файловых процедур. В QBasic и Паскале количество соответствующих процедур существенно меньше, однако подводные камни встречаются и там.

При работе с файлами приходится учитывать многочисленные форматы представления данных того или иного типа на разных носителях информации. Так, например, цепочка из `k` символов, представляющая текстовую строку, может храниться в одном из следующих форматов:

- "$S_1S_2S_3...S_k$" (переменное число символов, заключенных в одинарные или двойные кавычки);
- `kS1S2...Sk` (`k` — однобайтовый или двухбайтовый указатель числа символов, предшествующий тексту);

- S1S2...Sk\0 (\0 — однобайтовый признак конца строки, расположенный вслед за последним символом текста);
- S1S2...Sk 0D 0A (двухбайтовый признак конца строки, 0D — "возврат каретки", 0A — "перевод строки").

Числовая информация может быть записана в дисковый файл либо в машинном формате (а в Си и Паскале количество разных типов числовых данных достигает десятка), либо с предварительным преобразованием из машинного представления в символьное.

Кроме числовых и текстовых данных в файлах может храниться информация и другого происхождения. Например, графические изображения, которые в процессах обмена данными выступают как двоичные коды, условно разделенные на байты. Естественно, что на содержимое этих байтов нельзя реагировать так же, как на некоторые управляющие коды типа "Возврат каретки", "Перевод строки", "Признак конца файла", влияющие на передачу числовой и текстовой информации.

Кроме того, существует несколько способов доступа к файловым данным, из которых на практике чаще всего используют два — последовательный и произвольный. Последний иногда называют прямым (DIRECT ACCESS) или случайным (RANDOM ACCESS). Последовательный доступ при записи на диск характерен тем, что очередная записываемая порция пристраивается в хвост к предыдущей. Размеры смежных порций при этом могут оказаться разными по длине. При чтении такой набор данных начинает извлекаться с самой первой порции и очередность считываемых данных повторяет их последовательность во время записи.

Файлы с произвольным доступом состоят из данных, разбитых на порции фиксированной длины. При этом имеется возможность записывать или читать данные в произвольном порядке, указывая дополнительно номер нужной порции.

Наконец, необходимо учитывать и способы разделения отдельных числовых или символьных значений в дисковых наборах данных. В некоторых ситуациях роль таких разграничителей могут выполнять кавычки, запятые, пробелы и различные управляющие байты ("табуляторный пропуск", "возврат каретки", "перевод строки", "признак конца файла"). В других ситуациях для каждого данного может быть выделено поле фиксированной длины.

Программы, включенные в настоящий раздел, имеют целью научиться записывать данные на диск, читать их оттуда и точно представлять форматы хранения данных в файлах разного типа.

Основные типы файлов в системе QBasic

Система QBasic поддерживает работу с файлами трех типов — строковыми, записеориентированными и двоичными. Приведенные термины не являются

общеупотребительными, однако они достаточно точно отражают формат хранения данных в дисковых файлах.

Строковые файлы в системе QBasic

В строковом файле условной порцией хранения данных является строка переменной длины, завершаемая двумя управляющими байтами с кодами `0D` ("`Возврат каретки`") и `0A` ("`Перевод строки`"). Строка файла может быть либо пустой, либо содержать одно или несколько числовых и символьных значений. Символьное значение либо завершается запятой, либо заключается в двойные кавычки. Числовые значения представлены в символьном формате и завершаются либо запятой, либо пробелом. Последнее значение в файловой строке завершается парой управляющих кодов `0D0A`.

Строковый файл последовательного доступа для вывода открывается с помощью следующего оператора:

```
OPEN имя_файла FOR OUTPUT AS #k
```

Если мы собираемся добавлять информацию в уже существующий строковый файл, то его открывают с указанием `FOR APPEND`. Файл, из которого информация должна считываться, открывается с указанием `FOR INPUT`.

Открываемому файлу программист присваивает числовой номер `k` из диапазона [1,255], который впоследствии заменяет имя файла в операторах ввода (`INPUT`) или вывода (`PRINT`, `WRITE`). Символ `#` может опускаться.

Закрывается строковый файл оператором `CLOSE #k`.

Вывод в строковый файл, как правило, осуществляется по оператору `WRITE #k`. При таком выводе каждое текстовое значение автоматически заключается в кавычки и все данные разделяются запятыми. Чтение данных из строкового файла производится с помощью оператора `INPUT #k` и ничем принципиально не отличается от ввода данных из строки, набираемой пользователем на клавиатуре.

В принципе, в строковый файл можно произвести запись и по оператору `PRINT #k`. Однако при этом выводимый текст в кавычки не заключается и разделительные запятые между отдельными значениями не вставляются. Если в строке файла оказывается несколько текстовых и числовых значений, то их потом будет трудно извлечь. Коллизии подобного рода демонстрирует программа 7_01.bas.

Программа 7_01.bas

```
CLS : DEFINT A-Z: A$="Строка"
OPEN "bas_txt" FOR OUTPUT AS #1
FOR J=1 TO 10
```

```
  PRINT #1,A$;J,J*2
  PRINT A$;J,J*2
NEXT J
CLOSE #1
PRINT
OPEN "bas_txt" FOR INPUT AS #2
FOR J=1 TO 10
  INPUT #2,B$,K1,K2
  PRINT B$;K1,K2
NEXT J
CLOSE #2
END
```

Если ограничиться только первой половиной программы, которая записывает в дисковый файл `bas_txt` 10 строк и попутно выдает на экран содержимое этих строк, то кажется, что все в порядке. Содержимое файла `bas_txt`, которое можно увидеть, нажав клавишу F3, в точности повторяет ту информацию, которая отражена на экране:

```
Строка 1             2
Строка 2             4
Строка 3             6
Строка 4             8
Строка 5             10
Строка 6             12
Строка 7             14
Строка 8             16
Строка 9             18
Строка 10            20
```

Один пробел после слова `Строка` образовался потому, что в теле оператора `PRINT` разделитель "`точка с запятой`" после текста не изменяет текущую позицию выводной строки, но записываемые числа положительны и вместо знака "+" мы видим пробел. Вторые числа в каждой строке начинаются с 15-й позиции (там тоже находится пробел вместо знака "+"). Этот переход в начало очередной зоны вывода вызван разделителем "`запятая`" в списке оператора `PRINT`. Если переключиться в режим просмотра шестнадцатеричной информации (F3 → F4), то можно заметить дополнительные детали: однозначные числа представлены однобайтовыми кодами ASCII (1 → 31, 2 → → 32, ...), а двузначные — двухбайтовыми (10 → 3130, 12 → 3132, ...). Кроме того, каждая строка завершается двухбайтовым признаком конца строки — `0D0A`. Это означает, что при выводе числовые данные были переведены в

символьный формат, а каждая порция вывода была дополнена управляющими кодами `Возврат каретки` + `Перевод строки`.

Действие второй половины программы вызывает недоумение. Во-первых, на экране появляется сообщение `Input paste end of file`, которое свидетельствует о попытке чтения после исчерпания данных в файле. Во-вторых, переключившись на экран пользователя, вместо ожидавшихся строк с одним словом и двумя числами в каждой мы видим совсем не то:

```
Строка 1              2 0       2
4 0             3
6 0             4
.............................
```

Тем не менее все объясняется достаточно просто. При считывании самой первой порции в переменную `B$` заносятся первые 16 символов из строки дискового файла, завершаемые признаком конца строки (BASIC-система не настолько умна, чтобы остановиться после извлечения первых шести символов). При этом управляющие байты `0D0A` в переменную `B$` не записываются, но пропускаются. Для формирования значения числовой переменной `K1` данные начинают извлекаться уже из второй строки дискового файла. Но там до первого разделителя числовых значений (пробела) расположены только нечисловые символы (слово `Строка`), которые игнорируются, и в переменную `K1` ничего не поступает (`K1 = 0`). В переменную `K2` попадает первое числовое значение из второй строки (`K2 = 2`). При следующем повторении цикла в переменную `B$` заносится остаток второй файловой строки, но лидирующие пробелы при этом игнорируются (`B$ = "4"`). Таким образом, к десятому повторению цикла образуется недостача одного числового значения.

Однако из приведенного примера вы должны сделать правильный вывод — нельзя просто так, без всяких разделителей, смешивать в одной строке символьные и числовые данные.

На самом деле, описанные выше проблемы снимаются, если в дисковый файл числовые и символьные данные выводятся по оператору `WRITE #k`. Проделайте эксперимент с программой 7_02.bas и поинтересуйтесь содержимым дискового файла `bas_txt`:

Программа 7_02.bas

```
CLS : A$ = "Строка"
OPEN "bas_txt" FOR OUTPUT AS #1
FOR J%=1 TO 10
  WRITE #1,A$,J%,SQR(J%)
  PRINT A$,J%,SQR(J%)
NEXT J%
```

```
CLOSE #1
PRINT
OPEN "bas_txt" FOR INPUT AS #2
FOR J%=1 TO 10
  INPUT #2,B$,I%,R
  PRINT B$,I%,R
NEXT J%
CLOSE #2
END
```

Записеориентированные файлы в системе QBasic

Единицей обмена информацией для записеориентированных файлов является "запись", представляющая собой заранее описанную последовательность полей жесткой структуры. На каждом из таких полей располагается одно значение соответствующего типа в машинном формате. Последнее позволяет экономить место, необходимое для хранения данных, и исключает затраты на прямое или обратное преобразование данных между машинным представлением и символьным форматом в процессе обмена. Фиксированный размер каждого поля делает ненужным использование различных разделителей.

Описание структуры записи располагается между служебными словами `TYPE` — `END TYPE`. В программе 7_03.bas через `qq` обозначено наименование шаблона записи, состоящей из трех полей с именами `a` (6-байтовое символьное поле), `n` (2-байтовое поле для хранения короткого числа) и `r` (4-байтовое поле для хранения короткого вещественного числа). С помощью оператора `DIM` объявлена структурированная переменная `b` типа `qq`, содержащая три поля с именами `b.a`, `b.n` и `b.r`. Каждому из этих полей можно присвоить соответствующее значение и вывести запись `b` в дисковый файл:

```
PUT #1,,b              'Вывод значения b в текущую запись
PUT #1,5,b             'Вывод значения b в запись с номером 5
```

Отсчет записей в файле ведется от 1. Поскольку имеется возможность вывести значение структуры в любое место записеориентированного файла, то его инициализация производится следующим образом:

```
OPEN "bas_rec" FOR RANDOM AS #1 LEN=12
```

Последнее указание (`LEN=12`) задает длину записи в байтах, и естественно, что она должна быть равна суммарной длине всех полей структур, участвующих в обмене.

Программа 7_03.bas формирует в цикле значения полей записи `b` и выводит их на диск последовательно, а затем в цикле считывает эти записи в обратном порядке, демонстрируя тем самым произвольный доступ к записям.

Программа 7_03.bas

```
CLS
TYPE qq
  a AS STRING *6
  n AS INTEGER
  r AS SINGLE
END TYPE
DIM b AS qq
b.a="Строка"
OPEN "bas_rec" FOR RANDOM AS #1 LEN=12
FOR J%=1 TO 10
  b.n=J% : b.r=SQR(J%)
  PUT #1,,b
  PRINT b.a,b.n,b.r
NEXT J%
CLOSE #1
OPEN "bas_rec" FOR RANDOM AS #1
FOR J%=10 TO 1 STEP -1
  GET #1,J%,b
  PRINT b.a,b.n,b.r
NEXT J%
CLOSE #1
END
```

Двоичные файлы в системе QBasic

В двоичных файлах может храниться информация любого происхождения и рассматривается она только как последовательность байтов, пронумерованная от 1. Открываются двоичные файлы следующим образом:

```
OPEN имя_файла FOR BINARY AS #k
```

Доступ к данным в двоичном файле производится с помощью операторов `PUT` (вывод в файл) и `GET` (чтение из файла). При этом второй параметр в этих операторах обозначает номер байта дискового файла, с которого начинается обмен. Количество байтов, участвующих в обмене, определяется длиной третьего аргумента. С двоичным файлом можно работать, используя как последовательный (второй аргумент в операторах `GET`/`PUT` опущен), так и прямой (произвольный) доступ.

Программа 7_04.bas формирует в цикле и последовательно записывает в двоичный файл текст "`Строка`", целочисленное значение счетчика цикла `J%`

и вещественное значение квадратного корня из `J%`, а затем в цикле считывает эти данные в обратном порядке.

Программа 7_04.bas

```
CLS : A$="Строка"
OPEN "bas_bin" FOR BINARY AS #1
FOR J%=1 TO 10
  B=SQR(J%)
  PUT #1,,A$: PUT #1,,J% : PUT #1,,B
  PRINT A$, J%, B
NEXT J%
CLOSE #1
PRINT
OPEN "bas_bin" FOR BINARY AS #1
FOR J%=10 TO 1 STEP -1
  GET #1,(J%-1)*12+1,A$
  GET #1,(J%-1)*12+7,K%
  GET #1,(J%-1)*12+9,B
  PRINT A$,K%,B
NEXT J%
CLOSE #1
END
```

Основные типы файлов в Паскале

Подобно BASIC, в Паскале поддерживаются такие же три типа файлов — текстовые (строковые), типизированные (записеориентированные) и нетипизированные (двоичные).

Текстовые (строковые) файлы в Паскале

Текстовые файлы в Паскале относятся к дисковым файлам, каждая порция данных в которых представлена строкой переменной длины, содержащей не более 255 символов и завершающейся управляющими кодами `0D0A`. В отличие от QBasic, отдельные значения в файловой строке здесь не заключаются в кавычки и не разделяются запятыми. Единственным разделителем нескольких значений в пределах строки выступают пробелы, предусмотренные программистом при формировании содержимого текстового файла.

Для инициализации текстового файла необходимо объявить переменную соответствующего типа (`f1:text;`), связать ее с именем дискового файла

(`assign(f1,'pas_txt');`) и открыть для записи (`rewrite(f1);`), дозаписи (`append(f1);`) или для чтения (`reset(f1);`). Закрывается текстовый файл процедурой `close(f1);`.

Наиболее естественный программный способ создания текстового файла заключается в последовательной записи в него тем или иным способом сформированного значения какой-либо переменной типа `string` по оператору `writeln`. На самом деле, текущую строку текстового файла можно формировать в несколько приемов, записывая туда данные последовательностью операторов `write`, завершая их оператором `writeln`, который собственно и заносит признак конца строки.

Проще всего из текстового файла читать строку целиком по оператору `readln` в переменную типа `string`. Если в одной файловой строке находится несколько числовых значений, разделенных пробелами, то особых проблем при их извлечении не возникает. Требуется только соответствие количества и типов переменных считываемым значениям. Если из текущей строки текстового файла по оператору `readln` считывается меньшее количество значений, то следующий оператор `readln` будет извлекать данные с начала следующей строки, т. е. хвост предыдущей строки будет потерян. Если количество считываемых данных превышает длину текущей строки файла, то недостающие данные извлекаются из следующей строки. Однако, если числовая информация в текстовом файле перемежается со строковой, то вы можете столкнуться с описанным выше нежелательным поведением программы.

Программа 7_01.pas демонстрирует ошибку, напоминающую сбои в работе программы 7_01.bas. Отсутствие явных разделителей между символьной и числовой информацией приводит к непредусмотренному сдвигу данных (в переменную `b` считывается первая файловая строка целиком) и попытке чтения числового значения `k1` из начала второй строки файла. Но Паскаль в отличие от Basic жестко контролирует соответствие типов и выдает сообщение `Invalid numeric format` (Неправильный числовой формат) на первом же операторе `readln`.

Программа 7_01.pas

```
program bad_file;
uses Crt;
var
  j,k1,k2:integer;
  f:text;
  b:string;
begin
  clrscr;
  assign(f,'pas_txt');
```

```
  rewrite(f);
  for j:=1 to 10 do
  begin
    writeln('Строка',j:4,j*2:4);
    writeln(f,'Строка',j:4,j*2:4);
    {или write(f,'Строка'); write(f,j:4); writeln(j*2:4);}
  end;
  close(f);
  writeln;
  reset(f);
  for j:=1 to 10 do
  begin
    readln(f,b,k1,k2);
    writeln(b,k1:4,k2:4);
  end;
  close(f);
  readln;
end.
```

В программе 7_02.pas ошибка исправлена за счет изменения порядка числовых и символьных данных, но два текстовых значения в пределах одной строки ничем разделить нельзя. Заключение текста в одинарные кавычки ситуацию тоже не спасает.

Программа 7_02.pas

```
program txt_file;
uses Crt;
var
  j,k1,k2:integer;
  f:text;
  b:string;
begin
  clrscr;
  assign(f,'pas_txt');
  rewrite(f);
  for j:=1 to 10 do
  begin
    writeln(f,j:4,j*2:4,'Строка':8);
```

```
    writeln(j:4,j*2:4,'Строка':8);
  end;
  close(f);
  writeln;
  reset(f);
  for j:=1 to 10 do
  begin
    readln(f,k1,k2,b);
    writeln(k1:4,k2:4,b:8);
  end;
  close(f);
  readln;
end.
```

Типизированные (записеориентированные) файлы в Паскале

Шаблон записи объявляется в разделе описания типов:

```
type
  qq=record
    a:string[6];
    n:integer;
    r:real;
  end;
```

Для работы с типизированным файлом вводится файловая переменная (`f1:file of qq;`), которая связывается с дисковым файлом (`assign(f1,'pas_rec');`). Файл для ввода открывается оператором `reset(f1)`, а для вывода — оператором `rewrite(f1);`. Обмен с записеориентированным файлом производится операторами `read/write`, в списках которых можно указывать только переменные типа "`запись`". Записи в типизированном файле нумеруются от 0 и имеется возможность прямого доступа к любой записи. Для этого перед операцией обмена указатель файла перемещается на начало нужной записи:

```
seek(f1,номер_записи);
```

Программа 7_03.pas формирует поля записи `b` (строка из 6-ти символов, короткое целое число, вещественное число) и в цикле последовательно переписывает их на диск. Затем этот же файл открывается для ввода и его содержимое считывается в обратном порядке.

Программа 7_03.pas

```
program rec_file;
uses Crt;
type
  qq=record
    a:string[6];
    n:integer;
    r:real;
  end;
var
  f1:file of qq;
  j,k1:integer;
  rec:qq;
  d:real;
begin
  clrscr;
  assign(f1,'pas_rec');
  rewrite(f1);
  rec.a:='Строка';
  for j:=1 to 10 do
  begin
    rec.n:=j;
    rec.r:=sqrt(j);
    write(f1,rec);
    writeln(rec.a,rec.n:4,rec.r:10:4);
  end;
  close(f1);
  writeln;
  reset(f1);
  for j:=9 downto 0 do
  begin
    seek(f1,j);
    read(f1,rec);
    writeln(rec.a,rec.n:4,rec.r:10:4);
  end;
  readln;
end.
```

Нетипизированные (двоичные) файлы в Паскале

Для работы с нетипизированным (двоичным) файлом необходимо объявить файловую переменную (`f1:file;`), связать ее с именем дискового файла и открыть его для записи (`rewrite(f1,n);`) или для чтения (`reset(f1,n);`). Параметр `n` здесь является необязательным, по умолчанию его значение равно 128. Задает он размер порции данных, участвующих в обмене, в байтах. Для обмена с двоичными файлами используются специальные процедуры `blockread` и `blockwrite`:

```
blockwrite(f1,buf,n_rec,v1);
blockread(f1,buf,n_rec,v1);
```

Здесь `buf` — массив (обычно типа `byte`), в котором находятся данные, либо подготовленные для записи на диск, либо считанные с диска. Параметр `n_rec` задает количество порций, участвующих в обмене. В переменную `v1` заносится фактическое количество записанных или считанных порций. Несовпадение между `n_rec` и значением `v1` при чтении обычно связано с некратностью длины файла и объемом считываемых данных. При выводе такое событие обычно возникает при исчерпании дисковой памяти. Чтобы обеспечить доступ к любому фрагменту данных в двоичном файле, необходимо воспользоваться процедурой перемещения указателя файла в нужную позицию:

```
seek(f1,n_rec);
```

В двоичном файле "`записью`" считается порция, размер которой был установлен в момент открытия файла, и нумеруются эти записи от 0.

Программа 7_04.pas выполняет в цикле вывод 10-ти порций данных, заготавливаемых в массиве `buf` (размер каждой порции — 15 байт). Обратите внимание на то, каким образом в этом массиве выделяются участки памяти для хранения разнотипных данных. Нетипизированный указатель `p` допускает присваивание адреса любого типа, а его значение может быть присвоено любому типизированному указателю. Для текста "`Строка`", содержащего шесть символов, в массиве `buf` отводится семь байт с учетом указателя длины, предшествующего строковому значению.

Программа 7_04.pas

```
program bin_file;
uses Crt;
var
  buf:array [1..15] of byte;
  f1:file;
```

```
  p:pointer;
  ps:^string;
  pk:^integer;
  pr:^real;
  j:integer;
begin
  clrscr;
  p:=@buf[1];   ps:=p;
  p:=@buf[8];   pk:=p;
  p:=@buf[10];  pr:=p;
  assign(f1,'pas_bin');
  rewrite(f1,15);
  ps^:='Строка';
  for j:=1 to 10 do
  begin
    pk^:=j;
    pr^:=sqrt(j);
    blockwrite(f1,buf,1);
    writeln(ps^,pk^:4,pr^:10:4);
  end;
  close(f1);
  writeln;
  reset(f1,15);
  for j:=9 downto 0 do
  begin
    seek(f1,j);
    blockread(f1,buf,1);
    writeln(ps^,pk^:4,pr^:10:4);
  end;
  close(f1);
  readln;
end.
```

Основные типы файлов в Си

Система программирования Borland C поддерживает работу с файлами и потоками, данные в которых представлены либо в символьном, либо в двоичном формате. Однако все ранее описанные типы файлов доступны и в программах на Си.

Текстовые (строковые) файлы в Си

Текстовый файл в Си может быть создан путем записи на диск символьных и/или числовых данных по заданному формату с помощью оператора `fprintf`. В качестве признака конца строки здесь заносятся те же самые байты `0D0A`, которые появляются на диске в результате вывода управляющего символа `\n`.

Для инициализации текстового файла необходимо завести указатель на структуру типа `FILE` и открыть файл в одном из нужных режимов ("`rt`" — для ввода, "`wt`" — для вывода, "`at`" — для дозаписи в уже существующий набор данных) по оператору `fopen`:

```
  FILE *f1;
.........
  f1=fopen(имя_файла, "режим");
```

Формат оператора вывода данных в текстовый файл таков:

```
fprintf(f1,"список_форматов \n",список_вывода);
```

Если очередная строка текстового файла формируется из значения элементов символьного массива `str`, то вместо оператора `fprintf(f1,"%s\n",str)` проще воспользоваться оператором `fputs(f1,str)`.

Чтение данных из текстового файла осуществляется с помощью оператора `fscanf(f1,"список_форматов", список_ввода)` или `fgets(str,n,f1)`. В последней функции параметр `n` означает максимальное количество считываемых символов, если раньше не встретится управляющий байт `0A`.

В Си существуют и другие возможности для работы с текстовыми файлами — функции `open`, `creat`, `read`, `write`.

Программа 7_02.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
  main()
{
  FILE *f;
  int j,k;
  float d;
  char s[7]="Строка";
  f=fopen("c_txt","wt");
  for(j=1;j<11;j++)
```

```
    {
      fprintf(f,"%s %d %f\n",s,j,sqrt(j));
      fprintf("\nСтрока %d %n",j,sqrt(j));
    }
  fclose(f);
  fprintf("\n");
  f=fopen("c_txt","rt");
  for(j=10; j>0; j--)
    {
      fscanf(f,"%s %d %f",s,&k,&d);
      fprintf("\n%s %d %f",s,k,d);
    }
  getch();
}
```

Обратите внимание на возможную ошибку при наборе программы 7_01.c. Если между форматными указателями `%s` и `%d` не сделать пробел, то в файле текст "`Строка`" склеится с последующим целым числом. После этого при чтении в переменную `s` будут попадать строки вида "`Строка1`", "`Строка2`", ..., "`Строка10`", в переменную `k` будут считываться старшие цифры корня из `j` (до символа "точка"), а в переменной `d` окажутся дробные разряды соответствующего корня.

Записеориентированные файлы в Си

Записеориентированный файл является частным случаем двоичного файла, в котором в качестве порции обмена выступает структура Си, являющаяся точным аналогом записи в Паскале.

Для инициализации записеориентированного файла необходимо завести указатель на структуру типа `FILE` и открыть файл в одном из нужных режимов ("`rb`" — для ввода, "`wb`" — для вывода, "`ab`" — для дозаписи в уже существующий набор данных) по оператору `fopen`:

```
  FILE *f1;
.........
  f1=fopen(имя_файла, "режим");
```

Формат оператора вывода данных в записеориентированный файл таков:

```
fwrite(buf, size_rec, n_rec, f1);
```

Здесь `size_rec` — размер записи в байтах, а `n_rec` — количество записей, участвующих в обмене. Считывание данных из записеориентированного файла осуществляется с помощью функции `fread`:

```
fread(buf, size_rec, n_rec, f1);
```

Программа 7_02.c в цикле формирует значение записи b, состоящей из символьного (b.s, 7 байт, включая нулевой байт — признак конца строки), целочисленного (`b.n`, 2 байта) и вещественного (`b.r`, 4 байта) полей, и выводит содержимое полей на диск последовательными порциями. Затем файл открывается для ввода и содержимое записей извлекается в обратном порядке. Второй аргумент функции `fseek` определяет величину смещения указателя файла в байтах относительно точки, заданной третьим параметром. Параметр `SEEK_SET` **означает**, что сдвиг указателя в файле производится от его начала. Для сдвига от конца файла используется константа `SEEK_END`, а для сдвига относительно текущей позиции — константа `SEEK_CUR`.

Программа 7_03.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
  main( )
{
  FILE *f1;
  int j,k;
  struct {
    char s[7];
    int n;
    float r;
  } b;
  clrscr();
  strcpy(b.s,"Строка");
  f1=fopen("c_rec","wb");
  for(j=1;j<11;j++)
    {
      b.n=j;    b.r=sqrt(j);
      fwrite(&b,sizeof(b),1,f1);
      printf("\n%s %d %f",b.s,b.n,b.r);
    }
  fclose(f1);
  printf("\n");
  f1=fopen("c_rec","rb");
  for(j=10; j>0; j--)
    {
```

```
      fseek(f1,(j-1)*sizeof(b),SEEK_SET);
      fread(&b,sizeof(b),1,f1);
      printf("\n%s %d %f",b.s,b.n,b.r);
    }
  getch();
}
```

Двоичные файлы в Си

Работа с настоящими двоичными файлами в Си отличается от приведенного выше примера только тем, что обмены с одним и тем же дисковым файлом могут производиться порциями любой длины, тогда как в записеориентированном файле предполагается, что размеры всех порций одинаковы.

Кроме набора функций {fopen/fclose, fread/fwrite}, для работы с двоичными файлами в библиотеке Borland C предусмотрены и другие средства — {_dos_open/_dos_close, _dos_read/_dos_write}, {_creat/_close, _read/_write}. Однако знакомство со всеми возможностями этой библиотеки в рамках настоящего пособия не предусмотрено.

Программа 7_04.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
  main()
{
  FILE *f1;
  int j,k;
  char s[7]="Строка";
  int n;
  float r;
  clrscr();
  f1=fopen("c_bin","wb");
  for(j=1;j<11;j++)
    {
      r=sqrt(j);
      fwrite(s,sizeof(s),1,f1);
      fwrite(&j,sizeof(int),1,f1);
      fwrite(&r,sizeof(float),1,f1);
      printf("\n%s %d %f",s,j,r);
    }
```

```
  fclose(f1);
  printf("\n");
  f1=fopen("c_rec","rb");
  for(j=10; j>0; j--)
    {
      fseek(f1,(j-1)*(sizeof(s)+sizeof(int)+sizeof(float)),
            SEEK_SET);
      fread(&s,sizeof(s),1,f1);
      fread(&n,sizeof(int),1,f1);
      fread(&r,sizeof(float),1,f1);
      printf("\n%s %d %f",s,n,r);
    }
  getch();
}
```

Задачи, советы и ответы

Задание 7.05. Перекодировка текстов из MS-DOS в Windows

Составить программу перекодировки текстового файла, подготовленного редактором MS-DOS, в кодовую страницу 1251 для Windows. Следует попытаться, по возможности сохранить вид таблиц, сформированных с помощью символов псевдографики.

Совет 1 (общий)

Практически все программы перекодировки символов типа байт-в-байт используют 256-байтный словарь соответствий. По коду `j` перекодируемого символа из `j`-го байта словаря извлекается код, заменяющий исходный символ.

Совет 2 (общий)

Для максимального сохранения вида разделительных линий таблиц предлагаются следующие замены:

- символ одинарных горизонтальных линий на знак "`минус`";
- символ двойных горизонтальных линий на знак "`равно`";
- символы одинарных и двойных вертикальных линий на знак "`|`";
- все остальные символы псевдографики на знак "`плюс`".

Совет 3 (общий)

Вообще говоря, словари перекодировки формируются в виде массива из 256 констант. Но для того, чтобы сделать структуру словаря более прозрачной, мы сформируем его программным путем с помощью процедуры `to_win`.

Совет 4 (Си, Паскаль)

Имя перекодируемого файла может быть указано в качестве параметра командной строки. Если этот параметр при запуске программы перекодировки не был задан, то программа должна запросить имя файла. Имя выходного файла для простоты можно зафиксировать, например `tmpwin.txt`.

Программа 7_05.bas

```
DECLARE SUB TOWIN(TO1251() AS INTEGER)
DIM TO1251(256) AS INTEGER
INPUT "Задайте имя файла - ",SOURCE$
TOWIN TO1251()
OPEN SOURCE$ FOR INPUT AS #1
OPEN "TMPWIN.TXT" FOR OUTPUT AS #2
DO WHILE NOT EOF(1)
LINE INPUT #1, A$
FOR J=1 TO LEN(A$)
MID$(A$,J,1)=CHR$(TO1251(ASC(MID$(A$,J,1))))
NEXT J
PRINT #2, A$
LOOP
CLOSE 1,2
END

SUB TOWIN(TO1251() AS INTEGER)
' Сохраняем первую половину таблицы ASCII
FOR J=0 TO 127: TO1251(J)=J: NEXT J
' Увеличиваем на 64 коды букв от "А" до "п"
FOR J=128 TO 175: TO1251(J)=J+64: NEXT J
' Заменяем все символы псевдографики знаком "+"
FOR J=176 TO 223: TO1251(J)=ASC("+"): NEXT J
' Заменяем одинарную вертикальную черту
TO1251(179)=ASC("|")
' Заменяем двойную вертикальную черту
TO1251(186)=ASC("|")
' Заменяем одинарную горизонтальную черту
TO1251(196)=ASC("-")
' Заменяем двойную горизонтальную черту
TO1251(205)=ASC("=")
```

```
' Увеличиваем на 16 коды букв от "р" до "я"
FOR J=224 TO 239: TO1251(J)=J+16: NEXT J
TO1251(240)=168:              ' Заменяем код буквы "ё"
TO1251(241) = 184:   ' Заменяем код буквы "Ё"
END SUB
```

Программа 7_05.c

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void to_win(void);
unsigned char to1251[256];
  main(int narg, char **argv)
{
  FILE *fin,*fout;
  unsigned char str[80],source[80];
  int j,len;
  if(narg==2) strcpy(source,argv[1]);
  else
    {
      printf("\nЗадайте имя исходного файла - ");
      scanf("%s",source);
    }
  to_win();
  fin=fopen(source,"rt");
  fout=fopen("tmpwin.txt","wt");
  while (!feof(fin))
    {
      fgets(str,80,fin);
      len=strlen(str);
      for(j=0; j<len; j++) str[j]=to1251[str[j]];
      fputs(str,fout);
    }
  fcloseall();
}
/*-------------------------------------------*/
void to_win(void)
{
```

```
/* Формирование словаря перекодировки из MS-DOS в Windows */
  int i;
/* Сохраняем первую половину таблицы ASCII */
  for(i=0; i<128; i++) to1251[i]=i;
/* Увеличиваем на 64 коды букв от "А" до "п" */
  for(i=128; i<176; i++) to1251[i]=i+64;
/* Заменяем все символы псевдографики знаком "+" */
  for(i=176; i<224; i++) to1251[i]='+';
/* Заменяем одинарную вертикальную черту */
  to1251[179]='|';
/* Заменяем двойную вертикальную черту */
  to1251[186]='|';
/* Заменяем одинарную горизонтальную черту */
  to1251[196]='-';
/* Заменяем двойную горизонтальную черту */
  to1251[205]='=';
/* Увеличиваем на 16 коды букв от "р" до "я" */
  for(i=224; i<256; i++) to1251[i]=i+16;
  to1251[240]=168;   /* Заменяем код буквы "ё" */
  to1251[241]=184;   /* Заменяем код буквы "Ё" */
  return;
}
```

Программа 7_05.pas

```
program translate;
var
  to1251:array [0..255] of char;
  fin,fout:text;
  str:string[80];
  source:string[80];
  j,len:integer;
procedure to_win;
var
  i:integer;
begin
  for i:=0   to 127 do to1251[i]:=chr(i);
  for i:=128 to 175 do to1251[i]:=chr(i+64);
  for i:=176 to 223 do to1251[i]:='+';
  to1251[179]:='|';
```

```
  to1251[186]:='|';
  to1251[196]:= '-';
  to1251[205]:= '=';
  for i:=224 to 239 do to1251[i]:=chr(i+16);
  to1251[240]:=chr(168);
  to1251[241]:=chr(184);
end;
begin
  if(ParamCount=2) then source:=ParamStr(1)
  else begin
    write('Задайте имя исходного файла - ');
    readln(source);
  end;
  to_win;
  assign(fin,source);
  reset(fin);
  assign(fout, 'tmpwin.txt');
  rewrite(fout);
  while (not eof(fin)) do
  begin
    readln(fin,str);
    len:=length(str);
    for j:=0 to len do str[j]:=to1251[ord(str[j])];
    writeln(fout,str);
  end;
  close(fin);
  close(fout);
end.
```

Задание 7.06. Телефонный справочник

Составить программу, которая открывает текстовый файл для дозаписи и формирует в нем список, содержащий фамилию абонента и его телефон. Сортировкой фамилий в алфавитном порядке и исключением дублирующихся строк мы пока заниматься не будем.

Совет 1 (QBasic)

Для записи двух символьных значений в одну строку текстового файла разумно воспользоваться оператором `WRITE #`, т. к. в этом случае мы получим доступ к каждой компоненте строки дискового файла.

Совет 3 (Паскаль)

Паскаль не позволяет открыть несуществующий файл на дозапись. Поэтому перед обращением к процедуре `append` приходится отключать системный контроль за ошибками ввода/вывода. Если открытие на дозапись не состоялось, то в первый раз приходится открывать файл `notebook` на запись. При чтении два текстовых значения в одной файловой строке здесь, конечно, можно разделить — телефон занимает фиксированное количество позиций в конце строки, между фамилией и телефоном находится какое-то количество пробелов. Однако для вывода на экран это не требуется.

Программа 7_06.bas

```
CLS
OPEN "notebook" FOR APPEND AS #1
'Ввод данных с клавиатуры
DO
INPUT "ФАМИЛИЯ: ",Name$
INPUT "ТЕЛЕФОН: ",Phone$
WRITE #1, Name$,Phone$
INPUT "Добавим ";R$
LOOP WHILE LEFT$(R$,1)="д"
CLOSE #1
CLS
OPEN "notebook" FOR INPUT AS #1
PRINT "Список абонентов в файле:"
DO WHILE NOT EOF(1)
  'LINE INPUT #1, NP$   ' Чтение строки целиком
  'PRINT NP$            ' Вывод данных на экран
  INPUT #1, A$, B$      ' Чтение строки по компонентам
  PRINT A$, B$          ' Вывод данных на экран
LOOP
CLOSE #1
END
```

Программа 7_06.c

```
#include <conio.h>
#include <stdio.h>
  main()
{
  FILE *f;
```

```
  int k;
  char r,Name[20],Phone[10];
  clrscr();
  f=fopen("notebook","at");
m:  /* Ввод данных с клавиатуры */
  printf("\nФАМИЛИЯ: ");   scanf("%s",Name);
  printf("ТЕЛЕФОН: ");       scanf("%s",Phone);
  fprintf(f,"%-20s %10s\n",Name,Phone);
  printf("Добавим (д/н) - "); r=getche();
  if(r=='д') goto m;
  fclose(f);
  clrscr();
  f=fopen("notebook","rt");
  printf("\nСписок абонентов в файле:\n");
  while(!feof(f))
    {
      fscanf(f,"%20s %10s\n",Name,Phone);
      printf("%-20s %10s\n",Name,Phone);
    }
  fclose(f);
  getch();
}
```

Программа 7_06.pas

```
program notebook;
uses Crt;
var
  f:text;
  R,Name,Phone:string;
begin
  clrscr;
  assign(f, 'notebook ');
  {$I-} append(f); {$I+}
  if IOResult <> 0 then rewrite(f);
  { Ввод данных с клавиатуры }
  repeat
    write('ФАМИЛИЯ: ');   readln(Name);
    write('ТЕЛЕФОН: ');   readln(Phone);
```

```
    writeln(f,Name, ' ':15-length(Name), Phone:10);
    write('Добавим ');    readln(R);
  until r[1] <> 'д ';
  close(f);
  clrscr;
  reset(f);
  writeln('Список абонентов в файле: ');
  repeat
    readln(f,r);  { Чтение строки целиком }
    writeln(r);   { Вывод данных на экран }
  until eof(f);
  close(f);
  readln;
end.
```

Задание 7.07. Создание резервной копии файла

Написать программу, которая извлекает из командной строки имя файла и создает в том же каталоге резервную копию файла с расширением bak.

Совет 1 (общий)

Чтобы избежать нежелательного влияния управляющих символов с кодами `0D` ("`Возврат каретки`"), `0A` ("`Перевод строки`"), `1A` ("`Признак конца файла`") целесообразно рассматривать резервируемые данные независимо от их происхождения как двоичный файл.

Совет 2 (QBasic)

Так как в этой системе работа с командной строкой не реализована, мы ограничимся вводом имени исходного файла по запросу программы.

Совет 3 (Си)

Функция `strchr(name1,'.')` определяет позицию точки в имени исходного файла. Точнее, ее значение равно указателю на позицию точки, если таковая в имени `name1` содержится. В противном случае функция `strchr` возвращает `NULL`. В первом случае с помощью функции `strncpy` в строку `name2` копируется начало имени до символа '.' и скопированная часть имени принудительно завершается признаком конца строки (символом `0x0`).

Совет 4 (Паскаль)

Отключение системного контроля за ошибками ввода/вывода (`{$I-}`) перед открытием исходного файла сделано для того, чтобы взять на себя проверку несостоявшейся операции и выдать пользователю более осмысленное сообщение. Функция `fsplit` расчленяет полную спецификацию файла на путь до

каталога, содержащего исходный файл (`Dir`), собственно имя файла (`Name`) и его расширение (`Ext`). Программа будет работать быстрее, если размер буфера для копирования очередной порции увеличить до 32 768 (кратность 512 здесь желательна, т. к. это число совпадает с длиной физического сектора).

Программа 7_07.bas

```
CLS : DIM k AS STRING*1
INPUT "Задайте имя файла - ", NAME1$
OPEN NAME1$ FOR BINARY AS #1
K = INSTR(NAME1$, ".")
IF K = 0 THEN NAME2$ = NAME1$ ELSE NAME2$ = LEFT$(NAME1$, K - 1)
NAME2$ = NAME2$ + ".BAK"
PRINT NAME2$, K
OPEN NAME2$ FOR BINARY AS #2
DO
GET #1, , k
PUT #2, , k
LOOP UNTIL (EOF(1))
END
```

Программа 7_07.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 512
  main(int narg, char **argv)
{
  FILE *f1,*f2;
  int rd,wr;
  char buf[N];
  char name1[80],name2[80];
  char *point;
  if(narg < 2)
    { printf("\nНе задано имя файла"); exit(0); }
  strcpy(name1,argv[1]);
  f1=fopen(name1,"rb");
  if(f1==NULL)
    { printf("\nФайл %s не найден",name1);  exit(0); }
```

```
  point=strchr(name1, '. ');
  if(point)
    {
      strncpy(name2,name1,point-name1);
     name2[point-name1]=0x0;
    }
  else strcpy(name2,name1);
  strcat(name2,".bak");
  f2=fopen(name2,"wb");
  do
    {
     rd=fread(buf,1,N,f1);
     wr=fwrite(buf,1,rd,f2);
    }
  while (rd!=0 && rd == wr);
  fclose(f1);
  fclose(f2);
}
```

Программа 7_07.pas

```
program reserve;
uses Dos;
const
  N=512;
var
  f1,f2 : file;
  rd,wr : word;
  buf : array [1..N] of byte;
  name1, name2 : PathStr;
  dir : DirStr;
  name : NameStr;
  ext : ExtStr;
begin
  if ParamCount=0 then
     begin writeln('Не задано имя файла'); exit; end;
  name1:=ParamStr(1);
  assign(f1,name1);
  {$I-} reset(f1,1); {$I+}
```

```
  if IOResult <> 0 then
    begin writeln('Файл ',name1, ' не найден');  exit; end;
  fsplit(name1,dir,name,ext);
  name2:=dir+name+'.bak';
  assign(f2,name2);
  rewrite(f2,1);
  repeat
    blockread(f1,buf,N,rd);
    blockwrite(f2,buf,rd,wr);
  until (rd=0) or (rd <> wr);
  close(f1);    close(f2);
end.
```

Задание 7.08. Выдача каталога на экран

Составить программу, которая извлекает из текущего каталога имена файлов с заданным расширением (например, *.pas) и выводит их на экран подобно директиве MS-DOS (`dir *.pas`), суммируя количество и длину обнаруженных файлов.

Совет 1 (QBasic)

Система QBasic не включает в свой состав функции или процедуры для работы с файловой системой MS-DOS, присущие алгоритмическим языкам более высокого уровня типа Си или Паскаля. Поэтому мы ограничимся тем, что из программы на Basic можно выполнить любую директиву операционной системы по оператору `SHELL`. Программа 7_07.bas решает поставленную задачу, однако имена файлов из текущего или указанного каталога остаются недоступными для последующей программной обработки.

Совет 2 (Си)

Для извлечения файлов из текущего или указанного каталога можно воспользоваться функциями `findfirst(char *path, struct ffblk sr, int attr)` и `findnext(struct ffblk sr)`. Аргумент `path` является указателем на маску отбора файлов (например — "`*.c`"), расширенную спецификацию маски (например — "`c:\bc\source\*.c`") или просто путь (например — "`c:\bc`"). Аргумент `sr` представляет собой структуру типа `ffblk`, на поля которой заносятся атрибуты найденного набора. Эта структура описана в файле `dir.h` и имеет следующие поля:

```
struct ffblk
  {
    char ff_reserved[21]; //резерв на будущее
    char ff_attrib;       //атрибут набора данных
```

```
    unsigned ff_ftime;     //упакованное время создания
    unsigned ff_fdate;     //упакованная дата создания
    long ff_fsize;         //длина набора в байтах
    char ff_name[13];      //имя набора
  };
```

Третий аргумент определяет атрибуты, которыми должен быть снабжен отбираемый набор данных, и может принимать разумную логическую сумму из значений, определенных в файле `dos.h` и задаваемых в Си следующими константами:

- `FA_RDONLY` — отбирать файлы с признаком "Только для чтения";
- `FA_HIDDEN` — отбирать файлы с признаком "Скрытый";
- `FA_SYSTEM` — отбирать системные файлы;
- `FA_LABEL` — отбирать наборы с меткой "Идентификатор тома";
- `FA_DIREC` — отбирать каталоги;
- `FA_ARCH` — отбирать файлы с признаком "Архивировать".

Цикл поиска нужных наборов данных начинается с обращения к процедуре `findfirst` (поиск первого объекта) и повторяется многократными обращениями к процедуре `findnext` (поиск следующего объекта). Аргумент `sr` в обоих обращениях должен быть один и тот же. Цикл поиска продолжается до тех пор, пока функция поиска возвращает нулевое значение. Как только очередная попытка окажется неудачной, функция поиска возвращает значение –1.

Совет 3 (Си)

Самый кропотливый момент в программе связан с выводом упакованных значений времени и даты последнего обновления файла, т. к. в системной библиотеке Си подходящей функции распаковки нет. Поэтому в программе 7_07.c введены две структуры битовых полей, эквивалентные способам упаковки даты (`struct dat`) и времени (`struct tim`). Обратите внимание на последовательность описания битовых полей — от младших разрядов слова к старшим (а не наоборот). С помощью объединения `union` данные указанных структур накладываются на беззнаковые целочисленные (`unsigned`) переменные, в которые предварительно переписываются упакованные данные. Последующее извлечение содержимого битовых полей использует соответствующие составные имена, например из упакованного значения даты `v.d` извлекаются разряды, в которых находится номер дня `v.c.day`. Единственное изменение содержимого поля `v.c.year` связано с тем, что функции поиска в MS-DOS возвращают значение года, уменьшенное на 1980.

Вообще говоря, упакованные данные можно было бы извлечь и другим способом — логически умножить упакованную величину на константу, содержащую сплошные единицы в соответствующем поле, и сдвинуть полученный результат на нужное число разрядов вправо.

Совет 4 (Паскаль)

Для извлечения файлов из текущего или указанного каталога можно воспользоваться процедурами `FindFirst(path, attr, sr)` и `FindNext(sr)`, вклю-

ченными в состав модуля `Dos`. Аргумент `path` задает маску отбора файлов (например, `"*.pas"`), расширенную спецификацию маски (например, `"c:\tp\source\*.pas"`) или просто путь (например, `"c:\tp"`). Второй аргумент определяет атрибут, которым должен быть снабжен отбираемый набор данных, и может принимать логическую сумму из следующих значений, задаваемых в Паскале мнемоническими константами:

- `ReadOnly` – отбирать файлы с признаком "Только для чтения";
- `Hidden` – отбирать файлы с признаком "Скрытый";
- `SysFile` – отбирать системные файлы;
- `VolumeID` – отбирать наборы с меткой "Идентификатор тома";
- `Directory` – отбирать каталоги;
- `Archive` – отбирать файлы с признаком "Архивировать";
- `AnyFile` – брать любые файлы.

Аргумент `sr` представляет собой запись типа `SearchRec`, на поля которой заносятся атрибуты найденного набора. Этот тип описан в модуле `Dos` и имеет следующую структуру:

```
type
  SearchRec=record
    Fill:array [1..21] of byte;
    Attr:byte;        {атрибут набора данных}
    Time:longint;     {упакованные дата и время создания}
    Size:longint;     {длина набора в байтах}
    Name:string[12]; {имя набора}
  end;
```

Цикл поиска нужных наборов данных начинается с обращения к процедуре `FindFirst` (поиск первого объекта) и повторяется многократными обращениями к процедуре `FindNext` (поиск следующего объекта). Аргумент `sr` в обоих обращениях должен быть один и тот же. Цикл поиска продолжается до тех пор, пока системная переменная `DosError` принимает нулевое значение. Как только очередная попытка окажется неудачной, в `DosError` заносится 18.

Совет 5 (Паскаль)

Распаковка даты и времени осуществляется с помощью системной процедуры `UnpackTime`, первым аргументом которой является упакованный набор, возвращаемый процедурами поиска, а вторым — запись с распакованными полями типа `DateTime`, описанная в модуле `Dos`. Для придания единообразия колонкам с числовыми данными полезно малые (`k < 10`) числа дней, месяцев, часов, минут и секунд предварять лидирующим нулем. В случае необходимости такая добавка выполняется функцией `Zero`.

Программа 7_08.bas

```
SHELL "dir *.bas"
END
```

Программа 7_08.c

```
#include <stdio.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>
  main()
{
  struct ffblk sr;
  int k,nf=0;
  long lf=0;
  struct tim {
    unsigned sec:5;
    unsigned min:6;
    unsigned hour:5;
  };
  struct dat {
    unsigned day:5;
    unsigned month:4;
    unsigned year:7;
  };
  union {struct tim a; unsigned b;}u;
  union {struct dat c; unsigned d;}v;
  clrscr();
  printf("Имя файла Длина Дата Время");
  k=findfirst("*.c",&sr,FA_ARCH);
  while (k==0)
    {
      u.b=sr.ff_ftime;
      v.d=sr.ff_fdate;
      nf++;
      lf+= sr.ff_fsize;
      printf("\n%12s %6ld ",sr.ff_name,sr.ff_fsize);
      printf(" %02d/%02d/%4d ",v.c.day,v.c.month,1980+v.c.year);
```

```
      printf(" %02d:%02d",u.a.hour, u.a.min);
      k=findnext(&sr);
    }
  printf("\n%d файлов занимают %ld байт",nf,lf);
  getch();
}
```

Программа 7_08.pas

```
program dir;
uses Crt,Dos;
var
  dt:DateTime;
  sr:SearchRec;
const
  nf:integer=0;
  lf:longint=0;
function Zero(k:byte):string;
var
  s:string;
begin
  Str(k,s);
  if k>9 then Zero:=s
  else Zero:= '0 '+s;
end;
begin
  clrscr;
  writeln('Имя файла Длина Дата Время');
  FindFirst('*.pas',AnyFile,sr);
  while DosError=0 do begin
    with sr,dt do begin
      inc(nf);
      lf:=lf+Size;
      UnpackTime(Time,dt);
      write(Name:12, ' ',Size:6, ' ');
      write(Zero(Day), '/',Zero(Month), '/',Year:4, ' ');
      writeln(Zero(Hour), ': ',Zero(Min), ': ',Zero(Sec));
    end;
```

```
    FindNext(sr);
  end;
  writeln(nf, ' файлов занимают ',lf, ' байт');
  readln;
end.
```

Задание 7.09. Сдвиг содержимого текстового файла

Составить программу сдвига содержимого каждой строки текстового файла на заданное число позиций вправо. Это может оказаться полезным для формирования левого поля нужной ширины перед выводом содержимого файла на принтер по командам операционной системы `PRINT` или `COPY`.

Совет 1 (QBasic)

Для считывания строк из текстового файла нужно использовать оператор `LINE INPUT`, иначе головные пробелы в считываемых строках будут игнорироваться и каждая строка будет обрезана по первому же пробелу после значащих символов.

Совет 2 (Си)

В функции `ind_copy` начало массива `str` предварительно расписывается заданным количеством пробелов, а затем в его оставшуюся часть считывается очередная строка исходного файла.

Совет 3 (Паскаль)

В приведенных ниже двух вариантах программы использованы два разных подхода к чтению данных из текстового файла. В программе 7_08.pas очередная строка исходного файла считывается целиком и также целиком переписывается в выходной файл. Предварительно в ту же запись заносится n пробелов за счет указателя ширины поля после текстового операнда, содержащего единственный пробел. В программе 7_08a.pas, алгоритм которой менее рационален, строка исходного файла читается посимвольно до тех пор, пока функция `eoln(f1)` не обнаруживает признак конца строки. Такой прием может оказаться полезным, если строка текстового файла содержит несколько разнотипных компонент и разбираться с ними приходится, анализируя каждый символ. Однако и в этой ситуации целесообразно прочитать строку целиком и устроить аналогичную разборку в оперативной памяти.

Программа 7_09.bas

```
INPUT "Задайте имя исходного файла - ", NAME1$
INPUT "Задайте имя выходного файла - ", NAME2$
INPUT "Задайте величину сдвига - ", N%
```

```
OPEN NAME1$ FOR INPUT AS #1
OPEN NAME2$ FOR OUTPUT AS #2
DO WHILE NOT EOF(1)
LINE INPUT #1,A$: A$=SPACE$(N%)+A$
PRINT #2,A$
LOOP
CLOSE #1: CLOSE #2
END
```

Программа 7_09.c

```
#include <stdio.h>
#include <stdlib.h>
void ind_copy(FILE *f1,FILE *f2,int n);
  main(int narg, char **argv)
{
  FILE *f1,*f2;
  int n;
  if(narg < 4)
    {
      printf("\nОшибка. Должно быть :");
      printf("\n7_09.exe файл1 файл2 n");
      exit(0);
    }
  f1=fopen(argv[1],"rt");
  f2=fopen(argv[2],"wt");
  n=atoi(argv[3]);
  ind_copy(f1,f2,n);
  fcloseall();
}
/*-----------------------------------------*/
void ind_copy(FILE *f1,FILE *f2,int n)
{
  char str[80];
  int j;
  for(j=0; j<n; j++) str[j]=' ';
  while (!feof(f1))
    {
      fgets(&str[n],80,f1);
```

```
      fputs(str,f2);
    }
  return;
}
```

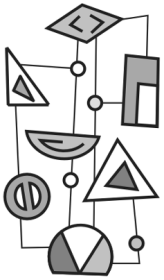
Программа 7_09.pas

```
program indent;
var
  f1,f2:text;
  name1,name2:string;
  n,k:integer;
procedure ind_copy(var f1,f2:text;n:integer);
var
  str:string;
begin
  while not eof(f1) do
  begin
    readln(f1,str);
    writeln(f2, ' ':n,str);
  end;
end;
begin
  if ParamCount < 3 then begin
    writeln('Параметры заданы неверно. Должно быть: ');
    writeln('7_09.exe файл1 файл2 сдвиг ');
    exit;
  end;
  name1:=ParamStr(1);
  name2:=ParamStr(2);
  Val(ParamStr(3),n,k);
  assign(f1,name1);    reset(f1);
  assign(f2,name2);    rewrite(f2);
  ind_copy(f1,f2,n);
  close(f1);
  close(f2);
end.
```

Программа 7_09a.pas

```
program indent1;
var
  f1,f2:text;
  name1,name2:string;
  n,k:integer;
procedure ind_copy(var f1,f2:text;n:integer);
var
  ch:char;
begin
  while not eof(f1) do begin
    write(f2, ' ':n);
    while not eoln(f1) do begin
      read(f1,ch);  write(f2,ch);
    end;
    readln(f1);
    writeln(f2);
  end;
end;
begin
  if ParamCount < 3 then begin
    writeln('Параметры заданы неверно. Должно быть: ');
    writeln('7_09a.exe файл1 файл2 сдвиг ');
    exit;
  end;
  name1:=ParamStr(1);
  name2:=ParamStr(2);
  Val(ParamStr(3),n,k);
  assign(f1,name1);     reset(f1);
  assign(f2,name2);     rewrite(f2);
  ind_copy(f1,f2,n);
  close(f1);
  close(f2);
end.
```

Глава 8



Машинная графика

Достижения современных мультимедийных технологий активно проявляются в компьютерных играх, в рекламе на многочисленных Internet-сайтах, в научно-фантастических и приключенческих фильмах, в телевизионных роликах и т. п. И наиболее важной программной компонентой во всем этом разнообразии является компьютерная графика. Рассматриваемые в нашем пособии системы программирования появились 12—15 лет тому назад и их изобразительные средства весьма ограничены. Но мы не можем не познакомить читателя с базовыми графическими процедурами, которые оказываются столь полезными при оформлении результатов работы наших программ.

О мониторах и графических системах

Существует довольно много разных типов мониторов, объединяемых названиями VGA (Video Graphic Array — видеографический массив) и SVGA (Super VGA). Монитор в сочетании с управляющей схемой, оформленной в виде отдельной платы (видеокарты) или специализированной микросхемы на материнской плате, образует так называемую графическую систему. Наиболее важными характеристиками графической системы являются габариты экрана, измеряемые длиной диагонали в дюймах (14", 15", 17", 19" и более), разрешающая способность, измеряемая максимальным количеством точек, отображаемых по горизонтали и вертикали (640x480, 800x600, 1024x768, 1280x1024 и более), и цветовая гамма, характеризуемая максимальным числом различимых цветовых оттенков (16 цветов, 64 цвета, 256 цветов, 256 тысяч цветов, 16 миллионов цветов). Несмотря на такое разнообразие, общей для всех видеосистем является возможность работать в текстовом или графическом режимах. На самом деле, и текстовых, и графических режимов насчитывается довольно много (порядка 18—20), но в рамках рассматриваемых систем программирования приходится иметь дело с весьма ограниченным их подмножеством.

В текстовом режиме экран напоминает лист бумаги в крупную клетку, на котором в каждой клетке можно поместить только один символ (букву, цифру, знак препинания и прочие разделители). При этом сама клетка может быть окрашена в один из 8 цветов, а контур буквы — в один из 16 цветов.

В графическом режиме экран напоминает миллиметровку, т. е. разбит на довольно много мелких клеток (сторона клетки порядка 0,3—0,4 мм для мо-

ниторов с 14—15-дюймовыми экранами). Такие клетки называют пикселами (pixel — от picture's element, "элемент рисунка") и каждый из них может быть окрашен в тот или иной цвет. Самый высокий графический режим, поддерживаемый нашими системами программирования, соответствует стандарту VGA — 640 точек по горизонтали, 480 — по вертикали, 16 цветов из довольно большого числа (256 K) возможных цветовых оттенков.

О системах координат и текущей точке

Как в текстовом, так и в графическом режимах начало экранных координат располагается в левом верхнем углу экрана. Однако в текстовом режиме самая левая верхняя клетка имеет координаты (1,1), тогда как самый первый пиксел в графическом режиме имеет координаты (0,0). Ось `x` направлена вправо, а ось `y` — вниз.

В отличие от Turbo C и Turbo Pascal система QBasic предоставляет пользователю возможность назначить свою собственную систему координат, привязав к противоположным углам области вывода минимальные и максимальные значения координат конкретной задачи по осям `x` и `y`.

Ряд графических процедур наряду с абсолютными координатами использует и относительные координаты, задаваемые в приращениях (`dx`,`dy`) относительно положения так называемой текущей точки (`CP` — Current Point). При активизации графического режима текущая точка помещается в начало координат. Ее последующие перемещения зависят от выполняемых операций. Например, при построении видимого или невидимого отрезка[КЮ2] текущая точка перемещается из начала отрезка в его конец. При построении окружности текущая точка находится в центре и никуда не смещается после выполнения операции.

Иногда текущую точку называют графическим курсором, который, в отличие от текстового, не изображается на экране, чтобы не исказить выводимую картинку. Однако программа в любой момент может узнать координаты текущей точки.

О видеопамяти

Все, что представлено на экране дисплея, является отображением содержимого видеопамяти — специального запоминающего устройства, расположенного на видеокарте. В большинстве ПК емкость видеопамяти варьируется сегодня в диапазоне от 2 до 32 Мбайт, и чем больше ее размер, тем большими возможностями по разрешению, цветовой гамме и специализированным графическим операциям располагает компьютер.

В тех графических режимах, с которыми мы будем знакомиться, видеопамять предоставляет пользователю одну или две страницы с номерами 0 и 1. Каждая из них может быть объявлена активной и/или видимой. Содержимое видимой страницы представлено на экране дисплея, а в активной странице происходит

формирование нового изображения. Одна и та же страница может быть и активной, и видимой. С целью ускорения процедуры смены кадров в мультимедийных приложениях желательно смотреть на одну страницу, а рисовать в другой и в нужный момент произвести переключение страниц.

Все системы программирования позволяют в два-три приема произвести быстрый обмен между содержимым ячеек видеопамяти и оперативной памяти, что бывает крайне необходимо при анимации изображений.

Как формируется RGB-цвет пикселов

Сокращение RGB происходит от английских обозначений базовых цветов — Red (красный), Green (зеленый) и Blue (синий). Именно эти цвета, смешанные в определенной пропорции, и управляют окраской пикселов на экране. В видеопамяти каждому пикселу отводится 2, 4, 8 или 24 двоичных разряда, запоминающих код цвета. В стандарте VGA используются только 4 бита, по которым определяется один из 16 регистров палитры, участвующий в формировании цвета. В каждом 6-разрядном регистре палитры по 2 разряда используются для задания интенсивности соответствующей RGB-компоненты. Это дает возможность манипулировать 64 цветами, но выбрать из них в каждый конкретный момент можно только 16. На самом деле механизм формирования сигнала управления цветом гораздо сложнее и в нем задействованы еще 256 18-разрядных регистров DAC (Digital Analog Converter — цифро-аналоговый преобразователь, русское сокращение ЦАП). С их помощью гамма цветовых оттенков увеличивается до 256 K.

В каждый момент графическая система имеет дело с двумя ранее установленными цветами — цветом переднего плана (цветом рисования —foreground color) и цветом заднего плана (цветом фона — background color). Цвету фона всегда соответствует нулевой программный код цвета. Физический цвет фона определяется содержимым регистра палитры с номером 0. По умолчанию в нем находится нулевой шестибитовый код, задающий нулевую интенсивность по каждой из базовых компонент цвета, что соответствует черному цвету. Изменение содержимого этого регистра изменяет фон экрана только в тот момент времени, когда программа обращается к процедуре очистки экрана.

В отличие от этого, изменение цвета пиксела в процедурах рисования происходит мгновенно, в момент записи нового кода цвета в соответствующую область видеопамяти. Видеокарта позволяет сформировать новый цвет как результат взаимодействия старого и нового кодов по выбранному правилу. Среди возможных правил — затирание прежнего кода новым кодом цвета или его инверсией, взаимодействие старого и нового кодов по одной из логических операций — `AND`, `OR`, `XOR`. Особо хотим обратить внимание на последнюю логическую операцию, которая используется для стирания или

"проявления" изображений. Если на код цвета пиксела накладывается такой же двоичный код по операции XOR, то результат будет нулевым, что соответствует цвету фона или стиранию прежней точки. Если на нулевой код цвета по операции XOR наложить прежний цвет пиксела, то точка "проявится". Этой возможностью часто пользуются при анимации изображений.

Краткий обзор графических возможностей систем программирования

В каждой из трех рассматриваемых систем программирования предусмотрены базовые графические средства, с помощью которых можно устанавливать нужный режим работы видеосистемы, управлять цветовой палитрой, рисовать простейшие геометрические фигуры и раскрашивать их, снабжать рисунки пояснительными подписями.

В системе QBasic эти средства встроены в язык — наряду с обычными операторами в программе можно использовать графические: LINE, CIRCLE, DRAW и др. С одной стороны, набор базовых графических операций QBasic, несколько беднее, чем состав аналогичных процедур и функций в системах фирмы Borland. В частности, QBasic значительно уступает своим конкурентам по возможностям отображения текстовых сообщений и управления шрифтами в графическом режиме.

С другой стороны, QBasic имеет в составе своих изобразительных средств элементы "черепашьей" графики, позволяющей строить графические процедуры любой сложности и использовать сформированные таким образом графические объекты как строительные кирпичики, включая их в состав более сложных объектов. "Черепашья" графика появилась как обобщение системы команд управления пишущим узлом перьевого плоттера. Такие процедуры, как смена, подъем и опускание пера, перемещение пишущего узла на заданное число шагов по одному из восьми направлений и некоторые другие непосредственно заимствованы из этой системы команд. Однако возможность создания графических подпрограмм и построение фигур с учетом различных преобразований (смещение, поворот, масштабирование, непропорциональное изменение размеров фигур вдоль осей координат) придают "черепашьей" графике необычайную гибкость. В нашем пособии оператор DRAW не затрагивается, однако для любителей экзотики мы рекомендуем увлекательную книгу [7], переводчик которой, на наш взгляд, вложил в нее гораздо больше, чем ее предполагаемый автор. Программирование в терминах микроскопических команд плоттера — занятие довольно утомительное, да и отладка таких программ доставляет немало забот. Не менее сложно запоминать односимвольные команды управления "черепашкой". До версии TP 3.0 фирма Borland использовала "черепашью" графику, но в последующих реализациях системы отказалась от нее.

Графические средства систем Turbo C и Turbo Pascal построены на базе общего подхода, сокращенно именуемого BGI — Borland Graphics Interface (графический интерфейс фирмы Borland). Они вынесены в системные библиотеки, содержащие почти одинаковый набор процедур и функций с полностью совпадающими именами и аналогичным набором аргументов. Состав этих библиотек довольно внушителен — 83 графические программы и более 60 системных констант. Наиболее содержательное описание BGI-пакета можно найти в [16].

Небольшая разница между Си и Паскалем наблюдается в записи имен процедур и системных констант, составленных из нескольких ключевых слов. В обозначениях процедур Паскаль предпочитает выделять начало каждого ключевого слова прописной буквой, например — `SetUserCharSize`. В системе Turbo C, в отличие от Паскаля, возможен режим работы, при котором прописные и строчные буквы считаются разными. Поэтому в Си составные имена графических функций записываются только строчными буквами — `setusercharsize`, хотя это и менее наглядно. Паскаль распространяет свою технику выделения ключевых слов и на составные обозначения системных констант (например — `WideDotFill`), тогда как в Си для этой цели обычно используют символ "_" (подчеркивание) и обозначают системные константы только большими буквами (например — `WIDE_DOT_FILL`). Еще одно небольшое отличие связано с записью функций без параметров — в программах на Си их имена всегда сопровождаются пустыми скобками:

```
TC: x=getmaxx();
TP: x:=GetMaxX;
```

Различия эти не носят принципиального характера, но в программах необходимо придерживаться правил, принятых в той или иной системе. В связи с этим графические программы на Си и Паскале мы постараемся не дублировать и будем отдавать предпочтение текстам программ на Си, т. к. существует достаточно много книг с примерами графических программ на Паскале.

Инициализация графического режима

Переход в графический режим очень напоминает подготовку к работе с файлами. В жаргоне программистов бытует термин "открыть графику", и это надо сделать прежде, чем вы обратитесь к любой графической процедуре. Что реально скрывается за этим действием, знать не обязательно.

В QBasic графический режим работы монитора устанавливается после выполнения оператора `SCREEN`, содержащего от одного до четырех параметров:

```
SCREEN n [,[cs][,ap][,vp]]
```

Обязательным является только первый параметр, определяющий номер графического режима. Рекомендуемые значения:

```
n= 9 (640×350 точек, 16  цветов из 64 возможных)
```

```
n=12 (640×480 точек, 16  цветов из 64 возможных)
n=13 (320×200 точек, 256 цветов из 256К возможных)
```

Параметр `cs` в приведенных режимах смысла не имеет. Два последних параметра задают номера активной (`ap = 0` или `ap = 1`) и видимой (`vp = 0` или `vp = 1`) страниц в режиме `n = 9`.

Возврат в текстовый режим осуществляется по оператору `SCREEN 0`.

Программы на Си или Паскале, использующие графический вывод, должны подключить соответствующие системные средства:

```
TC: #include <graphics.h>
TP: uses Graph;
```

В среде BGI графика "открывается" обращением к процедуре `initgraph`:

```
TC : initgraph(&gd,&gm,"path");
TP : InitGraph(gd,gm,'path');
```

Первые два параметра представлены именами целочисленных (`int`, `integer`) переменных, в которых графическая система запоминает условные номера графического драйвера (`gd`) и графического режима (`gm`). Программист больше этими переменными не пользуется, но и не должен затирать их значения. Обычно в переменную `gd` перед обращением к `initgraph` заносят нулевое значение (`gd = 0` или `gd = DETECT`), что заставляет графическую систему определить тип видеосистемы и выбрать подходящий режим ее работы без участия программиста. Как правило, будет установлен режим с названием VGAHI, соответствующий разрешению 640x480 с 16 цветами из 64 возможных.

Третий параметр задает путь к каталогу, в котором находится драйвер видеосистемы — программа, реализующая связь библиотечных процедур с видеокартой. В 99% случаев такой программой является файл `egavga.bgi`, который программисты копируют в свой текущий каталог или прокладывают к нему дорожку с помощью команды `PATH` в файле `autoexec.bat`. В любой из этих двух ситуаций третий параметр может быть задан пустой строкой. Но если графическая система не обнаружит нужный драйвер в доступных каталогах, то работа программы будет прекращена из-за того, что "открытие" графики не состоялось.

Возврат в текстовый режим обычно происходит при обращении к `closegraph`:

```
TC: closegraph();
TP: CloseGraph;
```

Не забывайте "закрывать" графику перед окончанием работы вашей программы, иначе после возвращения в интегрированную среду русские буквы в комментариях и текстовых константах превратятся в нечто странное.

Определение области графического вывода и выбор системы координат

По умолчанию областью вывода графики является весь экран. Однако каждая из систем программирования позволяет объявить зоной вывода некоторую прямоугольную область, выход за пределы которой обычно заблокирован, но может быть и разрешен, если вы отключите режим отсечения. Последняя возможность предусмотрена только в рамках BGI-пакета.

В системе QBasic область графического вывода переопределяется оператором `VIEW`, имеющим две модификации:

```
VIEW (x1,y1)-(x2,y2),cf,cb
VIEW SCREEN (x1,y1)-(x2,y2),cf,cb
```

Каждая из них устанавливает прямоугольную область, левая верхняя вершина которой задана экранными координатами (`x1`,`y1`), а противоположная вершина — координатами (`x2`,`y2`). Попытка задать область, выходящую за пределы экрана ($0 \le x \le 639$, $0 \le y \le 479$), системой пресекается. Целые числа `cf` и `cb` задают, соответственно, цвет фона и цвет рамки, которая окаймляет область вывода. Область вывода и ее границы сразу же окрашиваются в указанные цвета.

В определенной таким образом области вывода действует одна из двух оконных систем координат. В первой из них начало координат находится в точке с экранными координатами (`x1`,`y1`), т. е. в левом верхнем углу окна, ось `x` направлена вправо, ось `y` — вниз, абсолютные координаты точек только целочисленные и положительные. Добавка `SCREEN` переносит начало отсчета координат с таким же взаимным расположением осей в левый верхний угол экрана, т. е. в точку (0,0). В любом случае все, что имеет координаты за пределами области вывода, отображаться не будет.

Приводимый ниже пример сначала объявляет квадратную область вывода, начинающуюся в точке с экранными координатами (10,10), ширина и высота которой равна 200 пикселам. Начало координат в этом окне физически находится в точке с экранными координатами (10,10). Поэтому два следующих оператора `LINE` строят горизонтальную и вертикальную линии, точно проходящие через середину области. Второй оператор (`VIEW SCREEN`) объявляет такую же по размерам квадратную область, привязанную своим левым верхним углом к точке с экранными координатами (310,10). Однако в этой области действует экранная система координат с началом в точке (0,0). И для построения аналогичных прямых, проходящих через центр второй области, их концы приходится задавать в координатах экрана. Выполнив этот пример на компьютере, вы можете убедиться в том, что на одном экране могут одновременно сосуществовать изображения в не перекрывающихся зонах вывода, но в каждый конкретный момент выполнения программы действует только одно окно вывода.

Программа 8_01.bas

```
REM Демонстрация графических окон
SCREEN 12 :' Переход в графический режим
VIEW (10,10)-(210,210),2,1 : ' Объявление графического окна
LINE (0,100)-(200,100)
LINE (100,0)-(100,200)
SLEEP : ' Задержка до нажатия любой клавиши
VIEW SCREEN (310,10)-(510,210),4,14: ' Новое окно
LINE (310,110)-(510,110)
LINE (410,10)-(410,510)
SLEEP
END
```

В момент создания зоны вывода первого типа (`VIEW`) текущая точка устанавливается в центр окна, тогда как для области вывода второго типа (`VIEW SCREEN`) текущая точка переводится в центр экрана и может оказаться за пределами окна. С учетом последнего замечания построения в окне можно вести не только в абсолютных координатах, определяемых описанными выше способами, но и в относительных координатах. Последние задаются в приращениях относительно положения текущей точки и сопровождаются добавкой `STEP`:

Программа 8_02.bas

```
REM Построения в относительных координатах
SCREEN 12
VIEW (10,10)-(210,210),2,1
LINE STEP(0,0)-STEP(-20,0) : ' Отрезок белого цвета
COLOR 4 : ' Цвет рисования - красный
LINE STEP(0,0)-STEP(0,-20)
COLOR 2 : ' Цвет рисования - зеленый
VIEW SCREEN (310,10)-(510,210),4,14
LINE STEP(0,0)-STEP(-20,0)
END
```

В приведенном выше примере первый отрезок белого цвета начинается в центре области (нулевые смещения начальной точки отрезка означают совпадение с положением текущей точки). Конец отрезка смещен на 20 пикселов по горизонтали влево. В эту же точку перемещается `CP` после построения первого отрезка. Второй отрезок красного цвета начинается из конца первого отрезка

и продолжается вверх по вертикали на 20 пикселов (в направлении, противоположном оси `y`). Для второго окна текущая точка оказалась снаружи, поэтому горизонтальный отрезок зеленого цвета мы не увидим.

Программирование перемещений в целочисленных координатах пикселов далеко не всегда бывает удобным с точки зрения прикладной программы. Например, чтобы построить график синусоиды, ее амплитуду придется умножить на достаточно большое число, чтобы увеличить размах по оси `y`, принудительно увеличить значение функции, чтобы исключить отрицательные координаты точек, растянуть значение координаты `x` таким образом, чтобы на видимом участке оказались один-два более или менее красивых периода функции, и учесть специфику в направлении оси `y` (`Y = y - ymax`). Система QBasic берет все эти хлопоты на себя. Достаточно лишь после объявления области вывода указать пределы изменения координат (`x`,`y`) в отображаемых объектах с помощью оператора `WINDOW`:

```
WINDOW (XMIN,YMIN)-(XMAX,YMAX)
```

Построение одного периода синусоиды, например, может выглядеть следующим образом:

Программа 8_03.bas

```
REM Построение синусоиды
SCREEN 12
VIEW (10,10)-(410,210),2,1
LINE (400,100)-(0,100) ' Построение оси x и возврат CP в ее начало
WINDOW (0,-1)-(6.3,1)  ' Объявление пределов изменения функции
FOR X=.1 TO 6.3 STEP .1
  Y=SIN(X)          ' Вычисление значения функции
  LINE -(X,Y)       ' Проведение отрезка прямой из CP в (X,Y)
NEXT X
END
```

В пакете BGI область графического вывода устанавливается процедурой `setviewport`:

```
TC: setviewport(xmin,ymin,xmax,ymax,clip);
TP: SetViewPort(xmin,ymin,xmax,ymax,clip);
```

Экранные координаты точек (`xmin`,`ymin`) и (`xmax`,`ymax`) определяют, соответственно, положение левого верхнего и правого нижнего углов прямоугольной области вывода. Система координат в зоне вывода привязана своим началом к левому верхнему углу прямоугольника, и ее ось `y`, к сожалению, направлена вниз. Параметр `clip`, который может принимать только два значения (в Си — `0` или `не 0`, в Паскале — `true` или `false`), управляет режи-

мом отсечения графических изображений, выходящих своими частями за пределы области вывода. При `clip = 0` или `clip = false` отсечение не производится. Объявление окна графического вывода в TC и TP автоматически чистит окно и переводит текущую точку в начало координат. По умолчанию фон окна черный, а линии рисуются белым цветом.

Программа построения периода синусоиды на Паскале, например, может выглядеть следующим образом:

Программа 8_03.pas

```
program sin1;
uses graph;
var
  gd,gm,xe,ye,k:integer;
  x,y,dx:double;
begin
  gd:=0;
  InitGraph(gd,gm,'');
  dx:=0.1;
  SetViewPort(10,10,410,210,false);
  LineTo(0,200);        {Обводим границы установленного окна}
  LineTo(400,200);
  LineTo(400,0);
  LineTo(0,0);
{Проводим ось x и возвращаем CP в ее начало}
  Line(400,100,0,100);
  for k:=0 to 63 do
    begin
      x:=k*dx;
      y:=sin(x);
      xe:=round(x*400/6.28);
{Преобразуем программные координаты к экранным - xe, ye }
      ye:=round(100-y*100);
      LineTo(xe,ye)     {Соединяем предыдущую точку с новой}
    end;
  readln;
  closegraph;
end.
```

Для очистки окна вывода в системе QBasic используется оператор `CLS`, который заливает внутренность окна текущим цветом фона экрана (но не цветом фона, установленным при объявлении окна), зато сохраняет цветную окантовку окна. Оператор `CLS 2` чистит не только текущее окно, но и весь экран, переводя его в окно вывода по умолчанию.

Пакет BGI предлагает две процедуры без параметров — `clearviewport` и `cleardevice`. Первая из них заливает цветом фона область текущего окна вывода и возвращает `CP` в начало координат, вторая — чистит экран и объявляет его текущей зоной вывода.

Для определения максимальных размеров текущего окна вывода по каждой координате можно использовать функции `getmaxx` и `getmaxy`, не требующие параметров. Значениями этих функций удобно пользоваться, когда необходимо привязать то или иное изображение к фиксированному месту в окне вывода. Например, построение окружности, расположенной в центре зоны вывода, выполняется следующим образом:

```
Circle(GetMaxX div 2,GetMaxY div 2, 50);
```

Управление цветом

Процедуры воспроизведения элементарных геометрических объектов — точек, отрезков прямых, дуг окружностей или эллипсов, прямоугольников, — используют тот или иной цвет при окрашивании соответствующих пикселов. В системе QBasic цвет объекта может быть задан с помощью необязательного параметра в соответствующем графическом операторе. Если он явно не указан, то объект рисуется цветом переднего плана, установленным ранее в операторе `COLOR`:

```
COLOR cf,cb
```

В режимах 12 и 13 оператор `COLOR` допускает единственный параметр `cf`, определяющий программный цвет рисования. Его значением может быть любое число из интервала [0,15], по которому системные программы извлекут содержимое регистра палитры с номером `cf` и преобразуют его в физический цвет отображаемых на экране пикселов. Второй параметр `cb` допустим только в режиме 9 (EGA), он определяет цвет фона. В режимах 12 и 13 цвет фона устанавливается в момент объявления области вывода. По умолчанию им является черный цвет.

Приводимая ниже программа демонстрирует цветовую палитру режима 12, воспроизводя с задержкой в 1 с прямоугольники размером 100×75 пикселов, заливаемые последовательно цветами от 0 до 15.

Программа 8_04.bas

```
REM Демонстрация цветовой палитры
SCREEN 12
CLS
FOR Y=10 TO 310 STEP 100
FOR X=10 TO 460 STEP 150
  LINE (X-1,Y-1)-(X+101,Y+76),15,B
  LINE (X,Y)-(X+100,Y+75),COL,BF
  COL=COL+1
  SLEEP 1 : ' Задержка на 1 сек
NEXT X
NEXT Y
END
```

В BGI-пакете цвет переднего плана и цвет фона устанавливаются процедурами (функциями) `setcolor(cf)` и `setbkcolor(cb)`. В процедурах отображения геометрических объектов в системах Turbo C и Turbo Pascal цвет рисования явно не задается. Используется только тот цвет, который был установлен с помощью `setcolor`. Для тех, кому английский язык не чужд, наряду с числовыми значениями кода цветности BGI-пакет предлагает набор мнемонических констант (табл. 8.1).

Таблица 8.1. *Набор мнемонических констант BGI-пакета*

Код	Цвет	Turbo C	Turbo Pascal
0	Черный	BLACK	Black
1	Синий	BLUE	Blue
2	Зеленый	GREEN	Green
3	Бирюзовый	CYAN	Cyan
4	Красный	RED	Red
5	Малиновый	MAGENTA	Magenta
6	Коричневый	BROWN	Brown
7	Светло-серый	LIGHTGRAY	LightGray
8	Темно-серый	DARKGRAY	DarkGray
9	Светло-синий	LIGHTBLUE	LightBlue
10	Светло-зеленый	LIGHTGREEN	LightGreen
11	Светло-бирюзовый	LIGHTCYAN	LightCyan

Таблица 8.1 *(окончание)*

Код	Цвет	Turbo C	Turbo Pascal
12	Светло-красный	LIGHTRED	LightRed
13	Светло-малиновый	LIGHTMAGENTA	LightMagenta
14	Желтый	YELLOW	Yellow
15	Белый	WHITE	White

В некоторых руководствах вместо термина "малиновый" используют "фиолетовый", хотя в общепринятом смысле эти цветовые оттенки воспринимаются как разные. Вообще говоря, на разных видеосистемах приведенная палитра может иметь некоторые отклонения. Полным аналогом программы 8_04.bas является следующая программа на Си:

Программа 8_04.c

```
/* Демонстрация цветовой палитры */
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
  main()
{
  int x,y,col=0;
  int gd=0,gm;
  initgraph(&gd,&gm,"");
  for(y=10; y<=310; y+=100)
    for(x=10; x<=460; x+=150)
      {
        setfillstyle(1,col); /* Сплошная заливка цветом col */
        bar3d(x-1,y-1,x+101,y+76,0,1);
        col++;
        sleep(1); /* Задержка на 1 сек*/
      }
  getch();
  closegraph();
}
```

Кроме управления двумя основными цветами переднего и заднего планов, имеется возможность изменить содержимое одного или всех регистров па-

литры. В системе QBasic для этой цели используются операторы `PALETTE` и `PALETTE USING`:

```
PALETTE reg,col
PALETTE USING A%
```

Параметр `reg` задает номер модифицируемого регистра палитры, а значение `col` из диапазона [0,63] определяет новое содержимое указанного регистра. В случае групповой модификации всех регистров палитры их новые значения определяются байтами массива `A%`. Если в каком-то из этих байтов находится −1 (0×FF), то значение соответствующего регистра палитры не изменяется.

Функции `setpalette` и `setallpalette` в пакете BGI ничем не отличаются от описанных выше операторов:

```
SetPalette(reg,col);
SetAllPalette(A);
```

Работа с отдельными точками и растровыми изображениями

Вывод отдельных точек (пикселов) в программе имеет смысл, когда количество таких точек сравнительно невелико (от нескольких десятков до 2—3 тысяч). Если мы попытаемся заполнить весь экран путем вывода каждой точки, то эта процедура займет достаточно много времени. Для изменения кода цвета пиксела достаточно указать его координаты (оконные или экранные) и задать новый цвет.

В QBasic для этой цели используется один из двух операторов `PSET` или `PRESET`:

```
PSET x,y [,c]
PRESET x,y [,c]
```

Третий параметр является необязательным, но если он задан, то определяет новый код цвета. Если в этих операторах цвет не задавать, то оператор `PSET` использует цвет переднего плана, `PRESET` — цвет фона. Таким образом, один из них рисует точку, а второй — стирает ее на экране. Если очень надо, то программа может узнать цвет пиксела с заданными координатами. В QBasic для этой цели используется функция `POINT`:

```
c=POINT(x,y)
```

Аналогичные возможности присутствуют и в библиотеке BGI:

```
TC:  putpixel(x,y,c);          TP:  PutPixel(x,y,c);
     c=getpixel(x,y);               c:=GetPixel(x,y);
```

Гораздо больший интерес представляют процедуры группового копирования кодов цвета пикселов, заполняющих прямоугольную область экрана и образующих растровое изображение. Так как операция копирования между байтами видеопамяти и оперативной памяти выполняется достаточно быстро, то таким способом можно мгновенно обновить содержимое экрана. Правда, целиком скопировать или обновить содержимое экрана нельзя по совершенно тривиальной причине. В таком обмене должен участвовать массив, а в наших системах программирования размер массива не может превышать 64 Кбайт. Тогда как для запоминания кодов цвета всех пикселов экрана потребуется не менее чем 640×480×4/8 = 153 600 байт. Поэтому для полного обновления содержимого экрана необходимо не менее трех обращений к соответствующей процедуре.

В системе QBasic групповой обмен с видеопамятью осуществляют операторы `GET` и `PUT`:

```
GET (x1,y1)-(x2,y2), A%
PUT (x3,y3), A%, ovr
```

Координаты двух точек, как и всегда, определяют положение левого верхнего (`x1`,`y1`) и правого нижнего (`x2`,`y2`) углов копируемой прямоугольной области экрана. Коды цветности пикселов, попадающих в этот участок, переписываются в массив `A%`, который должен иметь достаточный объем. В самых первых элементах массива `A%` запоминаются габариты сохраняемой области экрана — число строк и число столбцов, вслед за которыми в каждый байт массива `A%` поступают коды цветности двух смежных пикселов. Если оказывается, что число пикселов в строке нечетно, то прихватывается код цветности еще одного лишнего пиксела, чтобы сохранить байтовую структуру данных.

Так как при копировании области экрана в оперативной памяти сохраняются не только цвета пикселов, но и размеры запоминаемой области, то в операторе `PUT` достаточно указать только одну точку на экране, начиная с которой будет размещаться восстанавливаемое изображение. Параметр `ovr` определяет способ взаимодействия кодов цветности налагаемых пикселов с их прежними цветовыми атрибутами. Он может принимать одно из следующих мнемонических значений:

- `PSET` — новый код цветности вытесняет предыдущий;
- `PRESET` — инвертированное значение нового кода цветности заменяет предыдущий код;
- `AND` — новый и старый коды цветности поразрядно логически умножаются;
- `OR` — новый и старый коды цветности поразрядно логически складываются;
- `XOR` — над новым и старым кодами выполняется операция "исключающее ИЛИ" (другими словами — выполняется поразрядное сложение по модулю 2).

Абсолютно те же функции в BGI-пакете выполняют процедуры `getimage` и `putimage`:

```
getimage(x1,y1,x2,y2,A);
putimage(x3,x4,A,ovr);
```

Единственное отличие заключается в мнемонике значений параметра `ovr`:

TC:	TP:	QBasic:
COPY_PUT	CopyPut	PSET
NOT_PUT	NotPut	PRESET
AND_PUT	AndPut	AND
OR_PUT	OrPut	OR
XOR_PUT	XorPut	XOR

В BGI-пакете предусмотрена еще одна функция, с помощью которой можно определить размер массива `A` в байтах до выполнения операции `getimage`:

```
size=imagesize(x1,y1,x2,y2);
```

Эта функция может оказаться полезной, если память для временного хранения растрового изображения динамически запрашивается и после использования возвращается.

Отрезки прямых и прямоугольники

В системе QBasic для построения отрезков прямых и прямоугольников используется один и тот же оператор `LINE` по той причине, что обе эти фигуры могут быть определены двумя точкам — началом и концом отрезка или координатами противоположных вершин в прямоугольнике. Более того, на этот же оператор в Бейсике возложили функцию отображения залитого прямоугольника:

```
LINE (x1,y1)-(x2,y2)[,[col][,mask]     :' Отрезок прямой
LINE (x1,y1)-(x2,y2)[,[col] B [,mask] :' Контур прямоугольника
LINE (x1,y1)-(x2,y2)[,[col] BF         :' Залитый прямоугольник
```

В приведенных выше форматах оператора `LINE` координаты точек заданы как абсолютные оконные или экранные координаты. Если построению предшествовало задание границ изменения координат (`WINDOW`), то координаты точек задаются в терминах физических единиц задачи.

Если координатам точек предшествует добавка `STEP`, то в скобках задаются приращения относительно позиции текущей точки. Координаты первой точки могут отсутствовать (`LINE – (x2,y2)...`), и тогда построение начнется с текущей точки.

Необязательный параметр `col` определяет цвет рисования, который может отличаться от цвета, установленного по оператору `COLOR`.

Если отрезок прямой или контуры прямоугольника проводятся сплошной линией, то параметр `mask` не задается. Его функция заключается в том, чтобы в последовательности из очередных 16 пикселов линии определить, какие пикселы будут окрашены цветом линии, а какие будут окрашены цветом фона, т. е. останутся невидимыми. Значение `mask` представляет собой 16-битовую шкалу, в которой единичные разряды соответствуют видимым пикселам линии, а нулевые — невидимым. Например, для построения штриховой линии, у которой длина штриха равна длине пробела и состоит из 8 пикселов, достаточно указать `mask = &HFF00`.

В BGI-пакете функции построения отрезков прямых, границ прямоугольников и залитых прямоугольников четко разделены:

- `moveto(x,y);` — перемещение CP в точку `(x,y)`;
- `moverel(dx,dy);` — перемещение CP в точку `(CP.x+dx,CP.y+dy)`;
- `lineto(x,y);` — соединение CP отрезком с точкой `(x,y)`;
- `linerel(dx,dy);` — соединение CP отрезком с точкой `(CP.x+dx,CP.y+dy)`;
- `line(x1,y1,x2,y2);` — соединение отрезком точек `(x1,y1)` и `(x2,y2)`;
- `rectangle(x1,y1,x2,y2);` — построение контура прямоугольника;
- `bar(x1,y1,x2,y2);` — построение залитого прямоугольника без обводки границ;
- `bar3d(x1,y1,x2,y2,d,b);` — построение параллелепипеда.

Перемещение `CP` (`moveto`, `moverel`) не оставляет следов на экране, иными словами — проводится невидимый отрезок. Это необходимо для перехода к очередной несвязной фигуре. Может показаться странным, что в QBasic в явном виде нет аналогичных операций. На самом деле, это не совсем так. Перевод `CP` в точку с заданными координатами (`x`,`y`) или смещение ее по известным приращениям без построения видимого отрезка осуществляется с помощью оператора `PRESET`:

```
PRESET (x,y)   или   PRESET STEP(dx,dy)
```

Однако, если до этого точка с координатами (`x`,`y`) была окрашена в цвет, отличный от цвета фона, то после оператора `PRESET` она будет перекрашена. Чтобы не затереть прежний цвет точки (`x`,`y`), можно предварительно опросить ее код цвета, а потом повторить его в одном из операторов `PSET` или `PRESET`:

```
col=POINT(x,y) :  PSET (x,y),col
```

или

```
PSET (x,y),POINT(x,y)
```

После построения видимых отрезков `CP` перемещается в конец отрезка. Построение прямоугольника также переводит `CP` в точку с координатами (`x2`,`y2`).

BGI-пакет располагает точно такой же возможностью по управлению видимостью пикселов линии с помощью 16-битовой маски. Но функция `setlinestyle`, определяющая стиль линии, несколько удобнее:

```
setlinestyle(style,mask,t);
```

Первый ее аргумент позволяет установить одну из нескольких заранее подготовленных системных масок, не перелагая на пользователя подбор соответствующего двоичного кода. Аргумент `style` — целое число из диапазона [0,4], которое удобно задавать в виде мнемонической константы:

TC: SOLID_LINE	TP: SolidLn	0	— сплошная линия
DOTTED_LINE	DottedLn	1	— пунктирная линия
CENTER_LINE	CenterLn	2	— штрих-пунктирная линия
DASHED_LINE	DashedLn	3	— штриховая линия
USERBIT_LINE	UserBitLn	4	— маска пользователя

Если ни одна из четырех системных масок вас не устраивает, то при `style = 4` в качестве маски используется значение параметра `mask`, подбираемое экспериментальным путем.

Последний параметр `t` может принимать только два значения 1 или 3, он задает толщину линии в пикселах. Две соответствующие мнемонические константы для Си — `NORM_WIDTH` и `THICK_WIDTH`, для Паскаля — `NormWidth` и `ThickWidth`.

В составе BGI-пакета присутствует еще одна функция, не имеющая аналога в QBasic. Она позволяет обвести границы многоугольника, заданного координатами точек его вершин:

```
drawpoly(k,xy);
```

Здесь `k` — количество точек, координаты которых хранятся в целочисленном массиве `xy` — `x1`, `y1`, `x2`, `y2`, ... Дополнительное удобство этой процедуры заключается в том, что, в случае необходимости, она сама осуществляет замыкание контура, соединяя последнюю точку массива с первой. В эту же точку перемещается и `CP` после построения многоугольника.

В системах программирования фирмы Borland некоторые трудности возникают при желании нарисовать черную линию на белом фоне. Дело в том, что черный цвет по умолчанию имеет нулевой программный код, и такой же нулевой код рассматривается в BGI-пакете как цвет фона. Поэтому попытка проделать нечто вроде:

```
setcolor(0);
setbkcolor(15);
line(0,0,100,100);
```

ни к какому результату не приводит. На белом фоне линия "рисуется" белым же цветом и поэтому не видна. В системе QBasic таких проблем просто не возникает. Попробуйте запустить программу:

Программа 8_05.bas

```
SCREEN 12
VIEW (0,0)-(100,100),15
LINE (0,0)-(100,100),0
END
```

В программах на Паскале или на Си приходится "обманывать" графическую систему следующим образом. Код нулевого цвета заносится в регистр палитры с номером, отличным от нуля. Затем содержимое этого регистра назначается в качестве цвета переднего плана.

Программа 8_05.pas

```
program black_white;
{ Построение черных линий на белом фоне }
uses graph;
var
  gd,gm,x,y:integer;
begin
  gd:=0;
  initgraph(gd,gm,'');
  setcolor(0);
  setbkcolor(15);
  line(0,0,100,100); {ничего не видно, т. к. рисуем цветом фона}
  readln;
  setpalette(1,0);
  setcolor(1);
  setbkcolor(15);
  line(0,0,100,100); {хорошая черная линия на белом фоне}
  readln;
  closegraph;
end.
```

Стирание ранее нарисованных отрезков можно осуществить путем повторного построения такого же отрезка в режиме `XOR_PUT`. BGI-пакет позволяет

на некоторое время установить один из двух режимов вывода — COPY_PUT или XOR_PUT с помощью процедуры setwritemode:

```
setwritemode(COPY_PUT);
setwritemode(XOR_PUT);
```

В первом режиме точки очередного отрезка или окружности будут затирать встречающиеся на их пути пикселы, а во втором — старый и новый цвета взаимодействующих точек будут поразрядно складываться по модулю 2.

Специфика построения залитых фигур рассматривается в одном из следующих разделов.

Окружности, эллипсы и дуги

В системе QBasic полные окружности, эллипсы, их дуги или соответствующие сектора воспроизводятся с помощью одного и того же оператора CIRCLE:

```
CIRCLE (x,y),R [,c]         : ' Полная окружность
CIRCLE (x,y),R,[c],a1,a2    : ' Дуга окружности
CIRCLE (x,y),R,[c],-a1,-a2 : ' Круговой сектор

CIRCLE (x,y),R,[c],,,k       : ' Полный эллипс
CIRCLE (x,y),R,[c],a1,a2,k   : ' Дуга эллипса
CIRCLE (x,y),R,[c],-a1,-a2,k : ' Эллиптический сектор
```

Координаты центра могут быть заданы как абсолютными, так и относительными (CIRCLE STEP(x,y)...) значениями. Для окружности R задает радиус, а для эллипса R и k участвуют в определении полуоси a (вдоль оси x) и полуоси b (вдоль оси y). При k < 1 a = R и b = k * R, а при k > 1 a = k * R и b = R. Проведите компьютерный эксперимент со следующей программой:

Программа 8_06.bas

```
REM Построение эллипсов
SCREEN 12
VIEW (0,0)-(400,400),,1
FOR X=10 TO 390 STEP 10
  FOR Y=10 TO 390 STEP 10
    LINE (X,10)-(X,390),7: ' построение вертикальных линий сетки
    LINE (10,Y)-(390,Y),7: ' построение горизонтальных линий сетки
  NEXT Y
NEXT X
```

```
LINE (0,200)-(400,200),14 : ' построение "оси" x
LINE (200,0)-(200,400),14 : ' построение "оси" y
CIRCLE (200,200),120,15,,,4/3 : ' белый эллипс
CIRCLE (200,200),120,14,,,3/4 : ' желтый эллипс
END
```

В квадрате со стороной 400 пикселов с шагом в 10 пикселов строится серая сетка и по центру зоны вывода проводятся желтые координатные оси. Затем строятся белый и желтый эллипсы с центрами в середине области графики. На этом рисунке можно легко определить размеры полуосей каждого из эллипсов.

В построении дуг принимают участие два угла `a1` и `a2`, задаваемые в радианах. Их абсолютные значения принадлежат интервалу $[0,2\times\pi]$. Углы отсчитываются от положительного направления оси `x` против часовой стрелки. Угол `a1` задает положение начального радиус-вектора, проведенного из центра в начало дуги, а угол `a2` — положение конечного радиус-вектора. Вообще говоря, это справедливо для дуг окружности. Для дуг эллипсов углы приходится подбирать, т. к. параметры оператора используются сначала для вычисления точек на дуге окружности, при трансформации которой в дугу эллипса углы начального и конечного радиус-векторов искажаются.

Если углы задаются со знаком минус, то соответствующие концы дуг соединяются с центром, образуя круговой или эллиптический сектор. Такие фигуры могут использоваться для построения круговых диаграмм. Приведенный ниже пример демонстрирует построение дуги и сектора, вписывающихся в один и тот же центральный угол эллипсов разного размера. Углы начального и конечного радиус-векторов заданы в радианах и, соответственно, равны 30 и 60 градусам.

Программа 8_07.bas

```
REM Построение дуги и сектора эллипса
SCREEN 12
VIEW (0,0)-(400,400)
LINE (0,200)-(400,200),7 :' построение "оси" x
LINE (200,0)-(200,400),7 :' построение "оси" y
CIRCLE (200,200),120,4,,,.75 :' полный эллипс
CIRCLE (200,200),90,4,,,.75  :' полный эллипс
CIRCLE (200,200),120,15,.5236,1.0472,.75 :' белая дуга
CIRCLE (200,200),90,14,-.5236,-1.0472,.75 :' желтый сектор
END
```

Четвертый необязательный параметр в операторе `CIRCLE` задает явный цвет соответствующей фигуры, который может отличаться от цвета переднего плана, установленного по оператору `COLOR`.

В BGI-пакете для построения каждой из описанных выше графических фигур используется отдельная процедура:

```
arc(x,y,a1,a2,r);          //дуга окружности
circle(x,y,r);             //окружность
ellipse(x,y,a1,a2,rx,ry); //дуга эллипса или полный эллипс
```

В отличие от QBasic положение начального и конечного радиус-векторов здесь задается целочисленными значениями углов `a1` и `a2` в градусах, которые могут быть как положительными, так и отрицательными. Второе отличие связано с явным заданием обеих полуосей эллипса `rx` и `ry`.

Закрашивание и заполнение замкнутых областей

Графическое изображение становится более красочным, когда на экране представлены не только контуры геометрических фигур, но и закрашенные или заштрихованные области. Выполнение такого рода операций осуществляется с помощью процедур заливки (англ. paint, fill). Подобно тому, как при построении линий задается точечный шаблон, управляющий воспроизведением пикселов, для площадных объектов существует точно такая же возможность задания двумерной маски, хранящейся в битовом массиве размером 8×8 (Си, Паскаль) или 8×8×k (QBasic, $1 \le k \le 8$). Такая маска, перемещаясь по горизонтали и вертикали, управляет окраской попадающих под ее биты пикселов. К сожалению, идеи такого управления, заложенные в системе QBasic и BGI-пакете, сильно отличаются.

Более естественными нам представляются алгоритмы заполнения площадных объектов, реализованные фирмой Borland. Для них характерно задание квадратной маски 8×8, управляющей окраской пикселов изображения, накрываемых этой маской. Если в соответствующем разряде маски находится единица, то расположенный под ним пиксел закрашивается в заранее установленный цвет. Пиксел экрана, попадающий под нулевой бит маски, окрашивается в цвет фона. После обработки очередной площадки изображения маска смещается на 8 пикселов вправо. Когда текущая полоска экрана исчерпана, маска перемещается в начало следующей полоски. Системная или пользовательская маска в сочетании с установленными цветами заливки и фона составляют то, что принято называть шаблоном (англ. pattern) заливки.

Для выбора шаблона заливки в BGI-пакете используются две функции (процедуры) — `setfillstyle` и `setfillpattern`:

```
setfillstyle(numpat,col);
setfillpattern(pattern,col);
```

Параметр `numpat` задается целым числом из диапазона [0,12] или значением соответствующей мнемонической константы (табл. 8.2). Он позволяет выбрать один из системных шаблонов (`numpat < 12`) или установить пользовательский шаблон (`numpat = 12`), описываемый с помощью функции `setfillpattern`. Параметр `col` задает код цвета, в который должны окрашиваться пикселы изображения, попадающие под единичные биты маски.

Таблица 8.2. *Шаблоны заливки в BGI-пакете*

Номер шаблона	Константа TC	Константа TP	Способ заполнения области
0	EMPTY_FILL	EmptyFill	Заливается цветом фона
1	SOLID_FILL	SolidFill	Заливается цветом col
2	LINE_FILL	LineFill	Штриховка горизонтальными линиями
3	LTSLASH_FILL	LtSlashFill	Тонкая штриховка под 45°
4	SLASH_FILL	SlashFill	Толстая штриховка под 45°
5	BKSLASH_FILL	BkSlashFill	Толстая штриховка под 135°
6	LTBKSLASH_FILL	LtBkSlashFill	Тонкая штриховка под 135°
7	HATCH_FILL	HatchFill	Двойная штриховка, 0° и 90°
8	XHATCH_FILL	XHatchFill	Двойная штриховка, 45° и 135°
9	INTERLEAVE_FILL	InterleaveFill	Короткие чередующиеся штрихи
10	WIDE_DOT_FILL	WideDotFill	Редкий точечный растр
11	CLOSE_DOT_FILL	CloseDotFill	Густой точечный растр
12	USER_FILL	UserFill	По шаблону пользователя

Наиболее простой вариант подбора системного шаблона предлагает следующая программа, которая перебирает их в цикле, останавливаясь после вывода очередного образца.

Программа 8_08.c

```
/* Системные шаблоны заполнения фигур */
#include <stdio.h>
```

```
#include <conio.h>
#include <graphics.h>
  main()
{
   int k,gd=0,gm;
   int col=14;   //желтый цвет для единичных пикселов шаблона
   initgraph(&gd,&gm,"");
   for(k=0; k<12;k++)
   {
     cleardevice();
     printf("\nномер шаблона=%d",k);
     setfillstyle(k,14);
     bar(300,100,400,200);
     getch();
   } Closegraph();
}
```

Если при выборе шаблона вы остановились на `numpat = 12`, то перед построением залитых фигур необходимо определить структуру нестандартного узора. В качестве параметра `pattern` в функции `fillpattern` может выступать любой 8-байтовый массив:

```
TC: char pat1[]={0xCC,0x33,0xCC,0x33,0xCC,0x33,0xCC,0x33};
    ......................
    setfillpattern(pat1,4);
TP: const
      pat1:FillPatternType=($CC,$33,$CC,$33,$CC,$33,$CC,$33);
    .......................
    SetFillPattern(pat1,4);
```

Тип данных `FillPatternType`, описанный в модуле `Graph`, представляет собой байтовый массив из 8 элементов — `array [1..8] of byte`.

В BGI-пакете предусмотрены пять процедур (функций) для построения заполненных геометрических фигур:

- `bar(x1,y1,x2,y2)` — залитый прямоугольник без контура;
- `bar3d(x1,y1,x2,y2,d,t)` — трехмерный столбик;
- `fillellipse(x,y,rx,ry)` — залитый эллипс или окружность;
- `fillpoly(n,xy)` — залитый многоугольник;
- `sector(x,y,a1,a2,rx,ry)` — залитый эллиптический сектор;
- `pieslice(x,y,a1,a2,r)` — залитый круговой сектор.

В трехмерном столбике первые четыре параметра определяют положение передней грани. Параметр `d` задает "глубину" столбика, и иногда его рекомендуют выбирать равным четверти ширины (`d = 0.25 * (x2 - x1)`). Последний аргумент в Си может принимать нулевое или ненулевое значение, а в Паскале — `true` или `false`. Если он отличен от `0` или равен `true`, то верхняя грань столбика ("крыша") рисуется. В противном случае столбик воспроизводится без крыши, что дает возможность поставить над ним еще один столбик и не заботиться об удалении невидимых линий. В трехмерном столбике заливается только передняя грань.

Процедура `bar3d`, как правило, используется для воспроизведения объемных диаграмм. Однако в ней можно задать нулевую глубину столбика (`d = 0`) и тогда результат ее работы заменяет выполнение двух последовательных процедур:

```
bar(11,11,99,99);
rectangle(10,10,100,100);
```

Первая из них отображает залитую внутренность прямоугольника, а вторая — обводит его границу.

Параметры остальных процедур довольно подробно рассматривались в предыдущих разделах.

Недавно одному из авторов пришлось разрабатывать программные средства выделения замкнутых областей в цифровых электронных картах. Кроме использования самых простых идей — выделение цветом и штриховками, — наиболее плодотворный результат при выводе на дисплей был достигнут регулярными заполнителями с помощью заливочных шаблонов. Самая богатая коллекция таких шаблонов, которая была известна нам по литературе, насчитывала 28 образцов [13]. Однако по своему разнообразию этот набор показался нам недостаточно выразительным. Кроме того, в нем отсутствовала какая-либо классификация шаблонов. Пришлось потратить несколько дней на эксперименты с редактором образов в системе Builder C++, после чего на свет появилось более сотни шаблонов. Их мы и представляем нашим читателям в надежде, что наиболее любознательные сумеют построить еще не один десяток приятных узоров.

Программа 8_09.c

```
/* Пользовательские шаблоны заполнения фигур
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#define Nmax 111
  main()
```

```
{
  int k,gd=0,gm;
  unsigned char pattern[Nmax][8] ={
//Массив заливочных шаблонов 8x8
//==== Сплошная заливка
//шаблон 0 = очистка цветом фона
  {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
//шаблон 1 = сплошная заливка установленным цветом
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF},
//==== Штриховка сплошными линиями
//= по горизонтали
//шаблон 2 = горизонтали (линия — 1 пиксел, зазор — 7)
  {0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
//шаблон 3 = горизонтали (2:6)
  {0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00},
//шаблон 4 = горизонтали (3:5)
  {0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00},
//шаблон 5 = горизонтали (4:4)
  {0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00},
//шаблон 6 = горизонтали (5:3)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00},
//шаблон 7 = горизонтали (6:2)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00},
//шаблон 8 = горизонтали (7:1)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00},
//шаблон 9 = горизонтали (1:1)
  {0xFF,0x00,0xFF,0x00,0xFF,0x00,0xFF,0x00},
//шаблон 10 = горизонтали (1:3)
  {0xFF,0x00,0x00,0x00,0xFF,0x00,0x00,0x00},
//шаблон 11 = горизонтали (3:1)
  {0xFF,0xFF,0xFF,0x00,0xFF,0xFF,0xFF,0x00},
//шаблон 12 = горизонтали (2:2)
  {0xFF,0xFF,0x00,0x00,0xFF,0xFF,0x00,0x00},
//= по вертикали
//шаблон 13 = вертикали (1:7)
  {0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80},
//шаблон 14 = вертикаль (1:3)
  {0x88,0x88,0x88,0x88,0x88,0x88,0x88,0x88},
//шаблон 15 = вертикаль (2:6)
```

```
  {0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0},
//шаблон 16 = вертикаль (3:5)
  {0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0},
//шаблон 17 = вертикаль (4:4)
  {0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0,0xF0},
//шаблон 18 = вертикали (5:3)
  {0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8},
//шаблон 19 = вертикали (6:2)
  {0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC},
//шаблон 20 = вертикали (7:1)
  {0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE},
//шаблон 21 = вертикали (1:1)
  {0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA,0xAA},
//шаблон 22 = вертикали (3:1)
  {0xEE,0xEE,0xEE,0xEE,0xEE,0xEE,0xEE,0xEE},
//шаблон 23 = вертикали (2:2)
  {0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC},
//==== прямоугольная двойная штриховка
//шаблон 24 = сетка (1:7,1:7)
  {0xFF,0x80,0x80,0x80,0x80,0x80,0x80,0x80},
//шаблон 25 = сетка (2:6,2:6)
  {0xFF,0xFF,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0},
//шаблон 26 = сетка (3:5,3:5)
  {0xFF,0xFF,0xFF,0xE0,0xE0,0xE0,0xE0,0xE0},
//шаблон 27 = сетка (4:4,4:4)
  {0xFF,0xFF,0xFF,0xFF,0xF0,0xF0,0xF0,0xF0},
//шаблон 28 = сетка (5:3,5:3)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xF8,0xF8,0xF8},
//шаблон 29 = сетка (6:2,6:2)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0xFC},
//шаблон 30 = сетка (7:1,7:1)
  {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE},
//шаблон 31 = сетка (2:2,2:2)
  {0xFF,0xFF,0x99,0x99,0xFF,0xFF,0x99,0x99},
//==== штриховка по диагонали
//шаблон 32 = справа налево (1:10)
  {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80},
//шаблон 33 = справа налево (3:7)
  {0x83,0x07,0x0E,0x1C,0x38,0x70,0xE0,0xC1},
```

```
//шаблон 34 = справа налево (6:3)
  {0xC7,0x8F,0x1F,0x3E,0x7C,0xF8,0xF1,0xE3},
//шаблон 35 = слева направо (1:10)
  {0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01},
//шаблон 36 = слева направо (3:7)
  {0xC1,0xE0,0x70,0x38,0x1C,0x0E,0x07,0x83},
//шаблон 37 = слева направо (6:3)
  {0xE3,0xF1,0xF8,0x7C,0x3E,0x1F,0x8F,0xC7},
//==== косоугольная двойная штриховка
//шаблон 38 = штриховка (1:6,1:6)
  {0x81,0x42,0x24,0x18,0x18,0x24,0x42,0x81},
//==== Штриховка штриховыми линиями
//= по горизонтали
//шаблон 39 = штрихи (4*1:4,4) в шахматном порядке
  {0xF0,0x00,0x00,0x00,0x0F,0x00,0x00,0x00},
//шаблон 40 = штрихи (5*1:3,4) в шахматном порядке
  {0xF8,0x00,0x00,0x00,0x1F,0x00,0x00,0x00},
//шаблон 41 = штрихи (4*2:4,4) в шахматном порядке
  {0xF0,0xF0,0x00,0x00,0x0F,0x0F,0x00,0x00},
//шаблон 42 = штрихи (5*2:3,4) в шахматном порядке
  {0xF8,0xF8,0x00,0x00,0x1F,0x1F,0x00,0x00},
//шаблон 43 = редкие штрихи (4*1:4,7)
  {0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
//шаблон 44 = сплошная и штриховая линии
  {0xFF,0x00,0x00,0x00,0xF0,0x00,0x00,0x00},
//= по вертикали
//шаблон 45 = редкие штрихи (4*1:4,7)
  {0x80,0x80,0x80,0x80,0x00,0x00,0x00,0x00},
//шаблон 46 = штрихи (4*1:4,4) в шахматном порядке
  {0x80,0x80,0x80,0x80,0x08,0x08,0x08,0x08},
//==== Волнообразные линии
//= по горизонтали
//шаблон 47 = тонкие с небольшой амплитудой
  {0xF0,0x0F,0x00,0x00,0x00,0x00,0x00,0x00},
//шаблон 48 = толстые с небольшой амплитудой
  {0xF0,0xFF,0x0F,0x00,0x00,0x00,0x00,0x00},
//шаблон 49 = тонкие с большой амплитудой
  {0x81,0x42,0x24,0x18,0x00,0x00,0x00,0x00},
//шаблон 50 = толстые с большой амплитудой
```

```
  {0x81,0xC3,0x66,0x3C,0x18,0x00,0x00,0x00},
//= по диагонали
//шаблон 51 = слева направо
  {0x10,0x10,0x10,0xF0,0x01,0x01,0x01,0x0F},
//==== Растровые сетки
//шаблон 52 = густые точки в шахматном порядке (1:1,1)
  {0x00,0x55,0x00,0xAA,0x00,0x55,0x00,0xAA},
//шаблон 53 = густые точки в шахматном порядке (1:3,2)
  {0x00,0x44,0x00,0x11,0x00,0x44,0x00,0x11},
//шаблон 54 = мелкие редкие точки в шахматном порядке
  {0x22,0x00,0x00,0x00,0x88,0x00,0x00,0x00},
//шаблон 55 = средние точки в шахматном порядке
  {0x00,0x66,0x66,0x00,0x00,0x99,0x99,0x00},
//шаблон 56 = средние редкие точки в шахматном порядке
  {0x00,0x00,0x30,0x30,0x00,0x00,0x03,0x03},
//шаблон 57 = средние очень редкие точки в шахматном порядке
  {0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00},
//шаблон 58 = крупные точки
  {0x00,0x00,0x3C,0x3C,0x3C,0x3C,0x00,0x00},
//шаблон 59 = густая шахматная доска
  {0xCC,0x33,0xCC,0x33,0xCC,0x33,0xCC,0x33},
//шаблон 60 = очень густая шахматная доска
  {0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55},
//==== Фигуры в узлах прямоугольной сетки
//шаблон 61 = крестики
  {0x10,0x10,0x7C,0x10,0x10,0x00,0x00,0x00},
//шаблон 62 = пунктирные крестики
  {0x10,0x00,0x54,0x00,0x10,0x00,0x00,0x00},
//шаблон 63 = жирные кресты
  {0x00,0x00,0x18,0x3C,0x3C,0x18,0x00,0x00},
//шаблон 64 = сетка из больших ромбиков
  {0x00,0x18,0x3C,0x7E,0x7E,0x3C,0x18,0x00},
//шаблон 65 = сетка из больших квадратов
  {0x00,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x00},
//шаблон 66 = залитые квадраты 5x5
  {0xF8,0xF8,0xF8,0xF8,0xF8,0x00,0x00,0x00},
//шаблон 67 = залитые квадраты 6x6
  {0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0x00,0x00},
//шаблон 68 = залитые квадраты 7x7
```

```
  {0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0x00},
//шаблон 69 = отдельные большие квадратики
  {0x00,0x7E,0x42,0x42,0x42,0x42,0x7E,0x00},
//шаблон 70 = отдельные маленькие квадраты
  {0x00,0x00,0x3C,0x24,0x24,0x3C,0x00,0x00},
//шаблон 71 = квадратики с точками в центре
  {0x00,0x7E,0x42,0x5A,0x5A,0x42,0x7E,0x00},
//шаблон 72 = символ "x"
  {0x00,0x42,0x24,0x18,0x18,0x24,0x42,0x00},
//шаблон 73 = символ "y"
  {0x00,0x44,0x28,0x10,0x10,0x10,0x00,0x00},
//шаблон 74 = галочки
  {0x00,0x42,0x24,0x18,0x00,0x00,0x00,0x00},
//шаблон 75 = перевернутые галочки
{0x00,0x00,0x00,0x18,0x24,0x42,0x00,0x00},
//шаблон 76 = тонкие стрелки вверх
  {0x10,0x38,0x54,0x10,0x10,0x10,0x00,0x00},
//шаблон 77 = тонкие стрелки вниз
  {0x10,0x10,0x10,0x54,0x38,0x10,0x00,0x00},
//шаблон 78 = тонкие стрелки влево
  {0x00,0x20,0x40,0xFC,0x40,0x20,0x00,0x00},
//шаблон 79 = тонкие стрелки вправо
  {0x00,0x10,0x08,0xFC,0x08,0x10,0x00,0x00},
//шаблон 80 = жирные стрелки влево
  {0x00,0x08,0x18,0x3F,0x3F,0x18,0x08,0x00},
//шаблон 81 = двойные стрелки по вертикали
  {0x20,0x70,0xA8,0x22,0x22,0x8A,0x07,0x02},
//шаблон 82 = двойные стрелки по горизонтали
  {0x24,0x02,0x1F,0x02,0x24,0x40,0xF8,0x40},
//шаблон 83 = контуры ромбиков по сетке
  {0x00,0x10,0x28,0x44,0x28,0x10,0x00,0x00},
//шаблон 84 = залитые ромбики и точки
  {0x81,0x18,0x3C,0x7E,0x7E,0x3C,0x18,0x81},
//шаблон 85 = залитые ромбики по вертикали
  {0x18,0x3C,0x3C,0x7E,0x7E,0x3C,0x3C,0x18},
//шаблон 86 = кружочки на сетке (d=6)
  {0x00,0x3C,0x42,0x42,0x42,0x42,0x3C,0x00},
//шаблон 87 = кружочки на сетке  (d=5)
  {0x44,0x38,0x00,0x00,0x00,0x38,0x44,0x44},
```

```
//шаблон 88 = не залитые квадратики
  {0x80,0x7F,0x41,0x41,0x41,0x41,0x41,0x7F},
//шаблон 89 = точечная сетка с точкам в узлах
  {0x08,0x00,0x08,0x5D,0x08,0x00,0x08,0x00},
//==== Фигуры в шахматном порядке
//шаблон 90 = ромбики в шахматном порядке
  {0x00,0x18,0x3C,0x7E,0x7E,0x3C,0x18,0x00},
//шаблон 91 = густые мелкие квадратики в шахматном порядке
  {0xF0,0xF0,0xF0,0xF0,0x0F,0x0F,0x0F,0x0F},
//шаблон 92 = более крупные квадраты в шахматном порядке
  {0xCC,0x33,0xCC,0x33,0xCC,0x33,0xCC,0x33},
//шаблон 93 = контуры квадратиков в шахматном порядке
  {0x42,0xC3,0x3C,0x24,0x24,0x3C,0xC3,0x42},
//==== Узоры
//шаблон 94 = залитые кирпичики по горизонтали
  {0x00,0xFB,0xFB,0xFB,0x00,0xDF,0xDF,0xDF},
//шаблон 95 = не залитые кирпичики по горизонтали
  {0x00,0xFB,0x0A,0xFB,0x00,0xDF,0x50,0xDF},
//шаблон 96 = залитые кирпичики по вертикали
  {0xEE,0xEE,0x0E,0xEE,0xEE,0xE0,0xEE,0xEE},
//шаблон 97 = не залитые кирпичи по диагонали
  {0x01,0x82,0x44,0x28,0x10,0x20,0x40,0x80},
//шаблон 98 = не залитые кирпичи по диагонали
  {0x80,0x41,0x22,0x14,0x08,0x04,0x02,0x01},
//шаблон 99 = волнистый узор
  {0x94,0x84,0x48,0x30,0x00,0xC1,0x22,0x14},
//шаблон 100 = спиральный узор
  {0xFF,0x01,0x7D,0x45,0x5D,0x41,0x7F,0x00},
//шаблон 101 = узор 1
  {0x00,0xEC,0x2A,0x2A,0x2A,0x2A,0xEC,0x00},
//шаблон 102 = узор 2
  {0x00,0x00,0x7E,0x42,0x7E,0x42,0x7E,0x42},
//шаблон 103 = узор 3
  {0x00,0x5D,0x3E,0x6D,0x7F,0x63,0x36,0xF0},
//шаблон 104 = узор 4
  {0x92,0x24,0x49,0x92,0x24,0x49,0x92,0x24},
//шаблон 105 = узор 5
  {0xB1,0x22,0x14,0x14,0x22,0x91,0x48,0x24},
//шаблон 106 = узор 6
```

```
  {0x02,0x91,0x68,0x08,0x10,0x16,0x89,0x40},
//шаблон 107 = узор 7
  {0xC3,0x42,0x5A,0x7E,0x7E,0x5A,0x42,0xC3},
//шаблон 108 = узор 8
  {0x03,0x18,0x60,0x80,0x80,0x40,0x20,0x18},
//шаблон 109 = узор 9
  {0xBF,0xA0,0xA0,0xBF,0xFB,0x0A,0x0A,0xFB},
//шаблон 110 = узор 10
  {0xFF,0x81,0xBD,0xA5,0xA5,0xBD,0x81,0xFF},
//шаблон 111 = узор 11
  {0xFF,0x81,0x81,0x99,0x99,0x81,0x81,0xFF}};

initgraph(&gd,&gm,"");
   for(k=0;k<Nmax;k++)
   {
     cleardevice();
     gotoxy(1,1);
     printf("\nномер шаблона=%d",k);
     setfillpattern(&pattern[k][0],4);
     setfillstyle(12, 14);
     bar3d(240,40,440,240,16,1);
     getch();
   }
}
```

Одной из самых интересных процедур заливки является функция `floodfill`:

```
floodfill(x,y,cg);
```

Работает эта процедура с замкнутой областью, внутренней или внешней части которой принадлежит точка с координатами (`x,y`). Параметр `cg` задает цвет пикселов, составляющих границу области. Заливке подвергается та часть области, которой принадлежит точка (`x,y`). При этом предполагается, что в заливаемой области не встречаются пикселы цвета `cg`. Если в границе области будет обнаружен микроскопический разрыв хотя бы в один пиксел, то узор-заполнитель обязательно просочится через него и заливка распространится как на внутреннюю, так и на внешнюю части области. При заливке цвет границы сохраняется. Однако в случае совпадения цвета границы с цветом окраски единичных пикселов шаблона получится область без ярко выраженной границы. Повторная перезаливка такой области уже невозможна.

В системе QBasic мы уже демонстрировали возможность построения залитого прямоугольника с помощью оператора `LINE...BF`. Заполнение его внут-

ренности ограничено сплошной окраской пикселов в заданный цвет. Все остальные возможности по построению закрашенных или заштрихованных фигур сосредоточены в операторе `PAINT`, пользоваться которым не так уж и легко. Оператор `PAINT` допускает несколько модификаций:

```
PAINT (x,y),[col],cg
PAINT (x,y),M1$,cg
PAINT (x,y),M1$,cg,M2$
```

Первый формат оператора очень напоминает функцию `floodfill`, заливающую внутреннюю или внешнюю часть области заданным цветом `col` или цветом переднего плана, ранее установленным по оператору `COLOR`.

Второй формат отличается от первого только тем, что цвет и узор заливки задаются с точностью до бита в строке `M1$`, длина которой может достигать 64 байт. Первые четыре байта строки `M1$` определяют цветовые атрибуты восьми смежных пикселов на экране. При этом в первом байте такой четверки сосредоточены биты управления синим цветом, во втором байте — биты управления зеленым цветом, в третьем байте — биты управления красным цветом, а в последнем байте — биты управления повышенной яркостью. Четыре следующих байта строки `M1$` определяют аналогичные атрибуты восьми смежных пикселов следующей строки и т. д. В полном объеме строка-шаблон позволяет задать цветовые атрибуты пикселов, образующих на экране прямоугольник размером 8×16.

Попробуем подобрать узор заливки, при котором область покрывается красными штрихами длиной по четыре пиксела с шахматным расположением штрихов между строками. Очевидно, что первые четыре байта строки-шаблона могут быть получены в виде суммы:

```
M1$=CHR$(&H0)+CHR$(&H0)+CHR$(&HF0)+CHR$(&HFF)
```

Первый байт подавляет биты синего цвета у всех восьми пикселов, второй — биты зеленого, третий байт формирует у первых четырех пикселов единичные биты красного цвета, сохраняя у оставшихся четырех нулевые разряды. Последний байт заносит по единице в бит яркости каждого пиксела, превращая черный цвет фона в темно-серый.

Три следующие четверки шаблона могут состоять из одинаковых наборов вида:

```
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
```

Пятая строка шаблона должна изменять расположение красного штриха:

```
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H0F)+CHR$(&HFF)
```

Наконец, три последних четверки должны повторять зазор между строками, описанный выше:

```
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
M1$=M1$+CHR$(&H0)+CHR$(&H0)+CHR$(&H00)+CHR$(&HFF)
```

Окончательный вариант программы, в которой можно поварьировать строку заливочного шаблона, приведен ниже.

Программа 8_10.bas

```
REM Красные штрихи в шахматном порядке
SCREEN 12
I1$=CHR$(&HFF):' байт с битами повышенной интенсивности
B0$=CHR$(&H0):' байт с нулевыми битами синего цвета
G0$=CHR$(&H0):' байт с нулевыми битами зеленого цвета
R0$=CHR$(&H0):' байт с нулевыми битами красного цвета
R40$=CHR$(&HF0):' байт с битами красного-черного
R04$=CHR$(&HF):' байт с битами черного-красного
M1$=M1$+B0$+G0$+R40$+I1$: M1$=M1$+B0$+G0$+R0$+I1$
M1$=M1$+B0$+G0$+R0$+I1$ : M1$=M1$+B0$+G0$+R0$+I1$
M1$=M1$+B0$+G0$+R04$+I1$: M1$=M1$+B0$+G0$+R0$+I1$
M1$=M1$+B0$+G0$+R0$+I1$ : M1$=M1$+B0$+G0$+R0$+I1$
LINE (100,100)-(200,200),,B
PAINT (150,150),M1$,15
END
```

Сравнительно несложная модификация этой программы позволяет изобразить фрагмент кирпичной стены:

Программа 8_11.bas

```
' Построение кирпичной стены
SCREEN 12
I1$=CHR$(&HFF):' байт с битами повышенной интенсивности
B0$=CHR$(&H0) :' байт с нулевыми битами синего цвета
G0$=CHR$(&H0) :' байт с нулевыми битами зеленого цвета
R8$=CHR$(&HFF):' байт с единичными битами красного цвета
RL$=CHR$(&H40):' красный бит слева
RR$=CHR$(&H2) :' красный бит справа
M1$=M1$+B0$+G0$+R8$+I1$: M1$=M1$+B0$+G0$+RL$+I1$
```

```
M1$=M1$+B0$+G0$+RL$+I1$: M1$=M1$+B0$+G0$+RL$+I1$
M1$=M1$+B0$+G0$+R8$+I1$: M1$=M1$+B0$+G0$+RR$+I1$
M1$=M1$+B0$+G0$+RR$+I1$: M1$=M1$+B0$+G0$+RR$+I1$
LINE (100,100)-(200,200),,B
PAINT (150,150),M1$,15
END
```

Заливка площадных фигур "прозрачными" шаблонами

При заполнении замкнутых областей тем или иным узором изменению цвета подвергаются все пикселы, попадающие под биты шаблона. Под "прозрачным" шаблоном мы будем понимать такой вариант заполнения, при котором единичные пикселы шаблона тем или иным способом взаимодействуют с пикселами изображения, а нулевые биты шаблона не изменяют цвет соответствующих элементов рисунка. Таким образом, в шаблоне могут быть созданы участки, сквозь которые просвечивает прежний рисунок или его фон.

Идея создания такого эффекта довольно проста. Из прямоугольной области черного экрана (цвет рисования по умолчанию равен 15, а цвет фона равен 0), на которую предстоит наложить прозрачный шаблон, логически вырезаются и запоминаются два битовых массива. Один из них формируется путем обычного наложения шаблона с его будущим цветом, а второй — также путем обычного наложения шаблона, получающегося в результате инвертирования битов исходного шаблона. Цвет такого "антишаблона" устанавливается равным 15 (в двоичном представлении такой цвет имеет код 1111). Второй массив предназначен для сохранения кодов цвета тех пикселов изображения, которые попадут под нулевые биты исходного шаблона, т. е. для прорезания в изображении траектории, образуемой единичными битами шаблона. Выполняется такое вырезание путем наложения "антимассива" на изображение по операции `AND`. На расчищенное таким образом место по операции `XOR` наносится изображение исходного шаблона.

В приведенной ниже программе область изображения представлена квадратом (0,0,300,300), закрашенным цветом `fon`, который выбирается случайным образом. В точках с координатами (60,60) и (240,240) проводятся красные окружности радиусом в 30 пикселов. "Прозрачный" и обычный шаблоны, покрывающие квадратные области размером 100×100, своими центрами совпадают с центрами окружностей. Их узор состоит из пилообразных линий цвета `c1`, а цвет фоновых пикселов — черный (по умолчанию).

Результат работы программы лучше увидеть своими глазами. После вывода очередного изображения программа ждет нажатия любой клавиши и прекращает работу, опознав код клавиши <Esc>.

Программа 8_12.c

```
/* "Прозрачный шаблон"*/
#include <graphics.h>
#include <stdlib.h>
#include <alloc.h>
  main()
{
  int gd=0,gm;
  int c1,fon;
  char q,
  pat1[]={0x81,0x42,0x24,0x18,0x00,0x00,0x00,0x00},
  pat2[]={0x7E,0xBD,0xDB,0xE7,0xFF,0xFF,0xFF,0xFF};
  int xy1[]={0,0,300,0,300,300,0,300};
  long k;
  char far *p1,*p2;
  initgraph(&gd,&gm,"");
  k=imagesize(10,10,110,110);
  p1=farmalloc(k); /* Запрос памяти */
  p2=farmalloc(k);

m:
  fon=random(15)+1;
m1:c1=random(15)+1;
  if((c1==fon)) goto m1;
/*Запоминание битового образа шаблона в заливаемой области*/
  setfillstyle(12,0);
  setfillpattern(pat1,c1);
  bar(10,10,110,110);
  getimage(10,10,110,110,p1);
/*Запоминание битового образа антишаблона */
  setfillstyle(12,0);
  setfillpattern(pat2,15);
  bar(10,10,110,110);
  getimage(10,10,110,110,p2);
/*Формирование изображения */
  setfillstyle(1,fon);
  fillpoly(4,xy1);
  setcolor(4);
```

```
  circle(60,60,30);
/* Вырезание профиля шаблона */
  putimage(10,10,p2,AND_PUT);
/*Вклеивание "прозрачного" шаблона*/
  putimage(10,10,p1,XOR_PUT);
/*Заливка обычным непрозрачным шаблоном*/
  setcolor(4);
  circle(240,240,30);
  setfillstyle(12,0);
  setfillpattern(pat1,c1);
  bar(190,190,290,290);
/*Ожидание нажатия клавиши. Выход по нажатию Esc*/
  q=getch();
  if(q != 0x1b) goto m;
  closegraph();
}
```

Текстовые сообщения в графическом режиме

Возможностью вывода пояснительных подписей, сопровождающих графические изображения, располагает каждая из рассматриваемых систем. Однако в системе QBasic отсутствуют средства по управлению шрифтами, масштабированию и точному (до пиксела) позиционированию текстов. В графическом режиме QBasic разрешает использование обычного оператора `PRINT`, который выводит тексты по правилам текстового режима.

Гораздо более богатыми возможностями располагает BGI-пакет, предоставляющий в распоряжение пользователя набор различных шрифтов и средства по управлению ими.

Собственно вывод текстовых сообщений осуществляется двумя процедурами (функциями):

```
outtext(сообщение);
outtextxy(x,y,сообщение);
```

В первом случае место расположения текста на экране задается позицией `CP`, которая после вывода перемещается за последний символ сообщения. Во втором случае точка привязки текста явно задается координатами в окне вывода. После работы процедуры `outtextxy` позиция `CP` не изменяется.

Существует девять способов расположения точки привязки текста относительно габаритного прямоугольника, окаймляющего выводимое сообщение (рис. 8.1).

```
1---------2---------3
|         |         |
4---------5---------6
|         |         |
7---------8---------9
```

Рис. 8.1. Расположение возможных точек привязки текста

По каждой из координат можно выбрать нижнюю, среднюю, верхнюю, левую или правую оси, пересечением которых и является точка привязки выводимого текста. По умолчанию в качестве точки привязки считается левый верхний угол габаритного прямоугольника. Выбор любой другой точки привязки осуществляется с помощью процедуры `settextjustify`:

```
SetTextJustify(horiz,vert);
```

Каждый из ее параметров может принимать одно из трех значений:

- `0` — снизу (`BOTTOM_TEXT`) или слева (`LEFT_TEXT`);
- `1` — по центру (`CENTER_TEXT`);
- `2` — сверху (`TOP_TEXT`) или справа (`RIGHT_TEXT`).

Учтите, что если какая-либо часть отображаемой буквы выходит за пределы экрана, то в растровом шрифте она вообще не рисуется, а в векторном рисуется либо ее часть, либо буква целиком, если режим отсечения заблокирован.

В состав базовой поставки Borland-систем программирования входит 11 шрифтов, среди которых шрифт по умолчанию (`DEFAULT_FONT`) относится к растровым, а остальные — к векторным. Символы растрового шрифта формируются из точек, образованных единичными битами в мини-растре 8×8 пикселов. Для организации зазоров между символами и строками в этом мини-растре правый столбец и нижняя строка оставляются пустыми. По мере увеличения размеров букв составляющие их точки превращаются в заметные квадратики, что существенно ухудшает контуры символов. Векторные шрифты представлены набором отрезков, концы которых располагаются в узлах растровой сетки экрана. При увеличении размеров букв длины отрезков, образующих контур, увеличиваются, сохраняя пропорции и не ухудшая качество изображения.

Векторные шрифты обеспечивают довольно разнообразное представление алфавита. Самые примитивные из них (например, `SMALL_FONT`) формируют контур в одну линию, и при увеличении таких букв их толщина не меняется. Более красивые (например, `SANS_SERIF_FONT`) используют двухконтурные модели букв, при увеличении которых до определенного размера толщина контура заметно увеличивается. Но при большем увеличении между внутренним и внешним контурами начинают проглядывать фоновые пикселы. Наиболее качественные шрифты (например, `TRIPLEX_FONT` или `GOTHIC_FONT`) для

создания разнотолщинных элементов контура используют до трех векторов на линию (рис. 8.2).

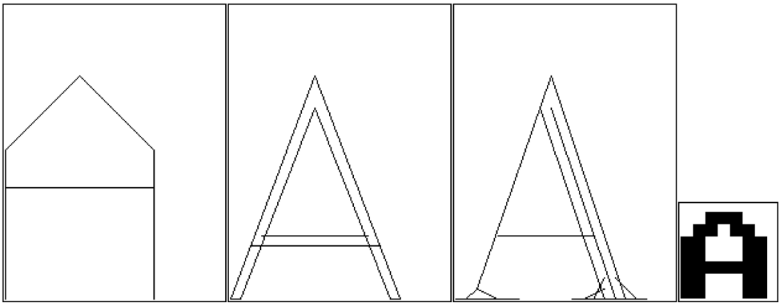


Рис. 8.2. Увеличенное изображение буквы в разных шрифтах

Векторные шрифты, входящие в состав фирменной поставки BC 3.1 и TP 7.0, вместо контуров букв русского алфавита содержат экзотические символы европейских шрифтов (буквы с умляутами из немецкого, особые буквы чешского и польского алфавитов и т. п.). Однако наши умельцы давным-давно модернизировали фирменные шрифты, пополнив их русскими буквами в соответствии с 866-й кодовой страницей. Найти русифицированные файлы с расширением chr особого труда не составляет.

Назначение шрифта, которым программа собирается выводить текстовые сообщения, производится с помощью процедуры `settextstyle`:

```
SetTextStyle(font,dir,size);
```

Параметр `font` определяет номер шрифта — число из диапазона от 0 до 10 — и может быть задан одной из мнемонических констант (табл. 8.3).

Таблица 8.3. *Мнемонические константы шрифтов*

Номер шрифта	Константа TC	Константа TP	Пояснение
0	`DEFAULT_FONT`	`DefaultFont`	Растровый, 8x8
1	`TRIPLEX_FONT`	`TriplexFont`	Трехобводный с засечками
2	`SMALL_FONT`	`SmallFont`	Однообводный, грубый
3	`SANS_SERIF_FONT`	`SansSerifFont`	Двухобводный, равнотолщинный
4	`GOTHIC_FONT`	`GothicFont`	Готический, трехобводный
5	`SCRIPT_FONT`	Мнемоника отсутствует	Рукописный, однообводный
6	`SIMPLEX_FONT`	Мнемоника отсутствует	Однообводный

7	TRIPLEX_SCR_FONT	Мнемоника отсутствует	Курсив, трехобводный

Таблица 8.3 *(окончание)*

Номер шрифта	Константа ТС	Константа ТР	Пояснение
8	COMPLEX_FONT	Мнемоника отсутствует	Двухобводный, разнотолщинный
9	EUROPEAN_FONT	Мнемоника отсутствует	Однообводный
10	BOLD_FONT	Мнемоника отсутствует	Жирный, равнотолщинный

Параметр `dir` принимает всего два значения — 0 (мнемоническая константа `HORIZ_DIR`) или 1 (`VERT_DIR`). В первом случае устанавливается режим воспроизведения горизонтальных надписей, во втором — повернутых на 90° против часовой стрелки. Последнее может оказаться полезным для подписи вертикальных осей на графиках.

Третий параметр `size`, значения которого должны принадлежать интервалу [1,10], выполняет роль масштабного коэффициента, с помощью которого можно незначительно увеличивать (`size > 4`) или уменьшать (`size < 4`) размеры букв.

Гораздо более гибкое управление размерами букв в векторных шрифтах обеспечивает процедура `setusercharsize`:

```
SetUserCharSize(mx,dx,my,dy);
```

Ее целочисленные аргументы выполняют роль множителей и делителей для координат (`x`,`y`) каждой точки контура букв:

```
X=x*mx/dx;
Y=y*my/dy;
```

Выбирая те или иные значения этих параметров, можно осуществить раздельное масштабирование по каждой оси, сделав, например, узкие или широкие буквы. Коэффициент увеличения можно довести до такого значения, при котором единственная буква занимает весь экран с четко различимыми элементами ее контура.

С помощью функций `textwidth` и `textheight`, единственным параметром которых является текстовая строка, можно определить ширину и высоту габаритного прямоугольника в пикселах. Предварительное знание размеров подписи может оказаться полезным при выборе места ее расположения в последующей процедуре `outtextxy`.

В качестве примера ниже приводится программа, с помощью которой были получены изображения букв на рис. 8.2. Подбор некоторых параметров этой программы был сделан экспериментальным путем. Сначала была выбрана линия строки, на которой должны были располагаться буквы разных шрифтов (`y = 470`). Затем путем подбора коэффициентов увеличения были установлены приемлемые габариты буквы `A` для шрифта `SMALL_FONT`. Получившиеся при этом "круглые" размеры габаритного прямоугольника (`w = 180`, `h = 270`) были использованы для определения масштабного коэффициента шрифтов `SANS_SERIF_FONT` и `TRIPLEX_FONT`. После нескольких пристрелочных проб, дававших "перелеты" и "недолеты", были найдены подходящие коэффициенты для каждого из шрифтов. Подбирать их пришлось по каждой оси свой и с точностью до второго знака после запятой. Обратите внимание на то, что одним и тем же коэффициентом не всегда удается выровнять размеры букв в разных шрифтах. Объясняется это тем, что разработка контуров векторных шрифтов выполнялась на координатных сетках с разными шагами. Более того, трансляция этой программы в системах TC 2.0 и BC 3.1 дает разные результаты т. к. в каждой из систем используются свои наборы шрифтов, которые слегка отличаются по начертаниям букв.

Самые большие неприятности доставил шрифт `DEFAULT_FONT`. Для него нельзя использовать функцию `setusercharsize` после выбора шрифта (`settextstyle`). Для того чтобы растровая буква оказалась на линии строки, ее пришлось опустить на межстрочный промежуток, равный 1 пикселу (с учетом масштабного множителя — на 10 пикселов).

Программа 8_13.c

```
/* Буква "A" в разных шрифтах */
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
  int x,y=470;
  void a_in_box(int dy)
{
  int w,h;
/* Определение размеров габаритного прямоугольника */
  w=textwidth("A");
  h=textheight("A");
  printf("w=%d h=%d ",w,h);
/* Обводка границ прямоугольника со смещением на 2 пиксела */
  rectangle(x-2,y-h,x+w-2,y+2);
/* Вывод буквы A */
```

```
  outtextxy(x,y+dy,"A");
}
  main()
{
  int gd=0,gm;

  initgraph(&gd,&gm,"");
/* Установки для рисования черным по белому */
  setpalette(1,0);
  setcolor(1);
  setbkcolor(15);
  settextjustify(0,0);
  settextstyle(2,0,1);          /* SmallFont */
  setusercharsize(30,1,267,1);
  x=4;
  a_in_box(0);
  settextstyle(3,0,1);          /* SansSerifFont */
  setusercharsize(86,10,75,10);
  x=186;
  a_in_box(0);
  settextstyle(1,0,1);          /* TriplexFont */
  setusercharsize(95,10,78,10);
  x=368;
  a_in_box(0);
  settextstyle(0,0,10); /* DefaultFont, общее увеличение=10 */
  x=550;
  a_in_box(10);
  getch();
  closegraph();
}
```

Задачи, советы и ответы

Задание 8.14. Построение шахматной доски

Написать программу, которая рисует на экране шахматную доску. Идея программы подсказана К. Э. Садыровым [7], однако мы предпочли другую реализацию с целью демонстрации идентичных программ на разных языках.

Совет 1 (общий)

Структура программы мало чем отличается от программ, демонстрировавших палитру (см. выше). Единственная тонкость заключается в организации правильного чередования цвета заливки смежных квадратов как в пределах строки, так и при переходе на следующую строку. Для этой цели использован прием "мерцающего бита". При каждом повторении внутреннего цикла величина `col` принимает чередующиеся значения 7 или 8 (в "мерцающем бите" аналогичная операция выглядела бы так: `bit=1- bit` или `bit:=not bit`). Если такое же действие не было бы предусмотрено и во внешнем цикле, то две смежные линейки начинались бы с квадратов одинакового цвета.

Совет 2 (QBasic)

Для подчеркивания границ каждого поля доски приходится сначала заливать очередную клетку, а потом обводить ее границу.

Совет 3 (Си, Паскаль)

Построение залитого поля с одновременной обводкой его границы можно выполнить процедурой `bar3d` с нулевой глубиной трехмерного столбика.

Программа 8_14.bas

```
REM Построение шахматной доски
DEFINT A-Z
SCREEN 12
x0=10: y0=10: col=8: w=50
FOR y=y0 TO y0+7*w STEP w
 col=15-col
 FOR x=x0 TO x0+7*w STEP w
   LINE (x,y)-(x+w,y+w),col,BF :' Заливка клетки
   LINE (x,y)-(x+w,y+w),15,B :' Обводка границ
   col=15-col:' Цвет для смежной клетки
 NEXT x
NEXT y
```

Программа 8_14.c

```
/* Построение шахматной доски */
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
  main()
{
```

```
  int x,y,x0=10,y0=10,col=8,w=50;
  int gd=0,gm;
  initgraph(&gd,&gm,"");
  for(y=y0; y<y0+8*w; y+=w)
    {col=15-col;
    for(x=x0; x<x0+8*w; x+=w)
      {
        setfillstyle(1,col);
        bar3d(x,y,x+w,y+w,0,1); /* Обводка и заливка клетки */
        col=15-col; /* Цвет для смежной клетки */
      }
    }
  getch();
  closegraph();
}
```

Программа 8_14.pas

```
program shach;
{ Построение шахматной доски }
uses Graph;
const
 x0=10; y0=10; w=50;
 col:integer=8;
var
  i,j,gd,gm,x,y:integer;
begin
  gd:=0;
  initgraph(gd,gm,'');
  for i:=0 to 7 do
  begin
    col:=15-col;
    y:=y0+i*w;
    for j:=0 to 7 do
    begin
      x:=x0+j*w;
      setfillstyle(1,col);
      bar3d(x,y,x+w,y+w,0,true); { Обводка и заливка клетки }
      col:=15-col; { Цвет для смежной клетки }
    end;
```

```
  end;
  readln;
  closegraph;
end.
```

Задание 8.15. Отображение семисегментных цифр

В электронных часах с цифровым индикатором цифры формируются как комбинации из семи сегментов (рис 8.3). Составить функцию (процедуру) `cifra`, которая по заданным координатам (`x`,`y`) и числовому значению цифры `k` (`0 <= k <= 9`) формирует на экране ее графическое изображение.

Совет 1 (общий)

Выберем в качестве точки привязки (`x`,`y`) вершину семисегментной комбинации, расположенную в левом верхнем углу. Очевидно, что для плотного примыкания смежных горизонтальных и вертикальных сегментов их срезы должны быть направлены под углами 45 и 135 градусов.

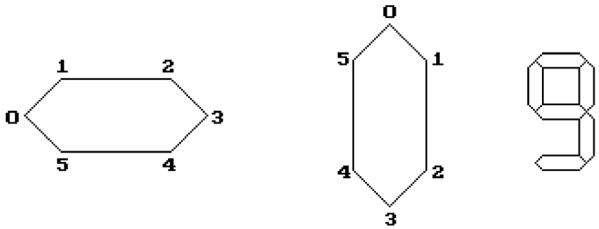


Рис. 8.3. Горизонтальный и вертикальный сегменты цифр

Если пронумеровать точки контура одного сегмента от 0 до 6 по часовой стрелке от самой левой (для горизонтальных) или самой верхней (для вертикальных) и считать, что начальная точка имеет координаты (`x0`,`y0`), то координаты остальных точек можно рассчитать, используя относительные смещения, приведенные в табл. 8.4. Через `a` и `b` здесь обозначены смещения по координатным осям для наклонных и горизонтальных (вертикальных) ребер контура. Задавая различные числовые значения этих параметров, можно строить сегменты нужного размера. Удобно выбирать `b = 4*a`, хотя не возбраняются и любые другие соотношения. В последующих расчетах нам понадобится и константа `c = = 2*a + b`, представляющая максимальную длину (высоту) сегмента.

Таблица 8.4. *Расчет координат точек с использованием относительного смещения*

Номер точки	Смещения по координатам относительно предыдущей точки			
	Для горизонтального сегмента		Для вертикального сегмента	
	По x	По y	По x	По y
0	0	0	0	0
1	a	-a	a	a
2	b	0	0	b

Таблица 8.4 *(окончание)*

Номер точки	Смещения по координатам относительно предыдущей точки			
	Для горизонтального сегмента		Для вертикального сегмента	
	По x	По y	По x	По y
3	a	a	-a	a
4	-a	a	-a	-a
5	-b	0	0	-b

Из табл. 8.4 видно, что при реализации программы достаточно иметь всего два массива смещений:

```
d1 = (0,a,b,a,-a,-a)  и  d2 = (0,-a,0,a,a,0)
```

Для вертикального сегмента они меняются местами и у одного из них меняются знаки.

Установим соотношения между точкой привязки цифры и координатами начальных точек каждого из семи сегментов. Пронумеруем сверху вниз числами 1, 2 и 3 горизонтальные сегменты, а числами 1, 2, 3 и 4 — вертикальные сегменты слева направо и сверху вниз. Несложно убедиться в том, что координаты их нулевых вершин вычисляются следующим образом.

Для горизонтальных сегментов:

```
x1 = x    x2 = x      x3 = x
y1 = y    y2 = y + c  y3 = y + 2*c
```

Для вертикальных сегментов:

```
x1 = x    x2 = x + c  x3 = x       x4 = x + c
y1 = y    y2 = y      y3 = y + c   y4 = y + c
```

Совет 2 (Си)

Для того чтобы установить соответствие между числовым значением отображаемой цифры и составляющим ее набором сегментов, предлагается использовать семиразрядные двоичные числа, единичные разряды в которых устанавливают присутствие того или иного сегмента. Например, контур цифры 0

образуется первым и третьим горизонтальными сегментами и всеми четырьмя вертикальными. Поэтому цифре 0 можно поставить в соответствие код 1011111. Цифра 9 описывается кодом 1111101 и т. д. В приведенном ниже тексте программы на Си эти коды представлены не самым рациональным способом — в виде двумерного массива. Более оптимальный вариант предложен в следующем задании.

Программа 8_15.c

```
/* Построение 7-сегментных цифр */
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void cifra(int x,int y,int n);

  void main()
{
  int gd=0,gm,k;
  initgraph(&gd,&gm,"");
  setcolor(4);
  for(k=0;k<10;k++)
    cifra((k+1)*50,30,k);
  getch();
  return;
}
/*----------------------------------------*/
void cifra(int x,int y,int n)
{
  const a=4,b=20,c=a+a+b;          //габариты сегмента
  const d1[]={a,b,a,-a,-b};        //приращения по координатам
  const d2[]={-a,0,a,a,0};         //приращения по координатам
  const dx[]={0,0,0,0,c,0,c};      //смещения начальных вершин по x
  const dy[]={0,c,2*c,0,0,c,c};    //смещения начальных вершин по y
  int j,k;
  struct {int x;int y;} xy[6];     //координаты точек контура
  char q[10][7]={ {1,0,1,1,1,1,1}, //перемычки цифры 0
  {0,0,0,0,1,0,1},{1,1,1,0,1,1,0}, //перемычки цифр 1,2
  {1,1,1,0,1,0,1},{0,1,0,1,1,0,1}, //перемычки цифр 3,4
  {1,1,1,1,0,0,1},{1,1,1,1,0,1,1}, //перемычки цифр 5,6
  {1,0,0,0,1,0,1},{1,1,1,1,1,1,1}, //перемычки цифр 7,8
```

```
  {1,1,1,1,1,0,1}};                //перемычки цифры 9

  for(j=0; j<7; j++)
    {
      if(q[n][j]==0) continue;
      xy[0].x=x+dx[j];
      xy[0].y=y+dy[j];
      for(k=1; k<6;k++)
        {//цикл вычисления координат точек контура
          if(j<3)
            {//пересчет для горизонтальных перемычек
              xy[k].x=xy[k-1].x+d1[k-1];
              xy[k].y=xy[k-1].y+d2[k-1];
            }
          else
            {//пересчет для вертикальных перемычек
              xy[k].x=xy[k-1].x-d2[k-1];
              xy[k].y=xy[k-1].y+d1[k-1];
            }
        }
      fillpoly(6,(int *)xy);       //заливка перемычки
    }
}
```

Задание 8.16. Цифровые часы

Используя составленную в предыдущем примере программу формирования семисегментных цифр, написать программу, отображающую на экране текущее показание компьютерных часов.

Попытка использовать предыдущую программу `cifra` сразу же выявляет ее недостаток. Она хорошо "рисует" цифры на чистом экране, но в часах приходится строить новую цифру поверх уже нарисованной. Поэтому в самом начале процедуры придется стереть предыдущее изображение, например с помощью процедуры `bar`.

Совет 1 (Паскаль)

Показания системных часов в Паскале можно опросить с помощью процедуры `gettime`. Для того чтобы исключить излишнее мигание на экране, целесообразно перерисовывать цифры часов, минут или секунд только при их изменении. С этой целью в программе предусмотрен двойной набор переменных для хранения пре-

дыдущего (h1,m1,s1) и текущего (h2,m2,s2) показаний часов. В начале работы программы в первый набор заносятся невообразимые значения, поэтому первое же обращение к gettime вызовет перерисовку всех трех цифр и запоминание нормальных значений часов, минут и секунд. В последующих циклах будут перерисовываться только те цифры, значения которых изменились.

Совет 2 (Паскаль)

Позиции цифр на экране и двоеточия, разделяющего часы, минуты и секунды, подбирались экспериментальным путем. В программе не очень изящно реализовано отображение двоеточия. Было бы лучше, если бы оно мигало с частотой раз в секунду. Но усовершенствования такого рода мы вполне доверяем читателям.

Совет 3 (Паскаль)

В отличие от приводившейся выше функции cifra здесь использован одномерный массив двоичных чисел, что уменьшает объем используемых данных и несколько ускоряет работу программы (операции с элементами одномерных массивов выполняются быстрее). Прощупывание двоичных битов ведется в цикле путем сдвига соответствующей константы на нужное число разрядов влево и логического умножения на код соответствующих перемычек.

Совет 4 (Паскаль)

Программа оформлена в виде бесконечного цикла, работа которого прерывается нажатием любой клавиши.

Программа 8_16.pas

```
program time;
uses Crt, Dos, Graph;
var
  gd,gm,k:integer;
  h1,m1,s1,h2,m2,s2,hs2:word;
procedure cifra(x,y,n:integer);
type
  a4=array [0..4] of integer;
  a6=array [1..7] of byte;
const
  a=4; b=20; c=a+a+b;
  d1:a4=(a,b,a,-a,-b);
  d2:a4=(-a,0,a,a,0);
  dx:a6=(0,0,0,0,c,0,c);
  dy:a6=(0,c,2*c,0,0,c,c);
  q:array[0..9]of byte=
```

```
    ($5F,$5,$76,$75,$2D,$79,$7B,$45,$7F,$7D);
var
  xy:array [0..5] of PointType;
  j,k,d:byte;
begin
  setfillstyle(0,0);
  bar(x-a,y-a,x+(c+a+a),y+2*(c+2*a));
  d:=q[n];
  for j:=1 to 7 do
    begin
      if ((d) and ($80 shr j))=0 then continue;
      xy[0].x:=x+dx[j];
      xy[0].y:=y+dy[j];
      for k:=1 to 5 do
      if j<4 then begin
        xy[k].x:=xy[k-1].x+d1[k-1];
        xy[k].y:=xy[k-1].y+d2[k-1]; end
      else begin
        xy[k].x:=xy[k-1].x-d2[k-1];
        xy[k].y:=xy[k-1].y+d1[k-1]; end;
      setfillstyle(1,14);
      fillpoly(6,xy);
    end;
end;

begin
  gd:=0;
  initgraph(gd,gm,'');
  settextstyle(0,0,4);
  setcolor(14);
  outtextxy(136,44,':'); { Разделитель часов и минут }
  outtextxy(256,44,':'); { Разделитель минут и секунд }
  setcolor(4);
  h1:=100; { Начальные установки }
  m1:=100;
  s1:=100;
  repeat
    gettime(h2,m2,s2,hs2); { Опрос текущего времени }
    if h1<>h2 then begin   { Если изменились часы }
```

```
      k:=h2 div 10; cifra(50,30,k); { Старшая цифра часов }
      k:=h2 mod 10; cifra(100,30,k); { Младшая цифра часов }
      h1:=h2;
   end;
   if m1<>m2 then begin { Если изменились минуты }
      k:=m2 div 10; cifra(170,30,k);
      k:=m2 mod 10; cifra(220,30,k);
      m1:=m2;
   end;
   if s1<>s2 then begin { Если изменились секунды }
      k:=s2 div 10; cifra(290,30,k);
      k:=s2 mod 10; cifra(340,30,k);
      s1:=s2;
   end;
  until KeyPressed;
  closegraph;
end.
```

Задание 8.17. Графические спрайты

Спрайты (от англ. Sprite — "дух, привидение") представляют собой небольшие графические изображения, перемещаемые по экрану для имитации движущихся фигур или предметов. Без спрайтов не обходится ни одна динамическая игра. Для некоторых изображений достаточно единственного спрайта, который просто перемещается по заданной траектории. Таким спрайтом, например, может быть изображение летящего самолета. В других случаях приходится манипулировать цепочкой спрайтов, на которых зафиксированы отдельные кадры, соответствующие разным фазам движения.

Основная идея простейших программ анимации заключается в организации быстрой смены кадров на экране, при которой отображаются последовательные фазы изменения состояния фигуры и производится ее перемещение. При этом крайне нежелательно, чтобы спрайты "затирали" находящийся под ними интерьер. Последнее достигается за счет разумного подбора цветовых оттенков как спрайта, так и фона, и вывода спрайтов в режиме `XOR`.

Продемонстрируем технику перемещения единственного спрайта —"летающей тарелки" на фоне "звездного неба". Увеличенное изображение "тарелки" было нарисовано на миллиметровой бумаге в прямоугольной области размером 43×24 мм и представляло собой эллипс с полуосями 20 и 8 мм, внутри которого проходил эллиптический поясок (дуга эллипса со смещенным цен-

тром). Из "корпуса" тарелки под небольшими углами расходились отрезки прямых — стойки локаторов, на концах которых размещались две круговые "антенны" — кружочки небольшого радиуса (2 мм). Контуры тарелки рисуются белым цветом, а внутренность корпуса заливается красным цветом.

Совет 1 (общий)

Для создания образа спрайта в оперативной памяти поступим следующим образом. Сначала воспроизведем спрайт в любом месте экрана и запомним содержимое соответствующего участка видеопамяти в массиве соответствующего размера (в QBasice — оператор `GET`, на Си или Паскале — процедура `getimage`). Затем повторно выведем спрайт на то же место экрана в режиме `XOR` для стирания статичного изображения.

Совет 2 (общий)

Для имитации звездного неба построим 1000 случайных точек на экране, окрашенных в случайные цвета.

Совет 3 (общий)

Организуем бесконечный цикл, в котором изображение тарелки на небольшое время (порядка 0,6 с) появляется в некоторой точке с координатами (`x`,`y`), а затем стирается. После этого точка (`x`,`y`) смещается на случайные перемещения (`dx`,`dy`) и цикл повторяется до нажатия какой-либо клавиши пользователем. Для того чтобы перемещения из текущей точки в следующую выглядели случайными, можно менять знаки приращений `dx` и `dy`, когда они оказываются четными или нечетными. Кроме того, необходимо организовать проверку очередной точки на принадлежность экрану и запретить ее выход из допустимого интервала.

Совет 4 (QBasic)

В связи с тем, что задержка по оператору `SLEEP` может быть организована только с точностью до секунды, предлагается воспользоваться оператором `SOUND f,dt`. Он организует выдачу звукового сигнала с частотой `f` Гц, длящегося `dt` тиков (в 1 с содержится 18,2 тика). А чтобы звук не доставлял неприятностей, можно выбрать частоту, не воспринимаемую человеческим ухом (например, `f = 32767` Гц).

Программа 8_17.bas

```
REM Летающая тарелка
DEFINT A-Z
DIM TARELKA(300)
X=320: Y=240
SCREEN 12
CIRCLE (100,50),20,15,,,.4
PAINT (100,50),4,15
```

```
CIRCLE (100,46),20,15,3.3,0,.3
LINE (107,44)-(110,38),15
CIRCLE (110,38),2,15
LINE (93,44)-(90,38),15
CIRCLE (90,38),2,15
' Запомнили образ
GET (79,36)-(121,59),TARELKA
' Стерли
PUT (79,36),TARELKA,XOR
' Построение звездного неба
FOR I=0 TO 1000
  PSET (INT(RND*639)+1,INT(RND*479)+1),INT(RND*15)+1
NEXT I
DO WHILE INKEY$=""
  PUT (X,Y),TARELKA,XOR :' Отображение тарелки
  SOUND 32767,4.92 :' Задержка на 0.6 сек
  PUT (X,Y),TARELKA,XOR :' Стирание тарелки
  DX=INT(RND*60)+1: IF DX MOD 2=1 THEN DX=-DX
  X=X+DX :' Смещение по горизонтали
  IF X>590 THEN X=590 :' Контроль за правой границей экрана
  IF X<0 THEN X=0 :' Контроль за левой границей экрана
  DY=INT(RND*40)+1: IF DY MOD 2=1 THEN DY=-DY
  Y=Y+DY :' Смещение по вертикали
  IF Y>450 THEN Y=450 :' Контроль за нижней границей экрана
  IF Y<0 THEN Y=0 :' Контроль за верхней границей экрана
LOOP
END
```

Программа 8_17.c

```
/* Летающая тарелка */
#include <graphics.h>
#include <stdlib.h>
  main()
{
  int x=320, y=240, i, dx, dy, gd=0, gm;
  char Tarelka[600];
  initgraph(&gd,&gm,"");
  randomize();
/* Построение летающей тарелки */
```

```
  setfillstyle( SOLID_FILL, 4);
  fillellipse(100, 50, 20, 8);
  ellipse(100, 46, 190, 357, 20, 6);
  line(107, 44, 110, 38);
  circle(110, 38, 2);
  line(93, 44, 90, 38);
  circle(90, 38, 2);
/* Запомнили изображение и стерли его */
  getimage(79, 36, 121, 59, Tarelka);
  putimage(79, 36, Tarelka, XOR_PUT);
/* Построение звездного неба  */
  for ( i=0 ; i<1000; ++i )
    putpixel(random(639), random(479), random(15)+1);
  while ( !kbhit() )
    { /* Бесконечный цикл до нажатия клавиши */
      putimage(x, y, Tarelka, XOR_PUT);   /* вывод тарелки */
      delay(6000);                        /* задержка */
      putimage(x, y, Tarelka, XOR_PUT);   /* стирание тарелки */
      /* Перемещение тарелки */
      dx = random(60);
      if (dx % 2 == 0 ) dx = - dx;
      x = x + dx;
      if (x > 590) x = 590;
      else if (x < 0) x = 0;
      dy = random(40);
      if (dy % 2 == 0 ) dy = - dy;
      y = y + dy;
      if (y > 450) y = 450;
      else if (y < 0) y = 0;
    }
  getch();
}
```

Программа 8_17.pas

```
{ Программа "Летающая тарелка" }
program nlo;
uses Crt,Graph;
var
  x,y,i,dx,dy,gd,gm:integer;
```

```
  Tarelka:array [1..600] of byte;
begin
  x:=320;
  y:=240;
  gd:=0;
  initgraph(gd,gm,'');
  randomize;
{ Построение летающей тарелки }
  setfillstyle(SolidFill, 4);
  fillellipse(100, 50, 20, 8);
  ellipse(100, 46, 190, 357, 20, 6);
  line(107, 44, 110, 38);
  circle(110, 38, 2);
  line(93, 44, 90, 38);
  circle(90, 38, 2);
{ Запомнили изображение и стерли его }
  getimage(79, 36, 121, 59, Tarelka);
  putimage(79, 36, Tarelka, XORput);
{ Построение звездного неба }
  for i:=0 to 1000 do
      putpixel(random(639), random(479), random(15)+1);
  repeat       { Бесконечный цикл до нажатия клавиши }
    putimage(x,y,Tarelka,XORput);   { вывод тарелки }
    delay(6000);                    { задержка }
    putimage(x,y,Tarelka,XORput);   { стирание тарелки }
{ Перемещение тарелки }
    dx:=random(60);
    if odd(dx) then dx:=- dx;
    x:=x+dx;
    if x>590 then x:=590;
    if x<0 then x:=0;
    dy:=random(40);
    if odd(dy) then dy:=- dy;
    y:=y+dy;
    if y>450 then y:=450;
    if y<0 then y:=0;
  until KeyPressed;
  closegraph;
end.
```

Задание 8_18. Биоритмы

Некоторые исследователи утверждают, что каждому человеку присущи три периодических процесса, сопровождающие его жизнь с момента рождения. Первый из них с периодом 23 дня соответствует физическому состоянию, второй с периодом 28 дней — интеллектуальному, третий с периодом 33 дня — эмоциональному. По каждому процессу можно построить график, отложив по горизонтальной оси время T в днях с момента рождения, а по вертикальной оси — амплитуду синусоиды со своим периодом:

```
PH=SIN(2*PI*T/23)
IN=SIN(2*PI*T/28)
EM=SIN(2*PI*T/33)
```

Совместив все графики на общем рисунке, можно определить моменты общего подъема или спада потенциала человека. Особого единства в трактовке "плохих" интервалов жизни человека нет. Одни считают, что к "черным" дням относятся моменты, когда все три кривые одновременно или в достаточно узком интервале пересекают ось времени, другие склонны рассматривать в качестве дней депрессии моменты, когда все кривые имеют общую или близкую точку минимума.

Составим программу построения графика биоритмов на текущий месяц по заданной дате рождения. Для упрощения анализа високосных лет будем считать, что эта дата относится к XX столетию. Тогда можно ограничиться проверкой остатка от деления последующих лет на 4.

Совет 1 (общий)

С целью структурирования программы, кроме ее главной части целесообразно выделить в отдельные функции следующие процедуры:

- `offset` — определение количества дней, прошедших от 1.01.1900 г. до заданной даты (`dd` — день, `mm` — месяц, `yy` — год);
- `axis` — построение осей для графика биоритмов с нанесением рисок по горизонтальной оси и числовых меток, упрощающих определение дня текущего месяца;
- `grafik` — построение графика синусоиды с заданными периодом `T` и начальным смещением `dfi`, окрашенной в указанный цвет.

Совет 2 (общий)

Алгоритм процедуры `offset` может быть самым примитивным. Например, можно прибавлять к сумме по 365 дней за каждый полностью прошедший год и компенсировать добавочным днем каждый високосный. В начале работы в сумму заносим 365 — число дней в невисокосном 1900 году. В текущем году количество дней подсчитываем с помощью массива `days`, задающего количе-

ство дней в каждом месяце. Нужно только откорректировать число дней февраля (`days[1] = 29`), если текущий год является високосным.

Совет 3 (общий)

Для определения текущей даты можно воспользоваться одной из функций, предлагаемых соответствующей системой программирования — `DATE$` (QBasic), `getdate` (Си), `GetDate` (Паскаль).

Совет 4 (QBasic)

Самым неудобным моментом в программе является подбор положения цифровых меток у графических штрихов на оси `x`. К сожалению, кроме метода проб и ошибок, мы не можем предложить ничего разумного. Дело в том, что оператор `LOCATE` позиционирует курсор в терминах строка-столбец, а графика оперирует с пикселами. Поэтому приходится слегка смещать либо положение рисунка, либо положение подписи.

Совет 5 (Си)

Для абсолютной идентичности результатов программ на Си и Паскале пришлось написать очень нехитрую функцию `round`, округляющую вещественный аргумент до ближайшего целого числа. К сожалению, воспользоваться напрямую одной из библиотечных функций `floor` или `ceil` нельзя.

Совет 6 (Паскаль)

Обратите внимание на то, что в графическом режиме мы не пользуемся процедурами и функциями модуля `Crt`, ибо они конфликтуют с процедурами модуля `Graph`. В отличие от этого в Си, например, функция `gotoxy` прекрасно работает и в графическом режиме.

Программа 8_18.bas

```
DECLARE SUB AXIS()
DECLARE SUB GRAFIK(t!,dfi!,col!)
DECLARE FUNCTION OFFSET!(d!,m!,y!)
REM Построение биоритмов на текущий месяц
DATA 30,28,31,30,31,30,31,31,30,31,30,31
DIM SHARED days(11),a
FOR k=0 TO 11: READ days(k): NEXT k
wwod:
PRINT "Биоритмы на текущий месяц"
PRINT "Введите день, месяц (числом) и год своего рождения"
INPUT d,m,y
d$=DATE$: m1=VAL(LEFT$(d$,2)): d1=VAL(MID$(d$,4,2))
y1=VAL(RIGHT$(d$,4))
```

```
IF (m<1)OR(m>12)OR(d<1)OR(d>days(m))OR(y<1900)OR(y>y1) THEN
  PRINT "Вы ошиблись. Повторите ввод"
  SLEEP 1:  GOTO wwod
END IF
IF (y1 MOD 4)=0 THEN days(2)=29:' поправка на високосный год
a=days(m1):' число дней в текущем месяце
' Интервал от дня рождения до начала текущего месяца
dd=OFFSET(1,m1,y1)-OFFSET(d,m,y)
SCREEN 12
PRINT "красный - физическое состояние"
PRINT "синий - эмоциональное состояние"
PRINT "зеленый - интеллектуальное состояние"
' Построение и разметка координатных осей
AXIS
GRAFIK 23,dd MOD 23,4
GRAFIK 28,dd MOD 28,2
GRAFIK 33,dd MOD 33,1
SLEEP
END

SUB AXIS
  LINE (0,140)-(0,340)
  LINE (0,240)-(a*20,240)
  FOR j=1 TO a
    stroke=5
    IF (j MOD 5)=0 THEN stroke=10
    LINE (j*20,240+stroke)-(j*20,240-stroke)
    IF stroke=10 THEN LOCATE 17,(j*20-4)\8: PRINT j
  NEXT j
END SUB

SUB GRAFIK(t,dfi,col)
CONST twopi=6.2831853#
  x=0: y=240-100*SIN(twopi*dfi/t): COLOR col
  PSET (x,y)
  FOR k=1 TO a
    x1=20*k
    y1=240-100*SIN(twopi*(k+dfi)/t)
    LINE -(x1,y1)
```

```
  NEXT k
END SUB

FUNCTION OFFSET (d,m,y)
REM Вычисляет количество дней от 1.01.1900 до d.m.y
  dd=365: 'Количество дней в 1900 г
  REM Цикл учета полных лет
  FOR k%=1901 TO y-1
    dd=dd+365
    REM Поправка на високосный год
    IF (k% MOD 4)=0 THEN dd=dd+1
  NEXT k%
  REM Учет дней в году y до месяца m
  FOR k%=1 TO m-1: dd=dd+days(k%): NEXT k%
  OFFSET=dd+d:' Добавление дней, прошедших в месяце m
END FUNCTION
```

Программа 8_18.c

```
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <graphics.h>
#include <conio.h>

#define twopi 2*M_PI

void axis(void);
void grafik(int t, int dfi, int color);
long offset(int d,int m,int y);
int round(double x);

  char days[]={30,28,31,30,31,30,31,31,30,31,30,31};
  int a;

  main()
{
  int gd=0,gm,k,d,m,y,m1,y1;
  struct date q;
```

```
  long dd;
wwod:
  clrscr();
  gotoxy(1,1);
  printf("Биоритмы на текущий месяц\n");
  printf("Введите день, месяц (числом) и год своего рождения\n");
  scanf("%d %d %d",&d,&m,&y);
  getdate(&q);
  m1=q.da_mon;
  y1=q.da_year;
  if(m<1 || m>12 || d<1 || d>days[m-1] || y<1900 || y>y1)
    {
      printf("\nВы ошиблись. Нажмите Enter и повторите ввод");
      getch();  goto wwod;
    }
  if(y1 % 4 == 0) days[1]=29;
  a=days[m1-1];
  dd=offset(1,m1,y1)-offset(d,m,y);
  initgraph(&gd, &gm, "");
  gotoxy(1,1); printf ("красный - физическое состояние");
  gotoxy(1,2); printf ("синий   - эмоциональное состояние");
  gotoxy(1,3); printf ("зеленый - интеллектуальное состояние");
  axis();
  grafik(23,dd % 23,RED);
  grafik(28,dd % 28,GREEN);
  grafik(33,dd % 33,BLUE);
  getch();
  closegraph();
}
/*-------------------------------------------------*/
void axis(void)
{ /* Построение и маркировка осей */
  int j,str;
  char qq[5];
  line(0,140,0,340);    /* вертикальная ось */
  line(0,240,a*20,240); /* горизонтальная ось */
  /* цикл построения малых и больших штрихов на оси x */
  for(j=1;j <= a;j++)
```

```
    {
      str=5;
      if(j % 5 == 0) str=10;
      line(j*20,240+str,j*20,240-str);
      if (str == 10)
        { /* маркировка больших штрихов через 5 дней */
          itoa(j,qq,10);
          outtextxy(j*20-5,240+20,qq);
        }
    }
}
/*-----------------------------------------------------*/
void grafik(int t,int dfi,int color)
/***************************/
/* построение синусоиды    */
/* t - период (в днях)     */
/* dfi - смещение для x=0  */
/* color - цвет графика    */
/***************************/
{
  int x,y,x1,y1,k;
  x=0;
  y=round(240-100*sin(twopi*dfi/t));
  setcolor(color);
  moveto(x,y);
  for(k=1; k<=a; k++)
    {
      x1=20*k;
      y1=round(240-100*sin(twopi*(k+dfi)/t));
      lineto(x1,y1);
   }
}
/*-----------------------------------------------*/
long offset(int d,int m,int y)
{ /* Вычисляет количество дней от 1.01.1900 до d.m.y */
  int k;
  long dd;
  dd=365; /* Количество дней в 1900 г */
  /* Цикл учета полных лет */
```

```
  for(k=1901; k<y; k++)
    {
      dd += 365;
      if(k % 4 == 0) dd++; /* Поправка на високосный год */
    }
  dd += d;
  /* Учет дней в году y до месяца m */
  for(k=0; k<m-1; k++)
    dd += days[k]; /* Добавление дней, прошедших в месяце m */
  return dd;
}
/*-----------------------------------------------------*/
int round(double x)
{ /* округление вещественного числа до ближайшего целого */
  if(x>=0) return (int)(x+0.5);
  return (int)(x-0.5);
}
```

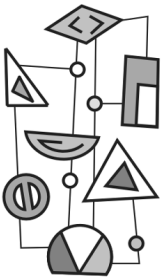
Программа 8_18.pas

```
program bioritms;
uses Graph,Dos;
label wwod;
const
  twopi=2*Pi;
  days:array [1..12] of byte = (30,28,31,30,31,30,31,31,30,31,30,31);
var
  a,k,d,m,y,d1,m1,y1:word;
  gd,gm:integer;
  dd:longint;

procedure axis;
var
  j,stroke:integer;
  s:string[2];
begin
  line(0,140,0,340);
  line(0,240,a*20,240);
  for j:=1 to a do
```

```
    begin
      stroke:=5; str(j,s);
      if j mod 5 = 0 then stroke:=10;
      line(j*20,240+stroke,j*20,240-stroke);
      if stroke=10 then outtextxy(j*20-5,240+20,s);
    end;
end;

procedure grafik(t,dfi,color:integer);
var
  x,y,x1,y1,k:integer;
begin
  x:=0;
  y:=round(240-100*sin(twopi*dfi/t));
  setcolor(color);
  moveto(x,y);
  for k:=1 to a do
    begin
      x1:=20*k;
      y1:=round(240-100*sin(twopi*(k+dfi)/t));
      lineto(x1,y1);
    end;
end;

function offset(d,m,y:integer):longint;
{Вычисляет количество дней от 1.01.1900 до d.m.y}
var
  k:integer;
  dd:longint;
begin
  dd:=365;  {Количество дней в 1900 г}
  {Цикл учета полных лет}
  for k:=1901 to y-1 do
    begin
      dd:=dd+365;
      {Поправка на високосный год}
      if k mod 4 = 0 then inc(dd);
    end;
  {Учет дней в году y до месяца m}
```

```
  for k:=1 to m-1 do  inc(dd,days[k]);
  sdwig:=dd+d; {Добавление дней, прошедших в месяце m}
end;

begin
  gd:=0;
wwod:
  writeln('Биоритмы на текущий месяц');
  writeln('Введите день, месяц (числом) и год своего рождения');
  readln(d,m,y);
  GetDate(y1,m1,d1,k); {опрос текущей даты}
  if (m<1)or(m>12)or(d<1)or(d>days[m])or(y<1900)or(y>y1) then
    begin
      write('Вы ошиблись. Нажмите Enter и повторите ввод');
      readln;  goto wwod;
    end;
  if y1 mod 4 = 0 then days[2]:=29; {поправка на високосный год}
  a:=days[m1]; {число дней в текущем месяце}
 {Интервал от дня рождения до начала текущего месяца}
  dd:=offset(1,m1,y1)-offset(d,m,y);
  initgraph(gd,gm,'');
  outtextxy(0,0,'красный - физическое состояние');
  outtextxy(0,20,'синий - эмоциональное состояние');
  outtextxy(0,40,'зеленый - интеллектуальное состояние');
  axis;  {Построение и разметка координатных осей}
  grafik(23,dd mod 23,RED);
  grafik(28,dd mod 28,GREEN);
  grafik(33,dd mod 33,BLUE);
  readln;
  closegraph;
end.
```

Глава 9



Работа с календарными датами

Тематика этой главы инициирована той бурной компанией по поводу проблемы 2000 года (на Западе она более известна под аббревиатурой Y2K), которая была развернута в средствах массовой информации не без поддержки со стороны наиболее крупных производителей компьютеров и программного обеспечения. Несмотря на то, что проблема была чрезмерно раздута, элементарная безграмотность производителей компьютерных программ уже в 2000 году нанесла серьезный урон ряду банков Японии. Они не учли, что наступивший год был високосным, и многие банковские операции 29 февраля не состоялись. Мы надеемся, что материал настоящей главы поможет нашим читателям при разработке правильных программ, связанных с учетом временных интервалов.

Немного истории

Термин "календарь" обязан своим происхождением латинским словам calendae ("первый день месяца") и calendarium ("долговая книга"). В Древнем Риме на первое число каждого месяца приходилось время уплаты процентов по долгам.

Наиболее стабильным длительным промежутком времени с точки зрения ученых, ведущих астрономические наблюдения, является тропический год — интервал между моментами прохождения центра солнца через так называемую точку весеннего или осеннего равноденствия. Наблюдается это явление 21 марта и 23 сентября в обычный год и со смещением на один день — в високосный год. В этот момент лучи солнца падают на экватор отвесно и продолжительности ночи и дня совпадают. Длительность тропического года составляет 365 суток 5 часов 48 минут и примерно 46 секунд (365,2422 солнечных суток, длительность последних — 24 часа).

Древние календари, в большинстве своем, базировались на фазах Луны. Так, например, в Вавилоне год состоял из 12 лунных месяцев с чередующейся длительностью месяцев 29→30→29→30→ ... . Однако суммарная длительность такого года составляла 354 дня.

Один из первых солнечных календарей был принят в Египте примерно в IV в. до н. э. По египетскому календарю год состоял из 365 дней и включал

12 месяцев по 30 дней. В конце года 5 дней объявлялись праздничными и не входили ни в один из месяцев.

Наиболее близким к тропическому году стал римский календарь, принятый по указу императора Юлия Цезаря в 47 г. до н. э. Для ликвидации разницы с астрономическим годом длительность 46-го года была установлена в 432 дня. После этого вводились 3 обычных года по 365 дней, за которыми следовал 1 високосный год. Таким образом, средняя продолжительность римского года достигла 365,25 суток, что немного превышало длительность тропического года.

Тем не менее за 1500 лет набежало порядка 10 лишних суток. За очередное улучшение календаря принялся ватиканский ученый Луиджи Лилио. В 1582 году по указу папы Григория XIII был принят новый календарь, переход на который в разных странах растянулся на 200—300 лет. В России, например, новый календарь был введен декретом СНК РСФСР в январе 1918 года (после 31 января наступило 14 февраля, день недели при этом сохранился автоматически). Даты римского календаря стали называть "старым стилем", а даты григорианского календаря — "новым стилем". Суть ватиканской реформы заключалась в следующем. После четверга 4 октября 1582 года следующим днем объявлялась пятница 15 октября. Так устранялись набежавшие излишки. Второе новшество заключалось в том, что из круглых дат столетий исключались високосные годы, которые не делились нацело на 400. При этом средняя длительность года по григорианскому календарю оказалась больше тропического года всего на 26 с. Так что, разница в 1 сутки сможет набежать примерно к 4860 году.

Обратите внимание на специфику перехода от летоисчисления до новой эры. Первый год новой эры (эры Дионисия), последовавший после первого года до новой эры, отсчитывается от даты рождества Христова. С точки зрения математики, ему должен предшествовать нулевой год, которого не было. Поэтому в алгоритмах исчисления непрерывных календарных дат номера лет до новой эры следует считать отрицательными и увеличивать их на 1.

В конце того же XVI века астроном Джозеф Скалигер предложил новый способ отсчета времени. Начинался он от некоторого условного нуля (12 часов дня всемирного времени в понедельник 1 января 4713 года до новой эры, а с учетом приведенного выше замечания начальный год равен −4712) и представлялся в виде постоянно работающего таймера, отсчитывающего сутки и текущее время. Целая часть Юлианской даты JD (она была так названа в честь отца Скалигера) определяла количество суток, прошедших от начала отсчета, а дробная часть от 0,0 до 0,99999 соответствовала показанию суточных часов. Переход к следующей единице отсчета происходил в 12 часов очередного дня.

Например:

1 января 4712 г. до н.э. в 12 часов дня	JD = 0.00000
2 января 4712 г. до н.э. в 12 часов дня	JD = 1.00000
3 января 4712 г. до н.э. в 18 часов вечера	JD = 2.25000

1 января 1900 г. в 12 часов дня	JD = 2415021.0
22 июня 1941 г. в 4 часа утра	JD = 2430167.66667
9 мая 1945 г. в 12 часов дня	JD = 2431585.0
1 января 1999 г. в 12 часов дня	JD = 2451180.0
1 января 2000 г. в 12 часов дня	JD = 2451545.0

Следует заметить, что предложенная идея отсчета времени используется не только астрономами. В системах визуального программирования Borland C++ Builder и Delphi появился класс данных типа `TDateTime`, в объектах которого хранится обобщенное значение даты и времени в формате вещественного числа с двойной точностью (`double`). Его целая часть равна количеству дней, прошедших с полуночи 30 декабря 1899 года, а дробная часть соответствует времени дня. По сравнению с юлианской датой изменилась только точка начала отсчета.

Хранение даты и времени в формате юлианского дня представляется достаточно экономичным. Потребуется всего 8 байт для величины типа `double`, тогда как для запоминания символьной строки вида "YYYY/MM/DD HH:MM:SS" необходимо не менее 20 байт. Если нужно хранить только целую часть JD (тип `long`), то можно ограничиться 4 байтами. Конечно, для запоминания даты и времени в числовом машинном формате можно обойтись и 7 байтами (2 байта под год и по 1 байту под остальные компоненты). В некоторых системных программах MS-DOS попытки упаковать дату привели к еще более сжатым форматам. Вместо года Y хранится двоичное число, равное Y-1980 и принадлежащее интервалу [0,119], на номер месяца отведено 4 бита, а на все остальные компоненты — по 5 бит. Это позволило втиснуть дату и время в 32 двоичных разряда. Однако вместо полного номера года так можно представить только его младшие цифры, а вместо полного числа секунд — количество полусекунд. Для хранения дат создания файлов такой способ вполне пригоден, но для программ обработки календарных дат из более широкого временного интервала явно не годится.

Непрерывный таймер Дж. Скалигера удобен еще и тем, что позволяет очень просто вычислять различные временные интервалы как с точностью до суток, прошедших между двумя календарными датами, так и с точностью до секунд и даже долей секунды. В дальнейшем мы покажем, что по JD вычислить день недели можно всего за две операции.

Сложность заключается только в одном — надо перевести дату и время дня в показания юлианского хронометра. Первые алгоритмы такого перевода, которыми пользовались астрономы, базировались на таблицах юлианских дат, приходящихся на начало года. Дальше оставалось подсчитать порядковый день в году, соответствующий преобразуемой дате, и выполнить сложение. Однако для компьютерных программ табличный алгоритм неудобен, т. к. он связан с хранением довольно больших массивов, которые ежегодно приходилось пополнять.

Вычисление юлианских дат

Существует довольно много алгоритмов преобразования григорианских дат в юлианские, не использующих таблицы. Мы остановимся на трех из них. Алгоритм JD1 впервые был опубликован на алгоритмическом языке АЛГОЛ-60 в журнале американской ассоциации вычислительных машин (Fliegel H. F., Van Flandern T. C. A Machine Algorithm for Processing Calender Dates (algorithm 657). Communications of the ACM, 1968). Позднее он был переведен на ФОРТРАН (Spaeth H. Ausgewaehlte Operations Research-Algorithmen in FORTRAN. Wien, 1975) и послужил прообразом нашей функции JD1. Неформальное описание алгоритма JD2, ориентированное на использование ручных калькуляторов, приводится в книге Ж. Меёса "Астрономические формулы для калькуляторов" (М.: Мир, 1988). Наконец, алгоритм JD3 описан в предисловии редактора перевода книги Меёса. Основным недостатком указанных публикаций является отсутствие информации о допустимом временном интервале, на котором тот или иной алгоритм работает правильно. Для определения таких интервалов нам пришлось провести довольно громоздкий вычислительный эксперимент.

Алгоритм *JD1*

Функция JD1 предназначена для вычисления юлианской даты по году (Y), номеру месяца (M) и номеру дня месяца (D) на 12 часов полудня. Вместо неформального описания алгоритма JD1 ниже приводится текст соответствующей функции JD1.c, содержащей всего три строки. При переводе фортрановской программы потребовалась аккуратность с явным заданием типа промежуточных данных и некоторых констант.

```
long JD1(int Y,int M,int D)
{
  long q,j;
  j=(M-14)/12;
  q=Y+j+4800;
  return D-32075L+1461*q/4+367L*(M-2-12*j)/12-3*((q+100)/100)/4;
}
```

Численный эксперимент показал, что формулы алгоритма JD1 справедливы только для дат позднее 15.10.1582. В соответствующих оригинальных публикациях это ограничение не приводится. Возможно, что на практике большинству из нас и не придется обрабатывать более ранние даты, однако оговорка о сфере применимости алгоритма представляется нам обязательной.

Алгоритм *JD2*

Пусть григорианская дата представлена в виде двух целых чисел Y (номер года, который может быть как положительным, так и отрицательным для дат до но-

вой эры), `M` (номер месяца — число от 1 до 12) и вещественного числа `DT = D.T` (`D` — номер дня в месяце, `T` — часть текущего дня от 0,00000 до 0,99999, прошедшая от полуночи). Единственная тонкость в формулах Меёса заключается в том, что используемая им функция `int` означает выделение целой части без какого-либо округления в большую или меньшую сторону.

Если `M > 2`, то положим `y = Y` и `m = M`, в противном случае `y = Y - 1` и `m = M + 12`. Если григорианская дата приходится на момент после 15.10.1582, то вычисляются:

```
A = int(y/100)
B = 2 - A + int(A/4)
```

Если заданная дата предшествует моменту принятия григорианского календаря, то следует положить `B = 0`. Для правильной обработки отрицательных номеров года следует вычислить:

```
C = int(365.25*y)                 при y > 0
C = int(365.25 * y - 0.75)        при y < 0
```

Окончательная формула для нахождения `JD` имеет вид:

```
JD = C + int(30.6001 * (m+1)) + DT + 1720994.5 + B
```

Численные эксперименты показали, что функция `JD2` выдает правильные результаты на всем интервале дат, начиная с 1.01.4713 до н. э. (то есть от `Y = -4712`, `M = 1`, `DT = 1.5`) и до наших дней. Ее работа была проверена нами в цикле для 12 часов полудня каждого дня начального года. А затем — также в цикле по первому дню каждого последующего года с учетом смены обычных и високосных лет.

Алгоритм *JD3*

Алгоритм `JD3`, указанный в предисловии редактором перевода книги Меёса, привлекает своей простотой. В нем нет никаких логических проверок и реализующая его функция на языке Си имеет вид:

```
double JD3(int Y,int M,double DT)
{
  return 367.0*Y-floor(7.0*(Y+floor((M+9.)/12.))/4.)+
         floor(275.0*M/9.)+DT+1721013.5;
}
```

Однако первый же эксперимент с вычислением даты `JD`, приходящейся на 101,1600 (`JD3 = 2305445.0`), привел к расхождению с результатом Меёса (`JD2 = = 2305448.0`) на 3 дня. Более тщательный эксперимент показал, что функция `JD3` начинает выдавать правильные результаты с 1.03.1900. А заключительной датой, до которой значения `JD3` совпадают с `JD2`, является 29.02.2100.

Задачи, советы и ответы

Задание 9.01. Вычисления юлианских дат

Составить функцию JD, вычисляющую юлианскую дату по году (Y), номеру месяца (M), номеру дня и значению времени, заданным в виде вещественного числа DT (целая часть DT — порядковый номер дня в месяце, дробная часть DT — часть суток, прошедшая после полуночи). Проверки на правильность задания исходной информации исключены, чтобы не загромождать текст программы.

Совет 1 (общий)

По результатам довольно обширного численного эксперимента мы выяснили, что наиболее точный алгоритм представлен формулами Меёса. Поэтому тексты приводимых ниже программ используют алгоритм JD2. Если ваши задачи относятся к более узким временным интервалам, то текст функции JD можно заменить, используя один из выше описанных алгоритмов.

Программа 9_01.bas

```
DECLARE FUNCTION JD! (Y%, M%, DT!)
REM Перевод даты григорианского календаря
REM в юлианскую дату (алгоритм Меёса)
CLS
INPUT "Задайте год григорианского календаря : ",Y%
INPUT "Задайте месяц григорианского календаря : ",M%
INPUT "Задайте день и время : ",DT
PRINT USING "##### ### ###.#";Y%;M%;DT;
jd1=JD(Y%,M%,DT)
PRINT USING "  JD =#########.#####";jd1
END

FUNCTION JD(Y%,M%,DT)
' Y - григорианский год [-4713,9999]
' M - григорианский месяц [1,12]
' целая часть DT - день [1,31]
' дробная часть DT - время, прошедшее
' от начала текущего дня [.0,.99999]
' Возвращаемое значение - юлианская дата
' Целая часть JD -  число дней, прошедших
```

```
' от 1 января 4713 года до новой эры
' Дробная часть JD - время, прошедшее
' после 12 часов полудня дня JD
b%=0: yy%=Y%: mm%=M%
IF M%<3 THEN yy%=yy%-1: mm%=mm%+12
a%=yy% \ 100
IF Y%+M% / 100!+DT/10000 > 1582.1015# THEN
    b%=b%+2-a%+(a% \ 4)
END IF
c&=INT(365.25*yy%)
IF yy%<0 THEN c&=INT(365.25*yy%-.75)
JD=c&+INT(30.6001*(mm%+1))+DT+1720994.5#+b%
END FUNCTION
```

Программа 9_01.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double JD (int Y,int M,double DT);
  main()
{
  int Y,M,day;
  double DT,jd;
  clrscr();
  printf("\nЗадайте год григорианского календаря : ");
  scanf("%d",&Y);
  printf("\nЗадайте месяц григорианского календаря : ");
  scanf("%d",&M);
  printf("\nЗадайте день и время : ");
  scanf("%lf",&DT);
  jd=JD(Y,M,DT);
  printf("\nJD =%15.5f",jd);
  getch();
}
/*-------------------------------------------------------*/
double JD(int Y,int M,double DT)
/*****************************************/
/* Перевод даты григорианского календаря   */
```

```
/* в юлианскую дату (алгоритм Меёса)       */
/* Y - григорианский год [-4713,9999]      */
/* M - григорианский месяц [1,12]          */
/* целая часть DT - день [1,31]            */
/* дробная часть DT - время, прошедшее     */
/* от начала текущего дня [.0,.99999]      */
/* Возвращаемое значение - юлианская дата  */
/* Целая часть JD -  число дней, прошедших */
/* от 1 января 4713 года до новой эры      */
/* Дробная часть JD - время, прошедшее     */
/* после 12 часов полудня дня JD           */
/*****************************************/
{
  int a,b=0,m,y;
  long c;
  y=Y;
  m=M;
  if(M<3)  { y--; m += 12; }
  a=y/100;
  if(Y+M/100.+DT/10000 > 1582.1015)
     b += 2-a+(int)(a/4.);
  c=365.25*y;
  if(y<0)c=365.25*y-0.75;
  return c+(long)(30.6001*(m+1))+DT+1720994.5+b;
}
```

Программа 9_01.pas

```
program gd_to_jd;
uses Crt;
var
  Y,M,day:integer;
  DT,jd1:double;
function JD(Y,M:integer;DT:double):double;
  { Перевод даты григорианского календаря   }
  { в юлианскую дату (алгоритм Меёса)       }
  { Y - григорианский год [-4713,9999]      }
  { M - григорианский месяц [1,12]          }
  { целая часть DT - день [1,31]            }
  { дробная часть DT - время, прошедшее     }
```

```
  { от начала текущего дня [.0,.99999]      }
  { Возвращаемое значение - юлианская дата  }
  { Целая часть JD -  число дней, прошедших }
  { от 1 января 4713 года до новой эры      }
  { Дробная часть JD - время, прошедшее     }
  { после 12 часов полудня дня JD           }
var
  a,b,mm,yy:integer;
  c:longint;
begin
  b:=0;
  yy:=Y;
  mm:=M;
  if M<3 then begin yy:=yy-1; mm:=mm+12; end;
  a:=yy div 100;
  if (Y+M/100.+DT/10000)>1582.1015  then
     b:=b+2-a+(a div 4);
  c:=trunc(365.25*yy);
  if yy<0 then c:=trunc(365.25*yy-0.75);
  JD:=c+trunc(30.6001*(mm+1))+DT+1720994.5+b;
end;
begin
  clrscr;
  write('Задайте год григорианского календаря : ');
  readln(Y);
  write('Задайте месяц григорианского календаря : ');
  readln(M);
  write('Задайте день и время : ');
  readln(DT);
  write(Y:4,M:3, ' ',DT:3:1);
  jd1:=JD(Y,M,DT);
  writeln('  JD =',jd1:15:5);
  readln;
end.
```

Задание 9.02. Вычисление дат григорианского календаря

Составить подпрограмму `GD`, вычисляющую дату григорианского календаря по юлианской дате, представленной вещественным числом `JD`.

Совет 1 (общий)

Учитывая высокое качество алгоритмов Меёса, воспользуемся его же алгоритмом для обратного преобразования дат. Мы тщательно проверили его на большом количестве прямых и обратных преобразований григорианских и юлианских дат. В приведенных ниже формулах сохранены обозначения Меёса.

Вычислим `JD+0.5` и обозначим через `Z` целую часть, а через `F` — дробную.

Если `Z ≥ 2299161`, то вычислим:

```
a=int((Z-1867216.25)/36524.25)
A=Z+1+a-int(a/4)
```

при `Z < 2299161` положим

```
A=Z
B=A+1524
C=int((B-122.1)/365.25
D=int(365.25*C)
E=int((B-D)/30.6001)
DT=B-D-int(30.6001*E)
```

Номер месяца

`M=E-1`, если `E<13.5`

`M=E-13`, если `E>13.5`

Номер года

`Y=C-4716`, если `M>2.5`

`Y=C-4715`, если `M<2.5`

Программа 9_02.bas

```
DECLARE SUB GD(jd!,y%,m%,DT!)
REM Преобразование юлианской даты JD в дату григорианского
REM календаря - Y (год), M (месяц),
REM DT (день и часть времени, прошедшую от начала суток
CLS
INPUT "Задайте юлианскую дату : ",jd
GD jd,y%,m%,DT
PRINT USING "год = ##### месяц = ## день и время = ###.#";y%;m%;DT
END

SUB GD(jd,y%,m%,DT)
' Восстановление даты григорианского календаря
' по юлианской дате jd. Результаты заносятся в
```

```
' y% - год
' m% - месяц
' dt - день (целая часть) и время дня (дробная часть)
DIM A AS LONG, B AS LONG, C AS LONG, D AS LONG, Z AS LONG
DIM aa AS LONG, E AS INTEGER, F AS DOUBLE
  Z=INT(jd+.5)
  F=jd+.5-Z
  A=Z
  IF Z>=2299161 THEN
     aa=INT((Z-1867216.25#)/36524.25)
     A=A+1+aa-(aa\4)
  END IF
  B=A+1524
  C=INT((B-122.1)/365.25)
  D=INT(365.25*C)
  E=INT((B-D)/30.6001)
  DT=B-D-INT(30.6001*E)+F
  IF E<13.5 THEN m%=E-1 ELSE m%=E-13
  IF m%>2.5 THEN y%=C-4716 ELSE y%=C-4715
END SUB
```

Программа 9_02.c

```
#include <stdio.h>
#include <conio.h>
void GD(double jd,int *y,int *m,double *dt);
  main()
{
  int Y,M;
  double DT,jd;
  clrscr();
  printf("\nЗадайте юлианскую дату : ");
  scanf("%lf",&jd);
  GD(jd,&Y,&M,&DT);
  printf("\nгод=%d месяц=%d день и время=%-15.5lf",Y,M,DT);
  getch();
}
/*-------------------------------------------*/
void GD(double jd,int *y,int *m,double *dt)
```

```
/*************************************/
/* Восстановление даты григорианского */
/* календаря по юлианской дате jd     */
/* y - год,      m - месяц            */
/* dt - день (целая часть)            */
/* и время дня (дробная часть)        */
/*************************************/
{
  long a,A,B,C,D,E,Z;
  double F;
  Z=jd+0.5;
  F=jd+0.5-Z;
  A=Z;
  if(Z>=2299161)
    {
      a=(Z-1867216.25)/36524.25;
      A += 1+a-a/4;
    }
  B=A+1524;
  C=(B-122.1)/365.25;
  D=365.25*C;
  E=(B-D)/30.6001;
  *dt=B-D-(int)(30.6001*E)+F;
  if(E<13.5) *m=E-1; else *m=E-13;
  if(*m>2.5) *y=C-4716; else *y=C-4715;
}
```

Программа 9_02.pas

```
program jd_to_gd;
{ Преобразование юлианской даты JD в дату григорианского
   календаря - Y (год), M (месяц), DT (день и часть времени,
   прошедшую от начала суток }
uses crt;
var
  Y,M:integer;
  DT,jd:double;

procedure GD(jd:double; var y,m:integer; var dt:double);
  { Восстановление даты григорианского }
```

```
  { календаря по юлианской дате jd     }
  { y - год, m - месяц                 }
  { dt - день (целая часть)            }
  { и время дня (дробная часть)        }
var
  aa,A,B,C,D,E,Z:longint;
  F:double;
begin
  Z:=trunc(jd+0.5);
  F:=jd+0.5-Z;
  A:=Z;
  if Z>=2299161 then begin
    aa:=trunc((Z-1867216.25)/36524.25);
    A:=A+1+aa-(aa div 4);
  end;
  B:=A+1524;
  C:=trunc((B-122.1)/365.25);
  D:=trunc(365.25*C);
  E:=trunc((B-D)/30.6001);
  dt:=B-D-trunc(30.6001*E)+F;
  if E<13.5 then m:=E-1 else m:=E-13;
  if m>2.5 then y:=C-4716 else y:=C-4715;
end;
begin
  clrscr;
  write('Задайте юлианскую дату : ');
  readln(jd);
  GD(jd,Y,M,DT);
  writeln('год = ',Y:4,' месяц = ',M:2,' день и время = ',DT:4:1);
  readln;
end.
```

Задание 9.03. Интервал времени между двумя датами

Составить функцию `dif_time`, вычисляющую интервал между двумя датами григорианского календаря.

Совет 1 (общий)

Преобразуйте даты григорианского календаря и найдите разницу между соответствующими юлианскими датами. Чтобы не выяснять, какая из дат является предшествующей, можно найти модуль разности.

Программа 9_03.bas

```
DECLARE FUNCTION JD! (Y%, M%, DT!)
DECLARE FUNCTION DifTime! (Y1%, M1%, DT1!, Y2%, M2%, DT2!)
REM Определение интервала времени между двумя датами
REM григорианского календаря : (Y1,M1,DT1) и (Y2,M2,DT2)
CLS
INPUT "Задайте год григорианского календаря : ",Y1%
INPUT "Задайте месяц григорианского календаря : ",M1%
INPUT "Задайте день и время : ",DT1
INPUT "Задайте год григорианского календаря : ",Y2%
INPUT "Задайте месяц григорианского календаря : ",M2%
INPUT "Задайте день и время : ",DT2
PRINT "Интервал = ",DifTime(Y1%,M1%,DT1,Y2%,M2%,DT2)
END

FUNCTION DifTime (Y1%,M1%,DT1,Y2%,M2%,DT2)
' Определение интервала времени между
' двумя григорианскими датами
' Y1,Y2 - годы   M1,M2 - месяцы
' целые части DT1, DT2 - дни
' дробные части DT1, DT2 - время дня
  jd1=JD(Y1%,M1%,DT1)
  jd2=JD(Y2%,M2%,DT2)
  DifTime=ABS(jd1-jd2)
END FUNCTION

FUNCTION JD(Y%,M%,DT)
  DIM a AS INTEGER, b AS INTEGER, mm AS INTEGER, yy AS INTEGER
  DIM c AS LONG
  b=0
  yy=Y%
  mm=M%
  IF M%<3 THEN yy=yy-1: mm=mm+12
```

```
  a=yy\100
  IF Y%+M%/100!+DT/10000>1582.1015# THEN
     b=b+2-a+a\4
  END IF
  c=INT(365.25*yy)
  IF yy<0 THEN c=INT(365.25*yy-.75)
  JD=c+INT(30.6001*(mm+1))+DT+1720994.5#+b
END FUNCTION
```

Программа 9_03.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

double dif_time(int Y1,int M1,double DT1,
                int Y2,int M2,double DT2);
double JD(int Y,int M,double DT);

  main()
{
  int Y1,Y2,M1,M2;
  double DT1,DT2;
  clrscr();
  printf("\nЗадайте год григорианского календаря : ");
  scanf("%d",&Y1);
  printf("\nЗадайте месяц григорианского календаря : ");
  scanf("%d",&M1);
  printf("\nЗадайте день и время : ");
  scanf("%lf",&DT1);
  printf("\nЗадайте год григорианского календаря : ");
  scanf("%d",&Y2);
  printf("\nЗадайте месяц григорианского календаря : ");
  scanf("%d",&M2);
  printf("\nЗадайте день и время : ");
  scanf("%lf",&DT2);
  printf("\nИнтервал = %lf",dif_time(Y1,M1,DT1,Y2,M2,DT2));
  getch();
}
```

```
/*-------------------------------------*/
double dif_time(int Y1,int M1,double DT1,
                int Y2,int M2, double DT2)
/*************************************/
/* Определение интервала времени между */
/* двумя григорианскими датами         */
/* Y1,Y2 - годы   M1,M2 - месяцы       */
/* целые части DT1, DT2 - дни          */
/* дробные части DT1, DT2 - время дня  */
/***************************************/

{
  return fabs(JD(Y1,M1,DT1)-JD(Y2,M2,DT2));
}
/*-----------------------------------------------*/
double JD(int Y,int M, double DT)
{
  int a,b=0,m,y;
  long c;
  y=Y;
  m=M;
  if (M<3)  { y--; m += 12; }
  a=y/100;
  if(Y+M/100.+DT/10000 > 1582.1015)
    b += 2-a+(int)(a/4.);
  c=365.25*y;
  if(y<0) c=365.25*y-0.75;
  return c+(long)(30.6001*(m+1))+DT+1720994.5+b;
}
```

Программа 9_03.pas

```
program gd_to_jd;
{
   Определение интервала времени между двумя датами
   григорианского календаря : (Y1,M1,DT1) и (Y2,M2,DT2)
}
uses Crt;
var
```

```
  Y1,M1,Y2,M2:integer;
  DT1,DT2:double;
function JD(Y,M:integer; DT:double):double;
var
  a,b,mm,yy:integer;
  c:longint;
begin
  b:=0;
  yy:=Y;
  mm:=M;
  if M<3 then begin yy:=yy-1; mm:=mm+12; end;
  a:=yy div 100;
  if (Y+M/100.+DT/10000)>1582.1015  then
     b:=b+2-a+(a div 4);
  c:=trunc(365.25*yy);
  if yy<0 then c:=trunc(365.25*yy-0.75);
  JD:=c+trunc(30.6001*(mm+1))+DT+1720994.5+b;
end;
function dif_time(Y1,M1:integer; DT1:double;
                  Y2,M2:integer; DT2:double):double;
  { Определение интервала времени между }
  { двумя григорианскими датами         }
  { Y1,Y2 - годы   M1,M2 - месяцы       }
  { целые части DT1, DT2 - дни          }
  { дробные части DT1, DT2 - время дня  }
begin
  dif_time:=abs(JD(Y1,M1,DT1)-JD(Y2,M2,DT2));
end;
begin
  clrscr;
  write(' Задайте год григорианского календаря : ');
  readln(Y1);
  write(' Задайте месяц григорианского календаря : ');
  readln(M1);
  write(' Задайте день и время : ');
  readln(DT1);
  write(' Задайте год григорианского календаря : ');
  readln(Y2);
  write(' Задайте месяц григорианского календаря : ');
```

```
  readln(M2);
  write(' Задайте день и время : ');
  readln(DT2);
  writeln('Интервал = ',dif_time(Y1,M1,DT1,Y2,M2,DT2):10:0);
  readln;
end.
```

Задание 9.04. День недели

Составить функцию `week_day`, определяющую день недели по дате григорианского календаря.

Совет 1 (общий)

Преобразуйте григорианскую дату в юлианский день и воспользуйтесь довольно очевидной формулой:

```
день недели=(JD+1.5) mod 7
```

В ее справедливости нетрудно убедиться, вспомнив, что понедельник 1 января условной начальной точки отсчета универсального времени (год 4712 до н. э.) начинался в момент `JD=-0.5`. Приведенная выше формула дает 0 для воскресенья, 1 — для понедельника, 2 — для вторника, ..., 6 — для субботы.

Совет 2 (общий)

Функция `week_day1` построена на базе аналогичной фортрановской программы, и сфера ее применения распространяется на даты позднее 15.10.1582.

Программа 9_04.bas

```
DECLARE FUNCTION JD!(Y%,M%,DT!)
DECLARE FUNCTION WeekDay!(Y%,M%,D%)
REM Определение дня недели
CLS
INPUT "Задайте год григорианского календаря : ",Y%
INPUT "Задайте месяц григорианского календаря : ",M%
INPUT "Задайте день : ",D%
PRINT "день недели = ";WeekDay(Y%, M%, D%)
END

FUNCTION JD(Y%,M%,DT)
  DIM a AS INTEGER, b AS INTEGER, mm AS INTEGER, yy AS INTEGER
  DIM c AS LONG
  b=0
```

```
  yy=Y%
  mm=M%
  IF M%<3 THEN yy=yy-1: mm=mm+12
  a=yy\100
  IF Y%+M%/100!+DT/10000>1582.1015# THEN
     b=b+2-a+a\4
  END IF
  c=INT(365.25*yy)
  IF yy<0 THEN c=INT(365.25*yy-.75)
  JD=c+INT(30.6001*(mm+1))+DT+1720994.5#+b
END FUNCTION

FUNCTION WeekDay(Y%,M%,D%)
' Определение дня недели
' Y - григорианский год
' M - григорианский месяц (1 - 12)
' D - день (1 - 31)
' Возвращаемое значение :
' 0 - воскр., 1 - понед., 2 - вторник,...
  WeekDay=INT(JD(Y%,M%,D%+1.5)) MOD 7
END FUNCTION
```

Программа 9_04.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int week_day(int Y,int M,int D);
double JD(int Y,int M,double DT);

  main()
{
  int Y,M,D,day;
  clrscr();
  printf("\nЗадайте год григорианского календаря : ");
  scanf("%d",&Y);
  printf("\nЗадайте месяц григорианского календаря : ");
  scanf("%d",&M);
```

```
  printf("\nЗадайте день : ");
  scanf("%d",&D);
  day=week_day(Y,M,D);
  printf("\nдень недели = %d",day);
  getch();
}
/*-----------------------------------------------*/
int week_day(int Y,int M,int D)
/*****************************************/
/* Вычисление дня недели                  */

/* Возвращаемое значение :                */
/* 0 - воскр., 1 - понед., 2 - вторник,... */
/*****************************************/
{
  return (long)(JD(Y,M,(double)D)+1.5) % 7;
}
/*-----------------------------------------------*/
double JD(int Y,int M,double DT)
int a,b=0,m,y;
  long c;
  y=Y;
  m=M;
  if (M<3)  { y--; m += 12; }
  a=y/100;
  if(Y+M/100.+DT/10000 > 1582.1015)
     b += 2-a+(int)(a/4.);
  c=365.25*y;
  if(y<0)c=365.25*y-0.75;
  return c+(long)(30.6001*(m+1))+DT+1720994.5+b;
}
```

Функция week_day1.c

```
int week_day1(int Y, int M, int D)
{
/*************************************/
/* Вычисление дня недели              */
/* Возвращаемые значения:             */
```

```
/* 1 - пон., 2 - вторник,..., 7 -воскр */
/*************************************/
  int q,m;
  q=M+10;
  m=(M-14)/12+Y;
  return ((13*(q-(q/13)*12)-1)/5+D+77+5*(m%100)/4+
                                m/400-(m/100)*2)%7;
}
```

Программа 9_04.pas

```
program WeekDay;
{ Определение дня недели }
uses Crt;
var
  Y,M,D,day:integer;
function JD(Y,M:integer; DT:double):double;
var
  a,b,mm,yy:integer;
  c:longint;
begin
  b:=0;
  yy:=Y;
  mm:=M;
  if M<3 then begin yy:=yy-1; mm:=mm+12; end;
  a:=yy div 100;
  if (Y+M/100.+DT/10000)>1582.1015  then
     b:=b+2-a+(a div 4);
  c:=trunc(365.25*yy);
  if yy<0 then c:=trunc(365.25*yy-0.75);
  JD:=c+trunc(30.6001*(mm+1))+DT+1720994.5+b;
end;
function week_day(Y,M,D:integer):integer;
  { Определение дня недели                  }
  { Y - григорианский год                   }
  { M - григорианский месяц (1 - 12)        }
  { D - день (1 - 31)                       }
  { Возвращаемое значение :                 }
  { 0 - воскр., 1 - понед., 2 - вторник,... }
```

```
begin
  week_day:=trunc(JD(Y,M,D+1.5)) mod 7;
end;

begin
  clrscr;
  write(' Задайте год григорианского календаря : ');
  readln(Y);
  write(' Задайте месяц григорианского календаря : ');
  readln(M);
  write(' Задайте день : ');
  readln(D);
  day:=week_day(Y,M,D);
  writeln('день недели = ',day);
  readln;
end.
```

Задание 9.05. Порядковый день в году

Составить функцию `dat_to_ord`, вычисляющую порядковый номер дня в году по его дате.

Совет 1 (общий)

Воспользуемся алгоритмом, описанным в книге Меёса.

Для обычного года:

```
N=int(275*M/9)-2*int((M+9)/12)+D-30
```

Для високосного года:

```
N=int(275*M/9)-int((M+9)/12)+D-30
```

Программа 9_05.bas

```
DECLARE FUNCTION DatToOrd!(y%,M%,D%)
REM Определение порядкового дня в году по
REM текущей дате (y - год, m -месяц, d - день)
CLS
INPUT "Задайте год : ",y%
INPUT "Задайте месяц : ",M%
INPUT "Задайте день : ",D%
PRINT "Порядковый номер дня в году = ";DatToOrd(y%,M%,D%)
END
```

```
FUNCTION DatToOrd(y%,M%,D%)
  DIM a1 AS INTEGER, a2 AS INTEGER
  a1=(M%+9)\12
  a2=INT(275!*M%/9!)-a1+D%
  IF (y% MOD 400=0) OR ((y MOD 4=0) AND (y MOD 100<>0)) THEN
    DatToOrd=a2-30
  ELSE DatToOrd=a2-a1-30
  END IF
END FUNCTION
```

Программа 9_05.c

```
#include <stdio.h>
#include <conio.h>
int dat_to_ord(int y,int m,int d);
  main()
{
  int Y,M,D;
  clrscr();
  printf("\nЗадайте год : ");
  scanf("%d",&Y);
  printf("\nЗадайте месяц : ");
  scanf("%d",&M);
  printf("\nЗадайте день : ");
  scanf("%d",&D);
  printf("\nПорядковый номер дня в году = %d", dat_to_ord(Y,M,D));
  getch();
}
/*----------------------------------------------*/
int dat_to_ord(int y,int m,int d)
{
 int a1,a2;
 a1=(m+9.0)/12.0;
 a2=(275.0*m/9.0)-a1+d-30;
 if((y%400==0)||((y%4==0)&&(y%100!=0))) return a2;
 return a2-a1;
}
```

Программа 9_05.pas

```
program ord_day;
uses Crt;
var
  Y,M,D:integer;
function dat_to_ord(y,m,d:integer):integer;
{
 Определение порядкового дня в году по
 текущей дате (y - год, m - месяц, d - день)
}
var
  a1,a2:integer;
begin
  a1:=(m+9) div 12;
  a2:=trunc(275.0*m/9.0)-a1+d-30;
  if((y mod 400 = 0) OR ((y mod 4 = 0) AND (y mod 100 <>0))) then
    dat_to_ord:=a2
  else dat_to_ord:=a2-a1;
end;
begin
  clrscr;
  write('Задайте год : ');
  readln(Y);
  write('Задайте месяц : ');
  readln(M);
  write('Задайте день : ');
  readln(D);
  writeln('Порядковый номер дня в году = ', dat_to_ord(Y,M,D));
  readln;
end.
```

Задание 9.06. Восстановление даты

Составить процедуру `month_day`, вычисляющую номера месяца (`M`) и дня (`D`) по порядковому номеру (`N`) дня в году.

Совет 1 (общий)

Воспользуемся алгоритмом, описанным в книге Меёса. Положим, что `A = 1889` для обычного года и `A = 1523` — для високосного.

Вычислим :

```
B=int((N+A-122.1)/365.25)
C=N+A-int(365.25*B)
E=int(C/30.6001)
M=E-1          при  E<13.5
M=E-13         при  E>13.5
D=C-int(30.6001*E)
```

Программа 9_06.bas

```
DECLARE SUB MonthDay(Y%,OrdDay%,M%,D%)
REM Восстановление даты по порядковому дню года
CLS
INPUT "Задайте год : ",Y%
INPUT "Задайте порядковый номер дня в году : ",OrdDay%
MonthDay Y%,OrdDay%,M%,D%
PRINT "Ему соответствует месяц = ",M%;" и день = ";D%
END

SUB MonthDay(Y%,OrdDay%,M%,D%)
' Восстановление месяца и дня в году Y%
' по порядковому номеру дня ord_day%
' M% - сюда засылается номер месяца
' D% - сюда засылается номер дня месяца
  DIM A AS INTEGER, B AS INTEGER, C AS INTEGER, E AS INTEGER
  A=1889
  IF (Y% MOD 400=0)OR((Y% MOD 4=0)AND(Y% MOD 100<>0)) THEN A=1523
  B=INT((OrdDay%+A-122.1)/365.25)
  C=OrdDay%+A-INT(365.25*B)
  E=INT(C/30.6001)
  IF E<13.5 THEN M%=E-1 ELSE M%=E-13
  D%=C-INT(30.6001*E)
END SUB
```

Программа 9_06.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
void month_day(int Y,int ord_day,int *M,int *D);

  main()
{
  int Y,M,D,ord_day;
  clrscr();
  printf("\nЗадайте год : ");
  scanf("%d",&Y);
  printf("\nЗадайте порядковый номер дня в году : ");
  scanf("%d",&ord_day);
  month_day(Y,ord_day,&M,&D);
  printf("\nЕму соответствует месяц = %d и день = %d",M,D);
  getch();
}
/*-----------------------------------------------*/
void month_day(int Y,int ord_day,int *M,int *D)
/****************************************/
/* Восстановление месяца и дня в году Y */
/* по порядковому номеру дня ord_day    */
/* M - сюда засылается номер месяца     */
/* D - сюда засылается номер дня месяца */
/****************************************/
{
  int A=1889,B,C,E;
  if((Y%400==0)||((Y%4==0)&&(Y%100!=0))) A=1523;
  B=(ord_day+A-122.1)/365.25;
  C=ord_day+A-floor(365.25*B);
  E=C/30.6001;
  if(E<13.5) *M=E-1;   else *M=E-13;
  *D=C-floor(30.6001*E);
}
```

Программа 9_06.pas

```
program ord_to_dat;
uses Crt;
var
  Y,M,D,ord_day:integer;
```

```
procedure month_day(Y,ord_day:integer;var M,D:integer);
  { Восстановление месяца и дня в году Y }
  { по порядковому номеру дня ord_day    }
  { M - сюда засылается номер месяца     }
  { D - сюда засылается номер дня месяца }
var
  A,B,C,E:integer;
begin
  A:=1889;
  if((Y mod 400 = 0) OR ((Y mod 4 = 0)AND(Y mod 100 <>0))) then
    A:=1523;
  B:=trunc((ord_day+A-122.1)/365.25);
  C:=ord_day+A-trunc(365.25 * B);
  E:=trunc(C/30.6001);
  if(E<13.5) then M:=E-1   else M:=E-13;
  D:=C-trunc(30.6001*E);
end;
begin
  clrscr;
  write('Задайте год : ');
  readln(Y);
  write('Задайте порядковый номер дня в году : ');
  readln(ord_day);
  month_day(Y,ord_day,M,D);
  writeln('Ему соответствует месяц = ',M, ' и день = ',D);
  readln;
end.
```

Задание 9.07. Количество дней в месяце

Составить функцию `max_day`, определяющую количество дней по номеру месяца.

Совет 1 (общий)

Наверное, самый простой алгоритм основан на выборке нужного числа из предварительно подготовленного массива с учетом коррекции количества февральских дней в високосном году. Високосный год должен делиться без остатка на 400 или делиться на 4, но не делиться на 100.

Программа 9_07.bas

```
DECLARE FUNCTION MaxDay!(Y%,M%)
REM Определение количества дней в месяце
DATA 31,28,31,30,31,30,31,31,30,31,30,31
DIM SHARED DAYS(1 TO 12)
FOR J=1 TO 12: READ DAYS(J): NEXT J
CLS
INPUT "Задайте год : ",Y%
INPUT "Задайте месяц : ",M%
PRINT "Число дней в этом месяце = ";MaxDay(Y%,M%)
END

FUNCTION MaxDay(Y%,M%)
  MaxDay=DAYS(M%)
  IF M%<>2 THEN EXIT FUNCTION
  IF (Y% MOD 400=0)OR((Y% MOD 4=0)AND(Y% MOD 100<>0)) THEN
      MaxDay=29
  END IF
END FUNCTION
```

Программа 9_07.c

```
#include <stdio.h>
#include <conio.h>
int max_day(int Y, int M);

  main()
{
  int Y,M;
  printf("\nЗадайте год : ");
  scanf("%d",&Y);
  printf("\Задайте месяц : ");
  scanf("%d",&M);
  printf("\nЧисло дней в этом месяце = %d",max_day(Y,M));
  getch();
}
/*-------------------------------------------------*/
int max_day(int Y, int M)
```

```
{ int days[]={31,28,31,30,31,31,30,31,30,31,30,31};
  if(M != 2) return days[M-1];
  if((Y%400==0)|||((Y%4==0)&&(Y%100!=0))) return 29;
  return 28;
}
```

Программа 9_07.pas

```
program days_in_month;
{ Определение количества дней в месяце }
var
  Y,M:integer;
function max_day(Y,M:integer):integer;
const
 days:array[1..12] of byte=(31,28,31,30,31,30,31,31,30,31,30,31);
begin
  max_day:=days[M];
  if(M<>2) then exit;
  if((Y mod 400=0)OR((Y mod 4=0)AND(Y mod 100<>0))) then
    max_day:=29;
end;
begin
  write('Задайте год : ');
  readln(Y);
  write('Задайте месяц : ');
  readln(M);
  writeln('Число дней в этом месяце = ',max_day(Y,M));
  readln;
end.
```

Задание 9.08. Календарь

Составить процедуру `calendar`, отображающую на экране дисплея календарь на заданный месяц любого года.

Совет 1 (общий)

Для понимания приведенных ниже программ достаточно разобраться с тремя моментами.

Во-первых, календарь любого месяца можно разместить в таблице, содержащей 7 строк (по числу дней недели) и 6 столбцов. В худшем варианте 1-е число

месяца выпадает на воскресенье, занимая тем самым последнюю клетку 1-го столбца, а два последних дня месяца — 30 и 31 — на понедельник и вторник последнего столбца. Двумерную таблицу календаря удобнее представить в виде одномерного массива из 42 элементов.

Во-вторых, для определения местоположения 1-го дня месяца мы можем обратиться к функции `week_day`.

Наконец, с помощью функции `max_day` мы можем определить число дней в заданном месяце. После этого остается очистить нулями массив из 42 элементов и расписать его с нужного места последовательными днями месяца. А затем отобразить на экране ненулевые элементы в соответствующих позициях экрана.

Программа 9_08.bas

```
DECLARE SUB calendar(Y%,M%)
DECLARE FUNCTION JD!(Y%,M%,DT!)
DECLARE FUNCTION WeekDay!(Y%,M%,D%)
DECLARE FUNCTION MaxDay!(Y%,M%)
REM  Программа вывода календаря на любой месяц
DATA 31,28,31,30,31,30,31,31,30,31,30,31
DIM SHARED DAYS(1 TO 12)
FOR j=1 TO 12: READ DAYS(j): NEXT j
CLS
INPUT "Задайте год : ",Y%
INPUT "Задайте месяц : ",M%
calendar Y%,M%
END

SUB calendar(Y%,M%)
DIM i AS INTEGER, j AS INTEGER, k AS INTEGER, q AS INTEGER
DIM a(42),b$(7)
b$(0)="понедельник": b$(1)="вторник    ": b$(2)="среда      "
b$(3)="четверг    ": b$(4)="пятница    ": b$(5)="суббота    "
b$(6)="воскресенье"
CLS
PRINT "Календарь на ";M%;" месяц ";Y%;" года"
i=WeekDay(Y%,M%,1)
IFi=0 THEN i=7
q=MaxDay(Y%,M%)
FOR j=0 TO 41: a(j)=0: NEXT j
k=1
FOR j=i-1 TO q+i-2: a(j)=k: k=k+1: NEXT j
```

```
FOR j=0 TO 6
  LOCATE j+3,10: PRINT b$(j);
  k=0
  WHILE k<=35
    IF a(k+j)<>0 THEN
       PRINT USING "####";a(k+j);
    ELSE PRINT "    ";
    END IF
    k=k+7
  WEND
NEXT j
END SUB

FUNCTION JD(Y%,M%,DT)
  DIM a AS INTEGER, b AS INTEGER, mm AS INTEGER, yy AS INTEGER
  DIM c AS LONG
  b=0
  yy=Y%
  mm=M%
  IF M%<3 THEN yy=yy-1: mm=mm+12
  a=yy\100
  IF Y%+M%/100!+DT/10000>1582.1015# THEN
     b=b+2-a+a\4
  END IF
  c=INT(365.25*yy)
  IF yy<0 THEN c=INT(365.25*yy-.75)
  JD=c+INT(30.6001*(mm+1))+DT+1720994.5#+b
END FUNCTION

FUNCTION MaxDay(Y%,M%)
  MaxDay=DAYS(M%)
  IF M%<>2 THEN EXIT FUNCTION
  IF (Y% MOD 400=0)OR((Y% MOD 4=0)AND(Y% MOD 100<>0)) THEN
      MaxDay=29
  END IF
END FUNCTION

FUNCTION WeekDay(Y%,M%,D%)
  d1!=D%+1.5
```

```
  WeekDay=INT(JD(Y%,M%,d1!)) MOD 7
END FUNCTION
```

Программа 9_08.c

```
#include <stdio.h>
#include <conio.h>

void calendar(int Y,int M);
int week_day(int Y,int M,int D);
double JD(int Y,int M,double DT);
int max_day(int Y,int M);

  main()
{
   int Y,M;
   clrscr();
   printf("Задайте год : ");
   scanf("%d",&Y);
   printf("Задайте месяц : ");
   scanf("%d",&M);
   calendar(Y,M);
   getch();
}
/*--------------------------------------------------*/
void calendar(int Y,int M)
{
  int a[42],i,j,k,q;
  char    *b[7]={"понедельник",
                 "вторник    ",
                 "среда      ",
                 "четверг    ",
                 "пятница    ",
                 "суббота    ",
                 "воскресенье"};
  clrscr();
  printf("\t Календарь на %d месяц %d года",M,Y);
  i=week_day(Y,M,1);  /* определение дня недели для 1.M.Y */
  if(i==0)i=7;
```

```
  q=max_day(Y,M);    /* определение количества дней в месяце */
  for(j=0; j<42; j++) a[j]=0;
  for(j=i-1,k=1; j<q+i-1; j++,k++) a[j]=k;
  for(j=0; j<7; j++)
    {
      gotoxy(10,j+3); printf("%s",b[j]);
      for(k=0; k<=35; k+=7)
        if(a[k+j] != 0) printf("%4d",a[k+j]);
                   else printf("%4c",' ');
    }
  return;
}
/*----------------------------------------------------*/
double JD(int Y,int M,double DT)
int a,b=0,m,y;
  long c;
  y=Y;
  m=M;
  if (M<3)  { y--; m += 12; }
  a = y/100;
  if(Y+M/100.+DT/10000 > 1582.1015)
     b += 2-a+(int)(a/4.);
  c=365.25*y; if(y<0)c=365.25*y-0.75;
  return c+(long)(30.6001*(m+1))+DT+1720994.5+b;
}
/*----------------------------------------------------*/
int week_day(int Y,int M,int D)
{
int q,m;
  q = M+10;
  m = (M-14)/12 + Y;
  return ((13*(q-(q/13)*12)-1)/5+D+77+5*(m%100)/4+m/400-(m/100)*2)%7;
}

/*----------------------------------------------------*/
int max_day(int Y,int M)
{ int days[]={31,28,31,30,31,31,30,31,30,31,30,31};
  if(M != 2) return days[M-1];
```

```
  if((Y%400==0)||((Y%4==0)&&(Y%100!=0))) return 29;
  return 28;
}
```

Программа 9_08.pas

```
program calend;
{ Программа вывода календаря на любой месяц }
uses Crt;
var
   Y,M:integer;

function JD(Y,M:integer; DT:double):double;
var
  a,b,mm,yy:integer;
  c:longint;
begin
  b:=0;
  yy:=Y;
  mm:=M;
  if M<3 then begin yy:=yy-1; mm:=mm+12; end;
  a:=yy div 100;
  if (Y+M/100.+DT/10000)>1582.1015  then
     b:=b+2-a+(a div 4);
  c:=trunc(365.25*yy);
  if yy<0 then c:=trunc(365.25*yy-0.75);
  JD:=c+trunc(30.6001*(mm+1))+DT+1720994.5+b;
end;

function week_day(Y,M,D:integer):integer;
var
  d1:double;
begin
  d1:=D+1.5;
  week_day:=round(JD(Y,M,d1)) mod 7;
end;

function max_day(Y,M:integer):byte;
const
 days:array [1..12] of byte =(31,28,31,30,31,30,31,31,30,31,30,31);
```

```
begin
  max_day:=days[M];
  if (M=2)and((Y mod 400=0)or((Y mod 4=0)and(Y mod 100<>0))) then
  max_day:=29;
end;

procedure calendar(Y,M:integer);
var
  i,j,k,q:integer;
  a:array [0..41] of byte;
const
  b:array [0..6] of string=('понедельник',
                            'вторник    ',
                            'среда      ',
                            'четверг    ',
                            'пятница    ',
                            'суббота    ',
                            'воскресенье ');
begin
  clrscr;
  write('          Календарь на ',M, ' месяц ',Y, ' года');
  i:=week_day(Y,M,1);
  if i=0 then i:=7;
  q:=max_day(Y,M);
  for j:=0  to 41 do a[j]:=0;
  k:=1;
  for j:=i-1 to q+i-2 do begin a[j]:=k; inc(k); end;
  for j:=0 to 6 do
    begin
      gotoxy(10,j+3); write(b[j]);
      k:=0;
      while k<= 35 do
        begin;
          if a[k+j] <> 0 then write(a[k+j]:4)
             else write(' ':4);
          k:=k+7;
        end;
    end;
end;
```

```
begin {main}
   clrscr;
   write('Задайте год : ');
   readln(Y);
   write('Задайте месяц : ');
   readln(M);
   calendar(Y,M);
   readln;
end.
```

Задание 9.09. Преобразования времени суток

Составить процедуры упаковки и распаковки времени суток. Исходными данными для их работы должны быть либо показания часов (`H` — часы, `M` — минуты, `S` — секунды), либо часть суток, представленная вещественным числом `t` (`0 ≤ t ≤ 1`).

Программа 9_09.bas

```
DECLARE SUB UnpackTime(T!,H%,M%,S%)
DECLARE FUNCTION PackTime!(H%,M%,S%)
REM Упаковка и распаковка показаний часов
CLS
H%=18: M%=0: S%=0
T=PackTime(H%,M%,S%)
PRINT USING "##:##:## = ###.##";H%;M%;S%;T
T=.5
UnpackTime T,H%,M%,S%
PRINT USING "##:##:## = ###.##";H%;M%;S%;T
END

FUNCTION PackTime(H%,M%,S%)
' Упаковка времени - перевод часов (H%)
'  минут (M%) и секунд (S%) в часть суток
  PackTime = (H% * 3600! + M% * 60 + S%) / 86400!
END FUNCTION

SUB UnpackTime(T,H%,M%,S%)
' Распаковка времени - перевод части суток
' в часы (H%), минуты (M%) и секунды (S%)
```

```
  t1!=T*86400
  H%=INT(t1!/3600!)
  t1!=t1!-3600!*H%
  M%=INT(t1!/60!)
  S%=INT(t1!-60!*M%)
END SUB
```

Программа 9_09.c

```
#include <stdio.h>
#include <conio.h>
double pack_time(int H,int M,int S);
void unpack_time(double t,int *H,int *M,int *S);
  main()
{
  int H=18,M=0,S=0;
  double T;
  clrscr();
  T=pack_time(H,M,S);
  printf("\n%2d:%2d:%2d=%f",H,M,S,T);
  T=0.75;
  unpack_time(T,&H,&M,&S);
  printf("\n%2d:%2d:%2d=%f",H,M,S,T);
  getch();

}
/*-------------------------------------*/
double pack_time(int H,int M,int S)
/****************************************/
/* Упаковка времени - перевод часов (H) */
/* минут (M) и секунд (S) в часть суток */
/****************************************/
{
  return (H*3600.0+M*60+S)/86400.0;
}
/*---------------------------------------------*/
void unpack_time(double t,int *H,int *M,int *S)
```

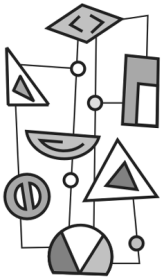
```
/*****************************************/
/* Распаковка времени - перевод части суток */
/* в часы (H), минуты (M) и секунды (S)     */
/*****************************************/
{
  double t1;
  t1=t*86400;
  *H=t1/3600.0;
  t1=t1-3600.0*(*H);
  *M=t1/60.0;
  *S=t1-60.0*(*M);
}
```

Программа 9_09.pas

```
program pack_unpack;
uses Crt;
var
  H,M,S:integer;
  T:double;
function pack_time(H,M,S:integer):double;
{ Упаковка времени - перевод часов (H)
  минут (M) и секунд (S) в часть суток }
begin
  pack_time:=(H*3600.0+M*60+S)/86400.0;
end;

procedure unpack_time(t:double;var H,M,S:integer);
{ Распаковка времени - перевод части суток
  в часы (H), минуты (M) и секунды (S)     }
var
  t1:double;
begin
  t1:=t*86400;
  H:=trunc(t1/3600.0);
  t1:=t1-3600.0*H;
  M:=trunc(t1/60.0);
  S:=trunc(t1-60.0*M);
end;
```

```
begin {main}
  clrscr;
  H:=18; M:=0; S:=0;
  T:=pack_time(H,M,S);
  writeln(H:2,': ',M:2, ': ',S:2, ' =',T:6:2);
  T:=0.5;
  unpack_time(T,H,M,S);
  writeln(H:2, ': ',M:2, ': ',S:2, ' =',T:6:2);
  readln;
end.
```

Глава 10



Использование системных функций

Системные функции MS-DOS и BIOS (Basic Input/Output System — базовая система ввода/вывода) предоставляют довольно широкие возможности по управлению внешними устройствами — дисками, видеосистемой, таймерами, манипулятором мыши и т. п. Интерфейс их вызова из программ, написанных на языке ассемблера, построен на использовании программного прерывания — машинной команды `INT` — по следующей схеме:

- запоминание в стеке содержимого регистров центрального процессора;
- формирование в регистрах исходной информации для работы системной функции `INT n`;
- извлечение из регистров результатов работы системной функции;
- восстановление регистров по информации из стека.

В принятой схеме аргумент `n` в команде `INT` задает номер группы системных функций, код регистра `AH` — номер подфункции, а содержимое других регистров, если это необходимо, рассматривается как исходные данные для работы выбранной подфункции. Результат работы системной функции также заносится в машинные регистры, содержимое которых должно быть сохранено для последующей обработки.

Перечень машинных регистров, задействованных в обработке программных прерываний, приведен в табл. 10.1. Вас не должно смущать, что некоторые из терминов в этой таблице использованы без детального объяснения.

Таблица 10.1. *Машинные регистры обработки программных прерываний*

Регистр	Назначение
`AX`	Основной сумматор. Используется и в качестве регистра данных при обмене с портами
`BX`	Дополнительный сумматор. Чаще используется как начальный адрес (база) в командах с индексной адресацией
`CX`	Счетчик циклов. Может использоваться в качестве операнда в арифметических операциях
`DX`	Основной регистр в операциях ввода/вывода. В сочетании с `AX` используется как часть 32-разрядного сумматора в "длинной арифметике"

Таблица 10.1 *(окончание)*

Регистр	Назначение
BP	Указатель базы. Используется для организации локальных стеков в процедурах и функциях
SI	В сочетании с базой сегмента данных (регистр DS) задает адрес источника информации (индекс источника)
DI	В сочетании с базой сегмента данных (регистр DS) задает адрес приемника информации (индекс приемника)
DS	Начальный адрес (база) основного сегмента данных
ES	Регистр базы дополнительного сегмента данных
Flags	Регистр с битами признаков состояния центрального процессора

Содержимое регистров AX, BX, CX и DX довольно часто рассматривается как пара 8-разрядных компонент, для обозначения которых используются сочетания AH (старшие 8 битов регистра AX), AL (младшие 8 битов регистра AX), BH, BL, CH, CL, DH, DL.

Если в программу на Си или Паскале не включены непосредственные команды ассемблера, то прямого доступа к машинным регистрам у пользователя нет. Для вызова системной функции по описанной выше схеме приходится обращаться к специальным процедурам-посредникам, которые пересылают значения из данных программы в машинные регистры, моделируют соответствующее программное прерывание и возвращают на поля программы содержимое машинных регистров.

Мы остановимся более детально на двух таких посредниках, считая, что остальными вы научитесь пользоваться самостоятельно.

```
TC:   int86(n,&in_regs,&out_regs);
      intdos(&in_regs,&out_regs);  //частный случай при n=0x21


TP:   Intr(n,regs);
      MSDOS(regs);                    {частный случай при n=$21}
```

Часть названия указанных процедур произошла от английского слова *interrupt* — прерывание. В функциях Си прослеживаются фрагменты обозначений старых процессоров фирмы Intel — 8086, 80186, 80286, 80386, 80486.

Основная часть системных функций поддерживается встроенным программным обеспечением материнской платы, которое раньше было жестко "зашито" в микросхемы BIOS, а теперь находится в более современной перепрограммируемой флэш-памяти. Значительная группа системных функций с общим номером 33 (0x21=$21) составляет часть операционной системы MS-DOS. В этой группе насчитывается 85 подфункций.

В заголовочном файле `dos.h` описано следующее объединение двух структур:

```
struct WORDREGS
  {unsigned int ax,bx,cx,dx,si,di,cflags,flags;};
struct BYTEREGS {unsigned char al,ah,bl,bh,cl,ch,dl,dh;};
union REGS {struct WORDREGS x; struct BYTEREGS h;};
```

К типу `REGS` относятся аргументы `in_regs` и `out_regs`, адреса которых задаются при вызове функций `int86` и `intdos`. Если мы включаем в свою программу, например, объединение с именем `reg` (`union REGS reg;`), то можем манипулировать с именами полей `reg.x.ax`, `reg.x.bx`, `reg.h.al`, `reg.h.ah`. При работе с программными прерываниями значения этих полей можно считать идентичными с содержимым соответствующих машинных регистров — `AX`, `BX`, `AL`, `AH`. На практике в программах на Си редко используют два разных объединения `in_regs` и `out_regs`, первое из которых выполняет роль полей с входной информацией, а второе — роль полей, на которые заносятся результаты работы системной функции. Обычно входные и выходные данные располагают на одном и том же поле.

В этом смысле авторы системы Turbo Pascal поступили более рационально, использовав в аналогичных процедурах на один аргумент меньше. Данные процедур `Intr` и `MSDOS` располагаются на полях записи с вариантами типа `Registers`, описанного в модуле `Dos` следующим образом:

```
type
  Registers=record
  case integer of
    0:(AX,BX,CX,DX,BP,SI,DI,DS,ES,flags:word);
    1:(AL,AH,BL,BH,CL,CH,DL,DH:byte);
  end;
```

Если в программе на Паскале объявлена запись `reg` (`reg:Registers;`), то мы можем использовать переменные с именами `reg.AX`, `reg.BX`, `reg.AL`, `reg.AH`, имея в виду, что их значения идентичны содержимому машинных регистров `AX`, `BX`, `AL`, `AH`.

Приводимые ниже фрагменты программ заменяют процедуру `gotoxy`, перемещающую курсор дисплея в позицию с заданными координатами на текстовой странице с указанным номером. Новая функция (процедура) `movetoxy` обходится только входными данными.

Программа 10_1.c

```
#include <dos.h>
void movetoxy(int x, int y, int page)
{
```

```
  union REGS r;
  r.h.ah=2;     //номер подфункции позиционирования курсора
  r.h.bh=page;  //номер текстовой страницы, от 0 до 7
  r.h.dl=x;     //номер столбца, от 1 до 80
  r.h.dh=y;     //номер строки, от 1 до 25
  int86(0x10,&r,&r); //обращение к группе функций BIOS с номером 16
}
```

Программа 10_1.pas

```
program cursor;
uses Dos;
.......
procedure movetoxy(x,y,page:byte);
var
  Registers r;
begin
  r.AH:=2;
  r.BH:=page;
  r.DL:=x;
  r.DH:=y;
  Intr($10,r);
end;
```

Примером системной функции, возвращающей несколько результатов, является процедура вывода текущей даты, использующая подфункцию с номером `0x2A` (`$2A`) в прерывании `0x33` (`$33`). После ее выполнения в участках памяти, соответствующих машинным регистрам, находится следующая информация:

`AL` — день недели (0 — воскресенье, 1 — понедельник, ..., 6 — суббота);

`DL` — день месяца;

`DH` — номер месяца;

`CX` — год.

Программа 10_2.c

```
/*Опрос текущей даты через прерывание, функция 33:42*/
#include <dos.h>
#include <stdio.h>
#include <conio.h>
```

```
void DatePrint(void);
  void main()
{
  DatePrint();
  getch();
}
/*-----------------------------------*/
void DatePrint(void)
{
  char *WeekDays[7]={"воскресенье","понедельник",
                     "вторник","среда","четверг","пятница","суббота"};
  union REGS  r;
  r.h.ah=0x2A;  /*Номер подфункции опроса текущей даты*/
  intdos(&r,&r); /*DL-день, DH-месяц, CX-год, AL-день недели*/
  printf("\nСегодня %d/%d/%d - %s",r.h.dl,r.h.dh,r.x.cx,
           WeekDays[r.h.al]);
}
```

Программа 10_2.pas

```
program int1;
uses Dos;
procedure DatePrint;
const
  WeekDays:array [0..6] of string=('воскресенье','понедельник',
                         'среда','четверг','пятница','суббота');
var
  r:Registers;
begin
  with r do begin
    AH:=$2A;  {Номер подфункции опроса текущей даты}
    MSDOS(r); {DL-день, DH-месяц, CX-год, AL-день недели}
    write('Сегодня ',DL,':',DH:2,':',CX,' - ');
    writeln(WeekDays[AL]);
  end;
end;
begin
  DatePrint;
  readln;
end.
```

К сожалению, система программирования QBasic не содержит в своем составе функции, аналогичной вышеописанным. Однако реализовать нечто подобное можно с помощью подпрограммы в машинных кодах, встраиваемой в текст на Бейсике операторами DATA, пересылаемой в оперативную память на место с известным адресом и вызываемой с помощью оператора CALL ABSOLUTE. Одна из таких подпрограмм описана в книге [7] и содержит всего 38 байт. Мы приводим полный текст аналогичного, но более экономного (32 байта), варианта на языке ассемблера для того, чтобы желающие могли расширить область сохраняемых данных и, при необходимости, подменить байт с номером прерывания.

Приведенный ниже текст представляет собой так называемый билистинг, получающийся в результате трансляции исходной программы в язык машинных команд. В левой колонке находятся адреса команд, справа от которых размещаются числовые коды операций и адреса операндов (некоторые машинные команды в явном виде не содержат адреса операндов). В средней колонке команды записаны на языке ассемблера. Каждая строка соответствует отдельной машинной команде или управляющему оператору языка ассемблера. Перед мнемоническим кодом команды может находиться метка, на которую могут ссылаться другие команды (в нашем примере такими метками являются символьные обозначения двухбайтовых слов RegAX, RegBx, RegCx, RegDx, предназначенных для хранения содержимого машинных регистров). Команды пересылки (код операции mov) перемещают значение второго операнда по первому адресу. Команды записи в стек (коды операции push, pusha) производят пересылку содержимого одного или нескольких машинных регистров в стек. Команды выборки из стека (коды операций pop, popa) осуществляют обратную пересылку из стека значения одного или нескольких регистров. Команда ret возвращает управление вызывающей программе.

Первая и последняя строки билистинга окаймляют подпрограмму. В них содержится название подпрограммы (IntDos) и указание о том, что она может находиться в памяти достаточно далеко от точки обращения (far — признак дальнего вызова).

В каждой строке может находиться комментарий, размещаемый справа после точки с запятой. В нашем примере прокомментированы все строки для тех, кто впервые сталкивается с текстом программы на языке ассемблера.

```
0000            IntDos PROC far         ;начало подпрограммы
0000 60                pusha            ;сохранение всех регистров
0001 1E                push ds          ;сохранение ds
0002 0E                push cs          ;сохранение cs
0003 1F                pop ds           ;пересылка cs в ds
0004 A1 0018           mov ax,RegAX     ;пересылка поля RegAX в ax
```

```
0007 CD 33              int 33h          ;прерывание с номером 33h
0009 89 1E 001A         mov RegBX,bx     ;запоминание регистра bx
000D 89 0E 001C         mov RegCX,cx     ;запоминание регистра cx
0011 89 16 001E         mov RegDX,dx     ;запоминание регистра dx
0015 1F                 pop ds           ;восстановление ds
0016 61                 popa             ;восстановление регистров
0017 CB                 ret              ;возврат из подпрограммы
0018 0000       RegAX   DW  0            ;поле для содержимого ax
001A 0000       RegBX   DW  0            ;поле для содержимого bx
001C 0000       RegCX   DW  0            ;поле для содержимого cx
001E 0000       RegDX   DW  0            ;поле для содержимого dx
                IntDos ENDP              ;конец подпрограммы
```

Текст приведенной выше подпрограммы в шестнадцатеричном формате запоминается в блоке данных:

```
DATA &H60,&H1E,&H0E,&H1F,&HA1,&H18,&H00,&HCD,&H33,&H89
DATA &H1E,&H1A,&H00,&H89,&H0E,&H1C,&H00,&H89,&H16,&H1E
DATA &H00,&H1F,&H61,&HCB
```

Для размещения этой подпрограммы и расположенных в ее хвосте четырех двухбайтовых полей в памяти резервируется массив длиной в 32 байта — `IntProg(1 TO 16) AS INTEGER`. Переписи в этот массив подлежат только первые 24 байта блока данных — собственно текст программы без полей `RegAX`, `RegBX`, `RegCX` и `RegDX`:

```
DIM IntProg(1 TO 16) AS INTEGER
' установка сегмента для работы подпрограммы
DEF SEG=VARSEG(IntProg(1))
' установка смещения для работы подпрограммы
Int33&=VARPTR(IntProg(1))
' Перепись подпрограммы из блока данных в массив
FOR J%=0 TO 23
  READ K% : POKE Int33&+J%,K%
NEXT J%
```

Обратите внимание на команды, изменяющие базовый (начальный) адрес сегмента. В качестве новой базы устанавливается адрес начала массива `IntProg`, что позволяет не производить перенастройку адресов машинной программы в зависимости от ее местоположения в оперативной памяти.

В тексте программы, как и в ее прототипе [7], имеется небольшой дефект — после обработки прерывания не запоминается содержимое регистра `AX`. А некоторые прерывания заносят в этот регистр результаты своей работы. Например, при опросе текущей даты в младшие разряды регистра `AX` зано-

сится день недели. Исправить эту оплошность довольно просто — необходимо после команды `int` добавить команду `mov RegAX,ax`, которая занимает три байта и в числовом представлении имеет вид `A3 xxyy` (здесь `xxyy` — относительный адрес поля `RegAX`). При этом адреса переменных `RegAX`, `RegBx`, `RegCX` и `RegDX` увеличатся на три байта. Но их желательно сохранить на границе полуслова, т. к. с ними оперируют как с элементами целочисленного массива. Поэтому в текст приводимой ниже модификации вставлен еще один байт — пустая команда `NOP` (числовой код — `90h`):

```
0000            IntDos PROC far          начало подпрограммы
0000 60                pusha             сохранение всех регистров
0001 1E                push ds           сохранение ds
0002 0E                push cs           сохранение cs
0003 1F                pop ds            пересылка cs в ds
0004 A1 001C           mov ax,RegAX      пересылка поля RegAX в ax
0007 90                nop               для выравнивания границы
0008 CD 33             int 33h           прерывание с номером 33h
000A A3 001C           mov RegAX,ax      запоминание регистра ax
000D 89 1E 001E        mov RegBX,bx      запоминание регистра bx
0011 89 0E 0020        mov RegCX,cx      запоминание регистра cx
0015 89 16 0022        mov RegDX,dx      запоминание регистра dx
0019 1F                pop ds            восстановление ds
001A 61                popa              восстановление регистров
001B CB                ret               возврат из подпрограммы
001C 0000       RegAX  DW  0             поле для содержимого ax
001E 0000       RegBX  DW  0             поле для содержимого bx
0020 0000       RegCX  DW  0             поле для содержимого cx
0022 0000       RegDX  DW  0             поле для содержимого dx
                IntDos ENDP              конец подпрограммы
```

Если вам понадобится вставить в текст подобной программы другие машинные команды, то для определения их цифровых кодов советуем зайти в программу td.exe (Turbo Debugger — Турбо Отладчик). Эта программа входит в комплект поставки любой Borland-системы. Набирая в окне ввода команды на языке ассемблера, вы сразу же увидите их аналог в шестнадцатеричной кодировке.

Замену команды `INT 33h` на команду `INT 21h` в новом варианте можно выполнить следующим образом:

```
IntProg(5)=&H21CD
```

Из-за того, что код операции команды `INT` занимает младший байт полуслова, т. е. байт с меньшим адресом, а номер прерывания — старший байт, в целочисленной константе `&H21CD` соответствующие значения переставлены

местами. Наверное, вы уже обратили внимание на аналогичные перестановки байтов в двухбайтовых адресах команд `mov`.

Задание номера подфункции, который должен попасть в регистр `AX`, следует выполнить путем присвоения нужного значения элементу массива `IntProg`, соответствующего полю `RegAX`. Так как нулевой адрес ссылается на начало этого массива, т. е. на элемент `IntProg(1)`, то адрес `&H1C=28` определяет местоположение элемента `IntProg(15)`. Поэтому адреса полей "регистров" `AX`, `BX`, `CX` и `DX` указывают на элементы `IntProg(15)`, `IntProg(16)`, `IntProg(17)` и `IntProg(18)`.

После выполнения подпрограммы обработки прерывания необходимо извлечь полученные данные из элементов массива `IntProg`, в которых были сохранены значения соответствующих регистров.

Ниже приведен текст программы опроса текущей даты на Бейсике, использующий в качестве подпрограммы обработки прерывания расширенную версию машинных кодов.

Программа 10_2.bas

```
DATA &H60,&H1E,&H0E,&H1F,&HA1,&H1C,&H00,&H90,&HCD,&H33
DATA &HA3,&H1C,&H00,&H89,&H1E,&H1E,&H00,&H89,&H0E,&H20
DATA &H00,&H89,&H16,&H22,&H00,&H1F,&H61,&HCB

DIM IntProg(1 TO 18) AS INTEGER
'установка сегмента для работы подпрограммы
DEF SEG=VARSEG(IntProg(1))
'установка смещения для работы подпрограммы
Int33& = VARPTR(IntProg(1))
'Перепись подпрограммы из блока данных в массив
FOR J%=0 TO 27
  READ K%: POKE Int33&+J%,K%
NEXT J%

IntProg(5)=&H21CD
IntProg(15)=&H2A00
CALL ABSOLUTE(Int33&)

FOR K=15 TO 18
  PRINT IntProg(K)
NEXT K
WeekDay=IntProg(15) MOD 256: ' День недели = AL
```

```
Day=IntProg(18) MOD 256:      ' День месяца = DL
Month=IntProg(18) \ 256:      ' Номер месяца = DH
Year=IntProg(17):             ' Год = CX
PRINT USING "Сегодня - ##/##/#### - #";Day;Month;Year;WeekDay
DEF SEG
END
```

Управление мышью

В составе штатных поставок систем TC, TP и QBasic отсутствуют средства управления мышью, а без этого манипулятора на порядок снижается ценность игровых и диалоговых программ. Непосредственную работу с мышью осуществляет системная программа — драйвер мыши, — загружаемая одновременно с загрузкой операционной системы MS-DOS или входящая в состав Windows. Драйверы мыши, ориентированные на работу с манипуляторами различных конструкций, могут отличаться друг от друга, например по количеству обслуживаемых кнопок. Но в большинстве своем их главные функции одинаковы и наша программа может ими воспользоваться через механизм прерываний.

В состав прерывания `0x33` (`$33`) входит более 30 подфункций, обеспечивающих связь с драйвером мыши. Наиболее полный их перечень приводится в [16]. Далеко не все эти функции являются предметом первой необходимости. Поэтому познакомимся только с некоторыми из них на примере программы, отслеживающей перемещения мыши по экрану в текстовом режиме. Достаточно полные модули управления мышью вы можете найти в книге В. В. Фаронова "Turbo Pascal 7.0. Практика программирования. Учебное пособие" и в [13].

Подфункция с номером 0 осуществляет "сброс" драйвера. В результате ее работы в регистр `AX` заносится состояние мыши и драйвера, а в `BX` — количество кнопок. Следует отметить, что обращение к этой подфункции в программе, работающей строго под управлением MS-DOS и запущенной из-под Windows 95/98, дает разные результаты. Можно, например, узнать, что на вашей трехкнопочной мыши присутствуют только две кнопки, что мышь или драйвер не установлены (`AX=0`). Дело в том, что обращаемся мы к разным драйверам и их ответы не всегда идентичны. Но основные действия по инициализации мыши все драйверы выполняют одинаково. К их числу относятся перевод курсора в центр экрана и его гашение, установка стандартной формы курсора, разрешение перемещать курсор по всему рабочему полю.

Подфункция с номером 1 включает режим отображения позиции курсора. Гашение образа курсора осуществляет подфункция 2, но драйвер продолжает отслеживать перемещения и погашенного курсора.

Наиболее важной является подфункция 3, которая сообщает текущие координаты курсора в пикселах (`x=CX`, `y=DX`) и состояние кнопок мыши в момент вызова подфункции (`BX=1` — нажата левая кнопка, `BX=2` — нажата правая кнопка, `BX=4` — нажата средняя кнопка). В принципе, значением `BX` может быть любая комбинация одновременно нажатых кнопок. В текстовом режиме значения координат однозначно определяются номерами текущей строки (`row`) и текущего столбца (`col`):

```
x=8*(col-1)        y=8*(row-1)
```

В этом вы можете убедиться на примерах следующих программ, построенных по единой схеме. После "сброса" драйвера дается 5-секундная задержка, чтобы рассмотреть текст сообщения о состоянии драйвера и мыши (курсор в это время не виден). Затем включается режим отображения курсора. Далее 60 раз с задержкой в 1 с выполняется цикл, в котором опрашивается и отображается на экране состояние мыши. Во время этого цикла вы можете перемещать курсор мыши, устанавливая его на пронумерованные позиции строк экрана и зажимая ту или иную кнопку. Координата `x` при этом меняется от 0 до 632, а координата `y` — от 0 до 192.

Программа 10_3.bas

```
' Модификация программы [7], использована первая версия подпрограммы в
' машинных кодах)
DATA &H60,&H1E,&H0E,&H1F,&HA1,&H18,&H00,&HCD,&H33,&H89
DATA &H1E,&H1A,&H00,&H89,&H0E,&H1C,&H00,&H89,&H16,&H1E
DATA &H00,&H1F,&H61,&HCB

DIM IntProg(1 TO 16) AS INTEGER
' установка сегмента для работы подпрограммы
DEF SEG=VARSEG(IntProg(1))
' установка смещения для работы подпрограммы
Int33&=VARPTR(IntProg(1))
' Перепись подпрограммы из блока данных в массив побайтно
FOR j%=0 TO 23
  READ K%: POKE IntDos&+j%,K%
NEXT j%
' Роспись экрана линейками через строку
CLS
COLOR 7,1
FOR I=1 TO 11
FOR j%=0 TO 79
  PRINT USING "#";j% MOD 10;
```

```
NEXT j%
  PRINT : PRINT
NEXT I
COLOR 7,0
IntProg(13)=0: ' Подфункция сброса драйвера мыши
CALL ABSOLUTE(Int33&)
LOCATE 2,1
IF IntProg(13)=-1 THEN PRINT "Работает драйвер MS-DOS  ";
PRINT "Число кнопок = ";IntProg(14);
SLEEP 5
IntProg(13)=1: ' Подфункция визуализации курсора на экране
CALL ABSOLUTE(Int33&)
IntProg(13)=3: ' Подфункция опроса состояния мыши
' 15:CX=x 16:DX=y 14:BX=состояние кнопок:
' 1 - нажата левая, 2 - правая, 4 - средняя
FOR j%=1 TO 60
  CALL ABSOLUTE(Int33&)
  LOCATE 4,1
  PRINT USING "x=### y=### кнопка #";IntProg(15);IntProg(16);
                                     IntProg(14)
  SLEEP 1
NEXT j%
LOCATE 6,1: PRINT "Цикл окончен. Нажмите любую клавишу"
SLEEP
DEF SEG
END
```

Программа 10_3.c

```
/* Работа с мышью в текстовом режиме */
#include <dos.h>
#include <stdio.h>
#include <conio.h>
  void main()
{
  int i,j;
  union REGS r;
  clrscr();
```

```
/*Роспись экрана линейками через строку*/
  textcolor(7);
  textbackground(1);
  for(i=1; i<12; i++)
     {
       for(j=0; j<80; j++)  cprintf("%d",j%10);
       printf("\n");
     }
  r.x.ax=0;
/* подфункция сброса драйвера */
  int86(0x33,&r,&r);
  textbackground(0);
  gotoxy(1,2);
  if(r.x.ax==0xFFFF) cprintf("Работает драйвер MS-DOS ");
  cprintf("Число кнопок = %d",r.x.bx);
  r.x.ax=1;
  sleep(5);
/* Подфункция визуализации курсора на экране */
  int86(0x33,&r,&r);
  r.x.ax=3;
/* Подфункция опроса состояния мыши*/
  for(j=0; j<60; j++)
  {
    int86(0x33,&r,&r);
    gotoxy(1,4);
    clreol();
    cprintf("x=%d y=%d нажата кнопка %d",r.x.cx,r.x.dx,r.x.bx);
    sleep(1);
  }
  gotoxy(1,6);
  cprintf("Цикл окончен. Нажмите любую клавишу");
  getch();
}
```

Программа 10_3.pas

```
program int1;
uses Crt,Dos;
```

```
var
  i,j:longint;
  r:Registers;
begin
  clrscr;
{Роспись экрана линейками через строку}
  textcolor(7);
  textbackground(1);
  for i:=0 to 10 do
    begin
      for j:=0 to 79 do  write(j mod 10);
      writeln;
    end;
  textbackground(0);
{подфункция сброса драйвера}
  r.AX:=0;
  Intr($33,r);
  gotoxy(1,2);
  if r.AX=$FFFF then write('Работает драйвер MS-DOS ');
  write('Число кнопок = ',r.BX);
  r.AX:=1;
{ Подфункция визуализации курсора на экране }
  Intr($33,r);
  r.AX:=3;
{Подфункция опроса состояния мыши}
  for j:=1 to 60 do
  begin
    Intr($33,r);
    gotoxy(1,4);
    clreol;
    write('x=',r.CX, ' y=',r.DX, ' кнопка =',r.BX);
    for i:=0 to 10000000 do i:=i;
  end;
  gotoxy(1,6);
  write('Цикл окончен. Нажмите Enter');
  readln;
end.
```

Красивые окна в текстовом режиме

В этом разделе мы познакомим вас с небольшим пакетом программ на Си, разработанным одним из авторов этой книги 1990 г., когда большинство программистов были вынуждены изобретать разные средства для управления выводом данных из-за их отсутствия в среде MS-DOS. Аналогичные пакеты с меньшими функциональными возможностями вы можете найти в книгах Р. Уинера "Язык Турбо Си" и В. В. Фаронова "Программирование на персональных ЭВМ в среде Турбо Паскаль".

Пакет с условным названием `TEXT_BOX` предназначен для оформления различных окон на экране дисплея и управления выводом текстовых данных в таких окнах. В его состав входит 21 функция для манипуляции со строками и текстовыми окнами, которые реализованы на базе подфункций прерывания BIOS с номером `0x10`. Их список приведен в табл. 10.2. Прерывание `0x10` обслуживает видеосистему не только в текстовых, но и в графических режимах, и представленные здесь возможности раскрывают примерно четверть этого арсенала.

Таблица 10.2. *Функции манипуляции*

Формат вызова функции	Назначение
`ask_attr(&cs,&cf,&in,&bl)`	Опрос цветовых атрибутов
`set_attr(cs,cf,in,bl)`	Установка цветовых атрибутов
`ask_cur(&x,&y)`	Опрос позиции курсора
`set_cur(x,y)`	Установка курсора в заданную позицию
`move_cur(n)`	Перемещение курсора на `n` позиций
`box_abs(row1,col1,row2,col2, bord,shade)`	Оформление окна
`box_rel(row1,col1,rows,cols, bord,shade)`	Оформление окна
`cl_rect(row,col,rows,cols, color)`	Очистка окна
`s_out(ch)`	Вывод символа в текущую позицию
`s_out_h(ch,n)`	Вывод `n` символов по горизонтали
`s_out_v(ch,n)`	Вывод `n` символов по вертикали
`s_box_abs(row1,col1,row2, col2,ch)`	Заполнение окна символом
`s_box_rel(row1,col1,rows, cols,ch)`	Заполнение окна символом
`xy_s_out(row,col,ch)`	Вывод символа в заданную позицию

Таблица 10.2 *(окончание)*

Формат вызова функции	Назначение
`s_out_c(row,col,nc,str)`	Вывод строки по центру
`s_out_l(row,col,nc,str)`	Вывод строки с прижимом влево
`s_out_r(row,col,nc,str)`	Вывод строки с прижимом вправо
`ask_page()`	Опрос активной страницы
`set_page(n)`	Установка активной страницы
`out_err(str)`	Вывод сообщения об ошибке
`init_txt()`	Инициализация текстового режима

Для удобства общения между функциями пакета определены следующие глобальные переменные:

- `_PAGE` — байт с номером активной страницы (начальное значение — 0);
- `_ATTR` — байт текущих цветовых атрибутов выводимого текста;
- `_COLR_S` — байт с номером цвета выводимых символов;
- `_COLOR_F` — байт с номером цвета фона;
- `_INTENS` — байт с признаком обычной (0) или повышенной (1) яркости;
- `_BLINK` — байт с признаком мерцания (0 — мерцания нет);
- `_ROW_CUR` — байт с номером текущей строки;
- `_COL_CUR` — байт с номером текущего столбца.

С целью сокращения обозначений и приближения их к идентификаторам регистров на Ассемблере введены следующие подстановки:

- `union REGS reg;`
- `#define AH reg.h.ah`
- `#define AL reg.h.al`
- `#define BH reg.h.bh`
- `#define BL reg.h.bl`
- `#define CH reg.h.ch`
- `#define CL reg.h.cl`
- `#define CX reg.x.cx`
- `#define DL reg.h.dl`
- `#define DH reg.h.dh`
- `#define INT10h int86(0x10,&reg,&reg)`

Программа `ask_attr` — опрос цветовых атрибутов

Если вы забыли, как выглядит байт цветовых атрибутов, то загляните в раздел 3.5.5.

```
int ask_attr(int *cs,int *cf,int *in,int *bl)
// cs - цвет символов, от 0 до 7
// cf - цвет фона, от 0 до 7
// in - яркость, 0 или 1
// bl - признак мерцания, 0 или 1
{
  AH=8;         //Подфункция опроса цветовых атрибутов
  BH=_PAGE;     //Номер текущей страницы
  INT10h ;            //Чтение текущего символа и атрибутов
 _ATTR=AH;      //Байт цветовых атрибутов
 _COLOR_S=_ATTR & 0x07;        //Цвет символа
 _COLOR_F=(_ATTR & 0x70) >> 4; //Цвет фона
 _INTENS=(_ATTR & 0x08) >> 3;  //Бит интенсивности
 _BLINK=(_ATTR & 0x80) >> 7;   //Бит мерцания
  *cs=_COLOR_S;    //возврат цвета символов
  *cf=_COLOR_F;    //возврат цвета фона
  *in=_INTENS;     //возврат признака яркости
  *bl=_BLINK;      //возврат признака мерцания
  return (_ATTR);  //возврат всех атрибутов цвета
}
```

Программа `set_attr` — установка цветовых атрибутов

```
void set_attr(int cs,int cf,int in,int bl)
{
//анализ атрибутов цвета на допустимость
   if(cs>=0 && cs<8 && cf>=0 && cf<8 &&
      in>=0 && in<2 && bl>=0 && bl<2)
     {
      _COLOR_S=cs;
      _COLOR_F=cf;
      _INTENS=in;
      _BLINK=bl;
//объединение атрибутов цвета в одном байте
      _ATTR=_BLINK << 7 | _COLOR_F << 4 | _INTENS << 3 | _COLOR_S;
```

```
    }
  else err_out("Ошибка при вызове set_attr");
}
```

Программа move_cur — перемещение курсора на n позиций вправо

Используя текущие координаты курсора (_ROW_CUR, _COL_CUR), функция вычисляет строку и столбец новой позиции и с помощью функции set_cur перемещает туда курсор. Если курсор выходит за пределы экрана, то его принудительно устанавливают в верхний левый угол.

```
void move_cur(int n)
{
  int pos;
  pos=_ROW_CUR*80+_COL_CUR+n;
  _ROW_CUR=pos/80;
  _COL_CUR=pos-_ROW_CUR*80;
  if(_ROW_CUR > 24)
    {
      _ROW_CUR=0;
      _COL_CUR=0;
    }
  set_cur(_ROW_CUR+1,_COL_CUR+1);
}
```

Программа box_abs — построение прямоугольника с рамкой и тенью

Контуры рамки образуются пробелами, одинарными и/или двойными "линиями" с помощью символов псевдографики. Массивы lu, ld, ru и rd заполнены кодами символов, используемыми для отображения левого верхнего (lu), левого нижнего (ld), правого верхнего (ru) и правого нижнего (rd) углов рамки. В массивах horiz и vert находятся коды символов, формирующие горизонтальные и вертикальные линии рамки. По индексу bord из них извлекаются знаки соответствующей окантовки и некоторые из них повторяются rpth раз по горизонтали и rptv раз по вертикали.

Тень создается с помощью строки и столбца пробелов, окрашенных в серый цвет и расположенных со сдвигом на одну позицию относительно нижней и левой (shade=-1) или нижней и правой (shade=1) границ рамки. Внутренность окна заполняется пробелами цветом фона, ранее установленного с помощью функции set_attr.

```
void box_abs(int row1,int col1,int row2,int col2,
             int bord,int shade)
```

```
// row1,col1 - левый верхний угол,
// row2,col2 - правый нижний угол
// bord  - номер типа рамки, от 0 до 4
// shade = -1(тень слева), 0(без), 1(тень справа)
{
  char lu[5]={ 0x20,0xDA,0xC9,0xD6,0xD5 };
  char ld[5]={ 0x20,0xC0,0xC8,0xD3,0xD4 };
  char ru[5]={ 0x20,0xBF,0xBB,0xB7,0xB8 };
  char rd[5]={ 0x20,0xD9,0xBC,0xBD,0xBE };
  char horiz[5]={ 0x20,0xC4,0xCD,0xC4,0xCD };
  char vert[5] ={ 0x20,0xB3,0xBA,0xBA,0xB3 };
  int rpth,rptv,attr;
  rptv=col2-col1-1;      //длина вертикали
  if(rptv <= 0)       rptv=1;
  rpth=row2-row1;        //длина горизонтали
  if(rpth <= 0)       rpth=1;
//анализ на допустимость параметров окна
  if(shade == 1 && col1+rptv >= 79) goto m1;
  if(shade ==-1 && col1==1) goto m1;
  if(shade != 0 && row1+rpth >= 25) goto m1;
  if(row1+rpth > 25 || col1+rptv+1 > 80) goto m1;
  xy_s_out(row1,col1,lu[bord]);  //верхний левый угол
  s_out_h(horiz[bord],rptv);     //верхняя горизонталь
  s_out_h(ru[bord],1);           //верхний правый угол
  set_cur(row1+1,col1);          //курсор в начало левой вертикали
  s_out_v(vert[bord],rpth);      //левая вертикаль
  set_cur(row1+1,col2);          //курсор в начало правой вертикали
  s_out_v(vert[bord],rpth);      //правая вертикаль
//роспись внутренности пробелами
  sbox_rel(row1+1,col1+1,rpth,rptv,32);
  xy_s_out(row2,col1,ld[bord]);  //левый нижний угол
  s_out_h(horiz[bord],rptv);     //нижняя горизонталь
  s_out_h(rd[bord],1);           //правый нижний угол
  if(shade == 0) goto m;         //обход, если нет тени
  attr=_ATTR;                    //запоминание атрибутов цвета
  set_attr(7,0,0,0);             //серый цвет для тени,
  if(shade == -1)                //если тень слева
    {
      set_cur(row1+1,col1-1);    //установка курсора левее и ниже
```

```
      s_out_v(219,rpth+1);        //вертикаль тени
      s_out_h(219,rptv+1);        //горизонталь тени,
    }
  else                            //если тень справа
    {
      set_cur(row1+1,col2+1);     //курсор правее и ниже
      s_out_v(219,rpth+1);        //вертикаль тени
      set_cur(row2+1,col1+1);     //курсор в начало горизонтали
      s_out_h(219,rptv+1);        //горизонталь тени
    }
  _ATTR=attr;                     //восстановление атрибутов цвета
m: set_cur (row1+1, col1+1) ;     //курсор в начало окна
  return;
m1:err_out("Ошибка при вызове box... ");
}
```

Программа box_rel — построение прямоугольника с рамкой и тенью

Эта программа отличается от предыдущей только способом задания габаритов рамки — вместо второго противоположного угла здесь задается количество строк (rows) и столбцов (cols). Программа определяет координаты противоположного угла и обращается к предыдущей функции.

```
void box_rel(int row1,int col1,int rows,int cols,int bord,int shade)
// rows,cols - число строк и столбцов
{
  box_abs(row1,col1,row1+rows-1,col1+cols-1,bord,shade);
}
```

Программа cl_rect — очистка прямоугольной области экрана

```
void cl_rect(int row,int col,int rows,int cols,int color)
// row, col - левый верхний угол,
// rows, cols - число строк и столбцов,
// color - цвет заливки, от 0 до 7
//( RED=4 GREEN=2 BLUE=1 )
{
  AL=rows+1;          //количество строк
  AH=0x06;            //номер подфункции
  CH=row-1;           //номер начальной строки
```

```
  CL=col-1;           //номер начального столбца
  DH=row+rows-2;      //номер конечной строки
  DL=col+cols-2;      //номер конечного столбца
  BH=color*16;        //цвет фона
  INT10h;
  set_cur(row,col);  //перевод курсора в начало окна
}
```

Программа s_out — вывод символа в текущую позицию

```
void s_out(char s)
{
  AH=9;         //подфункция вывода символа
  AL=s;         //код выводимого символа
  BH=_PAGE;     //номер активной страницы
  BL=_ATTR;     //текущие цветовые атрибуты
  CX=1;         //количество повторяемых символов
  INT10h;
  move_cur(1); //сдвиг курсора на 1 позицию вправо
}
```

Программа s_out_h — размножение символа с текущей позиции по строке

Функция отличается от предыдущей только установкой счетчика повторений символа. После размножения символа курсор переводится в ближайшую свободную позицию справа.

```
void s_out_h(char s,int rpt)
// s - размножаемый символ
// rpt - количество повторений
{
  AH=9;          //номер подфункции
  AL=s;
  BH=_PAGE;
  BL=_ATTR;
  CX=rpt;
  INT10h;
  move_cur(rpt);
}
```

Программа s_out_v — размножение символа с текущей позиции по столбцу

В цикле организуется перемещение курсора по вертикали вниз и вывод размножаемого символа. По окончании цикла курсор переводится за последний символ.

```
void s_out_v(char s,int rpt)
// s - размножаемый символ раз
// rpt - количество повторений
{
  int row,col,i;
  row=_ROW_CUR;
  col=_COL_CUR;
  BH=_PAGE;
  DL=col;
  BL=_ATTR;
  CX=1;
  for(i=row; i<row+rpt; i++)
    {
      AH=2;     //номер подфункции установки курсора
      DH=i;     //номер строки
      INT10h;
      AH=9;     //номер подфункции вывода символа
      AL=s;     //код выводимого символа
      INT10h;
      if(i==25) break;
    }
  set_cur(i,col+1);  //перевод курсора правее колонки
}
```

Программа sbox_abs — заполнение прямоугольной области заданным символом

Организуется цикл по количеству строк, в каждой из которых курсор устанавливается в начальную колонку строки и с помощью функции s_out_h выводится нужное количество символов по горизонтали. После заполнения области курсор переводится в левый верхний угол прямоугольника.

```
void sbox_abs(int row1,int col1,int row2,int col2,char s)
// s - символ-заполнитель
// row1,col1  - левый верхний угол
// row2,col2  - правый нижний угол
```

```
{
  int i;
  if(row1<=row2 && col1<=col2)
    {
      if(row2>25)  row2=25;
      if(col2>80)  col2=80;
      for(i=row1; i<row2+1; i++)
       {
         set_cur(i,col1);
         s_out_h(s,col2-col1+1);
       }
      set_cur (row1, col1);
    }
  else err_out("Ошибка при вызове sbox...");
}
```

Программа sbox_rel — заполнение прямоугольной области заданным символом

Функция отличается от предыдущей только тем, что вместо координат противоположного угла заданы ширина и высота области.

```
void sbox_rel(int row1,int col1,int rows,int cols,char s)
// s - символ-заполнитель
// row1, col1 - левый верхний угол
// rows, cols - число строк и столбцов
{
  sbox_abs(row1,col1,row1+rows-1,col1+cols-1,s);
}
```

Программа xy_s_out — вывод символа в заданную позицию

Функция переводит курсор в указанную позицию и по функции s_out отображает символ, одновременно смещая курсор на одну позицию вправо.

```
void xy_s_out(int row,int col,char s)
// row,col - координаты заданной позиции
// s - выводимый символ
{
  set_cur(row,col);
  s_out(s);
}
```

Программа s_out_c — вывод строки с центровкой в заданной полосе

Функция определяет длину выводимой строки и сравнивает ее с длиной предоставляемой полосы. Если полоса задана с запасом, то вывод текста производится с позиции, отстоящей от начала полосы на половину разницы длин. В противном случае строка размещается с начала полосы.

```
void st_out_c(int row,int col,int nc,char *string)
// row,col - начало полосы,
// nc  - длина полосы,
// string - выводимая строка
{
  int ls,i;
  ls=strlen(string);
  i=(nc-ls)/2;
  if(ls <= nc)
     st_out_l(row,col+i,nc,string);
  else
     st_out_l(row,col,nc,&string[i]);
}
```

Программа s_out_l — вывод строки в полосу с прижимом влево

Строка выводится посимвольно с начала полосы до тех пор, пока либо не будут исчерпаны все символы строки, либо не будет заполнена последняя позиция полосы. За пределами правой границы полосы вывод не производится.

```
void st_out_l(int row,int col,int nc,char *string)
// row,col - начало полосы,
// nc  - длина полосы,
// string - выводимая строка
{
  char s;
  int i;
  for(i=0; i<nc && string[i] != 0x00; i++)
    {
      s=string[i];
      xy_s_out(row,col+i,s);
    }
}
```

Программа `s_out_r` — вывод строки в полосу с прижимом вправо

Если ширина полосы превышает длину строки, то вычисляется начальная позиция строки в полосе, при которой последний символ строки попадает в последнюю позицию полосы, и вывод осуществляется по функции `st_out_l`. Если длина выводимой строки превышает ширину отведенной полосы, то у строки отсекаются лишние символы с начала и остаток строки заполняет полосу целиком.

```
void st_out_r(int row,int col,int nc,char *string)
// row,col - начало полосы,
// nc  - длина полосы,
// string - выводимая строка
{
  int ls,i;
  ls=strlen(string);
  i=nc-ls;
  if(ls <= nc)
     st_out_l(row,col+i,ls,string);
  else
     st_out_l(row,col,nc,&string[i]);
}
```

Программа `ask_page` — опрос активной страницы

Вместо этой функции вызывающая программа может проанализировать значение глобальной переменной `_PAGE`.

```
int ask_page(void)
{
  return (_PAGE);
}
```

Программа `set_page` — установка текущей страницы

Помимо записи номера активной страницы в глобальную переменную `_PAGE`, необходимо довести эту информацию и до BIOS с помощью подфункции номер 5.

```
void set_page(int p)
//p - номер, от 0 до 7
{
  if(p>=0 && p<8)
```

```
    {
      _PAGE=p;
      AH=5;
      AL=p;
      INT10h;
    }
  else err_out("Ошибка при вызове set_page");
}
```

Программа err_out — вывод сообщения об ошибке

Сообщение об ошибке выдается красными мигающими буквами в самой нижней строке экрана. После вывода сообщения делается выдержка до нажатия любой клавиши.

```
void err_out (char *string)
{
  set_attr(7,4,1,1);
  st_out_l(25,1,80,string);
  getch();
}
```

Программа init_txt — инициализация пакета text_box

```
void init_txt (void)
/*******************************************/
/* Инициализация экрана в текстовом режиме : */
/*                 размер экрана - 25 x 80   */
/*                маска символов -  8 x 14   */
/*******************************************/
{
  extern union REGS reg;
  AH=0;
  AL=2;
  INT10h;              /* установка режима */
  set_page(0); /* 0-я страница */
  set_cur(1,1);  /* курсор - в начало */
  set_attr(7,0,0,0);  /*  цветовые атрибуты */
  AH=9;
  AL=32;               /* код пробела */
```

```
  BL=7;         /* белым по черному */
  BH=0;         /* 0-я страница */
  CX=2000;
  INT10h ;             /* очистка экрана */
}
```

Программа `tst_text` — проверки пакета `text_box`

Для проверки работоспособности описанного выше пакета предлагается следующий тест, охватывающий почти все функции пакета:

```
#include "text.h"
  void main()
{
  init_txt();
  set_attr(4,2,1,0);      //красный цвет, зеленый фон
  box_abs(2,2,10,30,4,1); //прямоугольник с тенью справа
  getch();
  move_cur(2);            //сдвиг курсора вправо
  getch();
  s_out('A');             //вывод одной буквы
  getch();
  s_out_h('B',3);         //вывод трех букв в строке
  getch();
  s_out_v('C',4);         //вывод четырех букв по вертикали
  getch();
  cl_rect(3,3,7,27,4);    //заливка внутренности окна красным
  getch();
  set_attr(7,1,0,1);
  box_rel(13,40,8,28,4,-1);//прямоугольник с тенью слева
  getch();
  set_attr(7,1,0,0);
  sbox_rel(14,41,6,26,'7');
  getch();
  st_out_l(13,5,16,"1234567890123456"); //линейка в полосе
  st_out_l(14,5,16,"                "); //очистка полосы
  getch();
  st_out_l(14,5,16,"Привет"); //вывод в полосу с прижимом влево
  getch();
  st_out_l(15,5,16,"                "); //очистка полосы
```

```
  getch();
  st_out_c(15,5,16,"Привет"); //вывод в полосу по центру
  getch();
  st_out_l(16,5,16,"                "); //очистка полосы
  getch();
  st_out_r(16,5,16,"Привет"); //вывод в полосу с прижимом вправо
  getch();
}
```

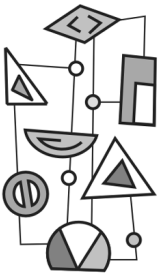
В состав файла с текстом головной программы ***можно*** включить все функции пакета и набрать заголовочный файл `text.h`:

```
#include <conio.h>
#include <dos.h>
#include <string.h>
union REGS reg;

#define AH reg.h.ah
#define AL reg.h.al
#define BH reg.h.bh
#define BL reg.h.bl
#define CH reg.h.ch
#define CL reg.h.cl
#define CX reg.x.cx
#define DL reg.h.dl
#define DH reg.h.dh
#define INT10h int86(0x10,&reg,&reg)

unsigned char _PAGE,_ATTR,_COLOR_S,_COLOR_F,_INTENS,_BLINK;
unsigned char _ROW_CUR,_COL_CUR;

int  ask_attr (int *cs,int *cf,int *in,int *bl);
void ask_cur (int *r, int *c);
int  ask_page (void);
void box_abs(int row1,int col1,int row2,int col2, int bord,int shade);
void box_rel(int row1,int col1,int rows,int cols, int bord,int shade);
void cl_rect(int row1,int col1,int rows,int cols,int color);
void err_out(char *string);
void init_txt (void);
void move_cur (int n);
void s_out (char s);
```

```
void s_out_h (char s,int rpt);
void s_out_v (char s,int rpt);
void sbox_abs (int row1,int col1,int row2,int col2,char s);
void sbox_rel (int row1,int col1,int rows,int cols,char s);
void set_attr (int cs,int cf,int in,int bl);
void set_cur (int r,int c);
void set_page (int p);
void st_out_c (int row,int col,int nc, char *string);
void st_out_l (int row,int col,int nc, char *string);
void st_out_r (int row,int col,int nc, char *string);
void xy_s_out (int row,int col,char s);
```

Если этот пакет вам понадобится в будущем, то функции пакета следует протранслировать и с помощью сервисной программы `tlib.exe` поместить полученные объектные модули в библиотеку. Эта библиотека может быть подключена к проекту вашего приложения.

Приложение 1



Указатель программ

№ п/п	Обозначение программы	Назначение программы	QB, TC, TP стр.
		Обработка числовой информации	
1	2_01	Ввод и вывод целочисленных данных	19, 19, 20
2	2_03	Преобразование десятичного числа в системы с основанием 2, 8 и 16	21, 23, 23
3	2_04	Преобразование десятичного числа в систему с основанием r	26, 27, 28
4	2_05	Симметричное разложение с наименьшим основанием	29, 30, 31
5	2_06	Суммирование цифр десятичного числа	33, 33, 34
6	2_07	"Счастливый" билет	35, 36, 36
7	2_08	Количество различных цифр в числе	38, 38, 39
8	2_09	Определение цифры в заданной позиции	40, 41, 42
9	2_10	Генерация чисел с заданной суммой цифр	42, 43, 43
10	2_11	Вывод числа словами	45, 47, 48
11	2_12	Суммирование двоичных цифр	50, 51, 51
12	2_13	Позиция старшей единицы в двоичном числе	52, 53, 54
13	2_14	Максимальное количество подряд идущих единиц в двоичном числе	55, 56, 57
14	2_15	Расстояние между двоичными кодами	59, 60, 61
15	2_16	Переворот цифр десятичного числа	62, 62, 63
16	2_17	Определение числового палиндрома	65, 65, 66
17	2_18	Генерация палиндромов с заданным свойством	68, 68, 69
18	2_19	Числовые преобразования до зацикливания	71, 72, 73
19	2_20	Разложение числа на простые сомножители	74, 75, 75
20	2_21	Анализ простого числа	76, 77, 78
21	2_22	Решето Эратосфена	79, 80, 81

(продолжение)

№ п/п	Обозначение программы	Назначение программы	QB, TC, TP стр.
		Обработка числовой информации	
22	2_23	Разложение четного числа на сумму простых чисел	83, 83, 84
23	2_24	Генерация чисел Хэмминга	86, 86, 87
24	2_25	Генерация неправильно сокращаемых дробей	89, 90, 91
25	2_26	Разложение натурального числа на сумму квадратов	92, 93, 94
26	2_27	Анализ взаимного расположения точки и треугольника	96, 96, 98
27	2_28	Игра на вычитание	99, 100, 101
28	2_29	Определение номера узла по его координатам	104, 105, 106
29	2_30	Определение расстояния между узлами	107, 107, 108
30	2_31	Определение соседних узлов	111, 112, 114
		Обработка символьной информации	
31	3_01	Формирование таблицы ASCII	128, 128, 129
32	3_02	Преобразование строк к верхнему регистру	130, 131, 132
33	3_03	Сортировка фамилий	134, 135, 137
34	3_04	Подсчет числа слов в строке	139, 139, 140
35	3_05	Анализ нажатой клавиши	141, 142, 142
36	3_06	Упорядочение трех цветов радуги	144, 145, 145
37	3_07	Вывод текста с различным прижимом	146, 146, 147
38	3_08	Сравнение строк с игнорированием пробелов	148, 149, 151
39	3_09	Вывод текста с разноцветными буквами	152, 153, 154
40	3_10	Преобразование обычной дроби в десятичную	155, 155, 157
41	3_11	Перевод числа в римскую систему	159, 159, 160
42	3_12	Перевод числа из римской системы	161, 162, 163
43	3_13	Вхождение строк с разрядкой	165, 165, 166
		Работа с массивами	
44	4_01	Сложение целочисленных квадратных матриц	175
45	bubble	Пузырьковая сортировка	182, 182, 183
46	select	Сортировка методом отбора	184, 184, 185
47	insert	Сортировка методом вставки	186, 186, 187
48	shell	Сортировка методом Шелла	187, 188, 188
49	hoare	Сортировка методом Хоара	190, 190, 191
50	ssearch	Последовательный поиск	193, 193, 194

(продолжение)

№ п/п	Обозначение программы	Назначение программы	QB, TC, TP стр.
		Работа с массивами	
51	bsearch	Бинарный поиск	194, 195, 195
52	4_02	Угадывание задуманного числа	196, 197, 198
53	4_03	Перестановка компонент одномерного массива	198, 199, 200
54	4_04	Перестановка головы и хвоста массива	202, 203, 204
55	4_05	Форматированный вывод целочисленного массива	205, 206, 207
56	4_06	Ход конем	209, 210, 211
57	4_07	Хронометраж методов сортировки	214, 214, 216
58	4_08	Количество счастливых билетов	217, 217, 218
59	4_09	Количество разных элементов в целочисленном массиве	219, 221, 224
60	4_10	Перемешивание колоды карт	226, 227, 228
61	4_11	Игра в НИМ	230, 232, 234
62	4_12	Игра "крестики-нолики"	239, 242, 245
63	4_13	Слияние массивов	249, 249, 250
		Рекурсивные программы	
64	5_01	Числа Фибоначчи	255, 255, 256
65	5_02	Наибольший общий делитель	257, 258, 258
66	5_03	Ханойские пирамиды	260, 260, 261
		Подпрограммы (процедуры) и функции	
67	6_01	Построение дерева решений	277, 279, 281
		Работа с дисковыми файлами	
68	7_01	Обмен со строковым файлом (с ошибкой) (Qbasic, Паскаль)	287 293
69	7_02	Обмен со строковым файлом	289, 299, 294
70	7_03	Обмен с записеориентированным файлом	291, 301, 296
71	7_04	Обмен с двоичным файлом	292, 302, 297
72	7_05	Перекодировка текстов из MS-DOS в Windows	304, 305, 306
73	7_06	Телефонный справочник	308, 308, 309
74	7_07	Создание резервной копии файла	311, 311, 312
		Работа с массивами	
75	7_08	Выдача каталога на экран	316, 316, 317
76	7_09	Сдвиг содержимого текстового файла	318, 319, 320

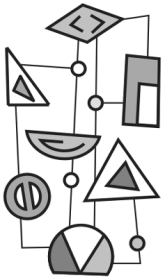
(продолжение)

№ п/п	Обозначение программы	Назначение программы	QB, TC, TP стр.		
		Машинная графика			
77	8_01	Демонстрация графических окон (QBasic)	330		
78	8_02	Построения в относительных координатах (QBasic)	330		
79	8_03	Построение синусоиды (QBasic, Паскаль)	331		332
80	8_04	Демонстрация цветовой палитры (QBasic, Си)	334	335	
81	8_05	Рисование белым по черному (QBasic, Паскаль)	341		341
82	8_06	Построение эллипса (QBasic)	342		
83	8_07	Построение дуги и сектора эллипса (QBasic)	343		
84	8_08	Демонстрация системных шаблонов заливки (Си)		345	
85	8_09	Демонстрация пользовательских шаблонов заливки (Си)		347	
86	8_10	Штриховка в шахматном порядке (QBasic)	356		
87	8_11	Построение кирпичной стены (QBasic)	356		
88	8_12	Заполнение прозрачным шаблоном (Си)		358	
89	8_13	Буква *А* в разных шрифтах (Си)		363	
90	8_14	Построение шахматной доски	365,	365,	366
91	8_15	Отображение семисегментных цифр (Си)		368	
92	8_16	Цифровые часы (Паскаль)			371
93	8_17	Летающая тарелка	374,	375,	376
94	8_18	Биоритмы	379,	381,	384
		Календарные даты и временные интервалы			
95	JD1	Вычисление юлианской даты (Си)		390	
96	JD2	Вычисление юлианской даты (Си)		390	
97	JD3	Вычисление юлианской даты (Си)		391	
98	9_01	Вычисление юлианских дат	392,	393,	394
99	9_02	Преобразование юлианских дат в григорианские	396,	397,	398
	9_03	Определение интервала времени между двумя датами	400,	401,	402
100	9_04	Определение дня недели	404,	405,	407
101	9_05	Определение порядкового дня в году	408,	409,	410
102	9_06	Восстановление даты по порядковому дню года	411,	411,	412
103	9_07	Количество дней в месяце	414,	414,	415

(окончание)

№ п/п	Обозначение программы	Назначение программы	QB, TC, TP стр.
		Календарные даты и временные интервалы	
104	9_08	Календарь на заданный месяц любого года	416, 418, 420
105	9_09	Упаковка и распаковка времени	422, 423, 424
		Прерывания и системные функции	
106	10_1	Перемещение курсора (Си, Паскаль)	429, 430
107	10_2	Опрос текущей даты	435, 430, 431
108	10_3	Управление мышью	437, 438, 439
109	`ask_attr`	Опрос цветовых атрибутов (Си)	443
110	`set_attr`	Установка цветовых атрибутов (Си)	443
111	`move_cur`	Перемещение курсора (Си)	444
112	`box_abs`	Построение окна с тенью (Си)	444
113	`box_rel`	Построение окна с тенью (Си)	446
114	`cl_rect`	Очистка прямоугольной области (Си)	446
115	`s_out`	Вывод символа в текущую позицию (Си)	447
116	`s_out_h`	Размножение символа по горизонтали (Си)	447
117	`s_out_v`	Размножение символа по вертикали (Си)	448
118	`sbox_abs`	Заполнение символом прямоугольной области (Си)	448
119	`sbox_rel`	Заполнение символом прямоугольной области (Си)	449
120	`xy_s_out`	Вывод символа в указанную позицию (Си)	449
121	`st_out_c`	Вывод строки в центре полосы (Си)	450
122	`st_out_l`	Вывод строки, прижатой к левой границе полосы (Си)	450
123	`st_out_r`	Вывод строки, прижатой к правой границе полосы (Си)	451
124	`ask_page`	Опрос активной страницы (Си)	451
125	`set_page`	Установка активной страницы (Си)	451
126	`err_out`	Вывод сообщения об ошибке (Си)	452
127	`init_txt`	Инициализация пакета `text_box` (Си)	452
128	`tst_text`	Проверка пакета `text_box` (Си)	453

Приложение 2



Список литературы

1. Абрамов С. А., Гнездилова Г. Г., Капустина Е. Н., Селюн М. И. Задачи по программированию. — М.: "Наука", 1988. — 224 с.
2. Березин Б. И., Березин С. Б. Начальный курс С и С++. — М.: Диалог-МИФИ, 1999. — 288 с.
3. Бобровский С. Программирование на языке QBasic для школьников и студентов. — М.: ДЕСС КОМ, 2000. — 207 с.
4. Брудно А. Л., Каплан Л. И. Олимпиады по программированию для школьников. — М.: Наука, 1985. — 96 с.
5. Грызлов В. И., Грызлова Т. П. Турбо Паскаль 7.0. — М.: ДМК, 1998. — 400 с.
6. Дагене В. А., Григас Г. К., Аугутис К. Ф. 100 задач по программированию. — М.: Просвещение, 1993. — 255 с.
7. Каспер Э. Освоим QBasic играючи. — М.: Радио и связь, 1999. — 264 с.
8. Кетков Ю. Л. GW-, Turbo- и Quick Basic для IBM PC. — М.: Финансы и статистика, 1992. — 240 с.
9. Кетков Ю. Л., Кетков А. Ю., Шапошников Д. Е. Персональный компьютер: Школьная энциклопедия. — М.: Большая Российская Энциклопедия, 1998. — 440 с.
10. Кирюхин В. М., Лапунов А. В., Окулов С. М. Задачи по информатике. Международные олимпиады. — М.: ABF, 1996. — 272 с.
11. Культин Н. Б. Turbo Pascal в задачах и примерах. — СПб.: БХВ-Петербург, 2000. — 256 с.
12. Культин Н. Б. C/C++ в задачах и примерах. — СПб.: БХВ-Петербург, 2001. — 288 с.
13. Немнюгин С. А. Turbo Pascal. — СПб.: Питер, 2000. — 496 с.

14. Очков В. Ф., Рахаев М. А. Этюды на языках QBasic, Quick Basic, Basic Compiler. — М.: Финансы и статистика, 1995. — 368 с.

15. Подбельский В. В. Язык Си++. — М.: Финансы и статистика, 2000. — 560 с.

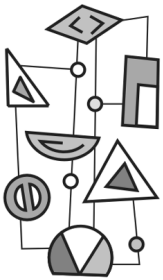
16. Прокофьев Б. П., Сухарев Н. Н., Храмов Ю. Е. Графические средства Turbo C и Turbo C++. — М.: Финансы и статистика, 1992. — 160 с.

17. Фаронов В. В. Turbo Pascal 7.0. Начальный курс: Учебное пособие. — М.: Нолидж, 1997. — 616 с.

18. Шилдт Г. Теория и практика C++. — СПб.: BHV—Санкт-Петербург, 1996. — 416 с.

19. Уэзерелл Ч. Этюды для программистов. — М.: Мир, 1982. — 288 с.

Приложение 3



Описание дискеты

К книге прилагается дискета, на которой собраны тексты всех исходных программ, приведенных в главах 2—10. Для экспериментов с программами на алгоритмических языках вам понадобится одна из (или все) систем программирования — QBasic, Borland C++ 3.1 (для большинства примеров подойдет и более ранняя версия Turbo C 2.0), Turbo Pascal (6.0 или 7.0).

Кроме текстов исходных программ, на дискете находятся exe-файлы, полученные, главным образом, в процессе компиляции программ с расширением pas (их объем существенно меньше, чем результат трансляции аналогичных программ на Си). Для демонстрации их работы вам не потребуется никаких систем программирования.

В корневом каталоге дискеты находятся два файла — readme.txt и disketa.exe. В первом из них содержится краткая информация о дискете, аналогичная тексту данного приложения. Второй файл представляет собой самораспаковывающийся архив. Его нужно переписать на свой винчестер и запустить. После распаковки появится каталог DISKETA, внутри которого находятся подкаталоги "Глава_01", "Глава_02", ..., "Глава_10". В каждом из них, в свою очередь, расположено по четыре подкаталога с именами Basic, C, Pascal и Exe. В них собраны тексты исходных и готовых к исполнению программ, содержащиеся в соответствующих разделах книги. Имена файлов на дискете идентичны книжным обозначениям программ. В некоторых подкаталогах Exe, кроме исполняемых программ, находятся вспомогательные файлы, необходимые для работы программ. К их числу относятся драйвер монитора, обеспечивающий работу графических программ на языках Си и Паскаль, и файлы с расширениями chr, демонстрирующие начертания букв в разных шрифтах.