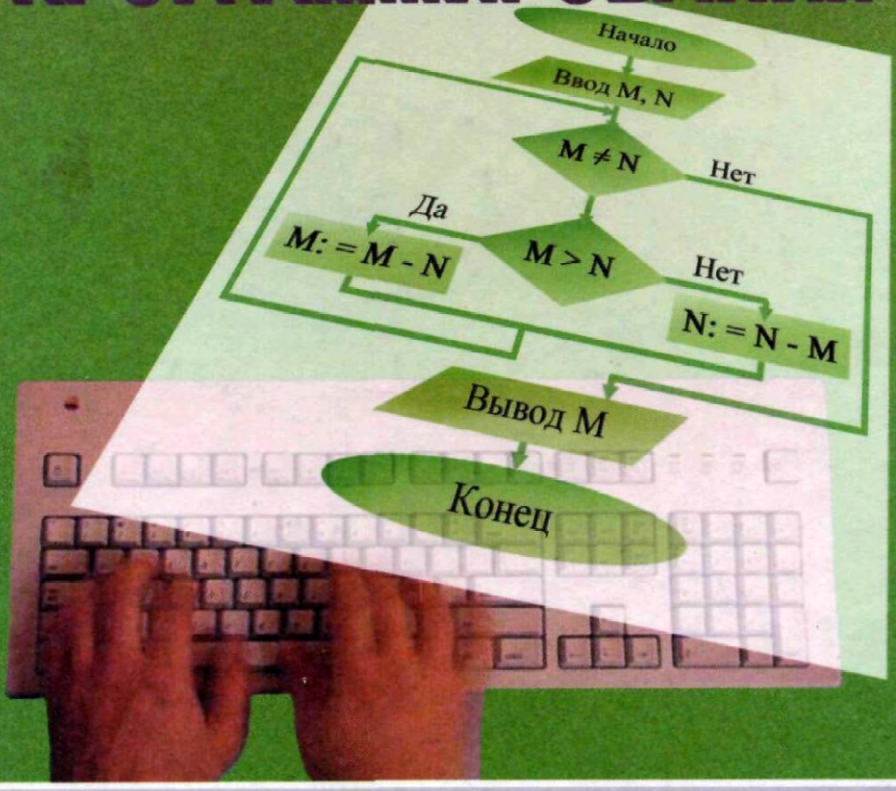
СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

И.Г.Семакин
А.П.Шестаков

ОСНОВЫ ПРОГРАММИРОВАНИЯ



издательство Мастерство

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

И. Г. СЕМАКИН, А. П. ШЕСТАКОВ

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Допущено
Министерством образования Российской Федерации
в качестве учебника для студентов образовательных учреждений
среднего профессионального образования, обучающихся
по специальностям 2202 «Автоматизированные системы
обработки информации и управления (по отраслям)»,
2203 «Программное обеспечение вычислительной техники
и автоматизированных систем»

Москва
ИЗДАТЕЛЬСТВО Мастерство
2002

УДК 681.3.06
ББК 22.18
С 12

Рецензент —
зав. кафедрой прикладной математики и информатики Пермского государственного университета, д-р физ.-мат. наук, проф. *С. В. Русаков*

Семакин И. Г., Шестаков А. П.

С 12 Основы программирования: Учебник. — М.: Мастерство, 2002. — 432 с.

ISBN 5-294-00054-7

Изложены основы структурной методики построения алгоритмов. Рассмотрены основы программирования на базе языка Паскаль (в версии Турбо Паскаль-7.0). Изложен стандартный язык Си с некоторыми элементами его расширения в версии Си++. Представлены задачи по программированию, предназначенные для организации практикума на ЭВМ (более 800 заданий).

Для студентов средних профессиональных учебных заведений. Может быть использован учениками старших классов средней школы и студентами начальных курсов высших учебных заведений.

УДК 681.3.06
ББК 22.18

ISBN 5-294-00054-7

ПРЕДИСЛОВИЕ

Программирование все в большей степени становится занятием лишь для профессионалов. Объявленный в середине 1980-х гг. лозунг «Программирование — вторая грамотность» остался в прошлом. В понятие «компьютерная грамотность» сегодня входит прежде всего навык использования многообразных средств информационных технологий. Решая ту или иную информационную задачу, необходимо выбрать адекватное программное средство. Это могут быть электронные таблицы, системы управления базами данных, математические пакеты и т.п. И только в том случае, когда подобные средства не дают возможности решить задачу, следует прибегать к универсальным языкам программирования.

Принято различать программистов двух категорий: прикладных и системных. Системные программисты — это разработчики базовых программных средств ЭВМ (операционных систем, трансляторов, сервисных средств и т.п.). Они являются профессионалами высочайшего уровня в программировании. Прикладные программисты разрабатывают средства прикладного программного обеспечения ЭВМ, предназначенные для решения задач из различных областей (наука, техника, производство, сфера обслуживания, обучение и т.п.). Требования к качеству как прикладных программ, так и системных сегодня очень высоки. Программа должна не только правильно решать задачу, но и иметь современный интерфейс, быть высоконадежной, дружественной по отношению к пользователю и т.д. Только такие программы могут выдерживать конкуренцию на мировом рынке программных продуктов. Программирование на любительском уровне сегодня никому не нужно.

По мере развития компьютерной техники развивались также и методика, и технология программирования. Сначала возникает командное и операторное программирование, в 1960-х гг. бурно развивается структурное программирование, появляются линии логического и функционального программирования, а в последнее время — объектно-ориентированное и визуальное программирование.

Задача, которую следует ставить при первоначальном изучении программирования, — освоение основ структурной методики программирования. Для указанной цели наиболее подходящим средством является язык программирования Паскаль. Автор языка Паскаль — швейцарский профессор Никлаус Вирт — создавал его имен-

но для этого. Структурная методика остается основой программистской культуры. Не освоив ее, человек, взявшийся изучать программирование, не имеет никаких шансов стать профессионалом.

Реализации Паскаля в версиях фирмы Borland для IBM, известных под названием Турбо Паскаль, значительно расширили язык по сравнению с вариантом Вирта. Начиная с версии 5.5 Турбо Паскаль становится также и языком объектного программирования.

Содержание третьей главы настоящего учебника ориентировано на глубокое освоение студентами базовых понятий языков программирования высокого уровня в их реализации на Паскале. Такая подготовка облегчает изучение других языков программирования. Изложение основ языков Си/Си++ в четвертой главе опирается на знания и навыки, полученные учащимися при изучении Паскаля.

При изучении данного курса студентам понадобятся знания основ алгоритмизации в рамках школьного базового курса информатики. Обычно в школе алгоритмизация изучается с использованием учебных исполнителей, с помощью которых можно успешно освоить основы структурной методики, а именно:

- построение алгоритмов из базовых структур;
- применение метода последовательной детализации.

Желательным является знакомство с архитектурой ЭВМ на уровне машинных команд (достаточно на модельных примерах учебных компьютеров, изучаемых в школьной информатике; совсем не обязательно изучение реальных языков команд или ассемблера). Эти знания позволяют освоить основные понятия программирования, такие как переменная, присваивание; «входить в положение транслятора» и благодаря этому не делать ошибок, даже не помня каких-то деталей синтаксиса языка; предвидеть те «подводные камни», на которые может «напороться» ваша программа в процессе выполнения. По существу, все эти качества и отличают профессионального программиста от дилетанта.

Еще одно качество профессионала — способность воспринимать красоту программы, получать эстетическое удовольствие оттого, что она хорошо написана. Нередко это чувство помогает интуитивно отличить неправильную программу от правильной. Однако основным критерием правильности является, безусловно, не интуиция, а грамотно организованное тестирование.

Процесс изучения и практического освоения программирования делится на три части:

- изучение методов построения алгоритмов;
- изучение языка программирования;
- изучение и практическое освоение определенной системы программирования.

Решению первой задачи посвящены вторая и пятая главы учебника. Во второй главе даются основные, базовые понятия и прин-

ципы построения алгоритмов работы с величинами. В пятой главе излагаются некоторые известные методики полного построения алгоритмов, обсуждаются проблемы тестирования программ, оценки сложности алгоритмов.

Языки программирования Турбо Паскаль и Си/Си++ излагаются в третьей и четвертой главах учебника соответственно. Подчеркнем, что данная книга — это прежде всего учебник по программированию, а не по языкам Паскаль и Си. Поэтому исчерпывающего описания данных языков вы здесь не найдете. Языки излагаются в том объеме, который необходим для начального курса программирования. Более подробное описание языков можно найти в книгах, приведенных в списке литературы.

В учебнике нет инструкций по работе с конкретными системами программирования для изучаемых языков. С ними студенты должны познакомиться в процессе практики на ЭВМ, используя другие источники.

Шестая глава представляет собой достаточно большой задачник по программированию. Этот задачник можно использовать для организации практических и лабораторных занятий по любому из изучаемых языков.

ГЛАВА 1. ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1. Алгоритмы и величины

Этапы решения задачи на ЭВМ. Работа по решению любой задачи с использованием компьютера делится на следующие этапы:

1. Постановка задачи.
2. Формализация задачи.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Часто эту последовательность называют *технологической цепочкой решения задачи на ЭВМ.* Непосредственно к программированию в этом списке относятся пункты 3, 4, 5.

На этапе постановки задачи должно быть четко сформулировано, *что дано и что требуется найти.* Здесь очень важно определить полный набор исходных данных, необходимых для получения решения.

Второй этап — формализация задачи. Здесь чаще всего задача переводится на язык математических формул, уравнений, отношений. Если решение требует математического описания какого-то реального объекта, явления или процесса, то формализация равносильна получению соответствующей *математической модели.*

Третий этап — построение алгоритма. Опытные программисты часто сразу пишут программы на языках, не прибегая к каким-либо специальным способам описания алгоритмов (блок-схемам, псевдокодам). Однако в учебных целях полезно использовать эти средства, а затем переводить полученный алгоритм на язык программирования.

Первые три этапа предусматривают работу без компьютера. Дальше следует собственно программирование на определенном языке, в определенной системе программирования. Последний (шестой) этап — это использование уже разработанной программы в практических целях.

Таким образом, программист должен обладать следующими знаниями и навыками:

- уметь строить алгоритмы;
- знать языки программирования;
- уметь работать в соответствующей системе программирования.

Основой программистской грамотности является развитое алгоритмическое мышление.

Понятие алгоритма. Одним из фундаментальных понятий в информатике является понятие алгоритма. Происхождение самого термина «алгоритм» связано с математикой. Это слово происходит от *Algorithmi* — латинского написания имени Мухаммеда аль-Хорезми (787—850), выдающегося математика средневекового Востока. В XII в. был выполнен латинский перевод его математического трактата, из которого европейцы узнали о десятичной позиционной системе счисления и правилах арифметики многозначных чисел. Именно эти правила в то время называли алгоритмами. Сложение, вычитание, умножение столбиком, деление уголком многозначных чисел — вот первые алгоритмы в математике. Правила алгебраических преобразований, способы вычислений корней уравнений также можно отнести к математическим алгоритмам.

В наше время понятие алгоритма трактуется шире. *Алгоритм — это последовательность команд управления каким-либо исполнителем.* В школьном курсе информатики с понятием алгоритма, с методами построения алгоритмов ученики знакомятся на примерах учебных исполнителей: Робота, Черепахи, Чертежника и т.д. Эти исполнители ничего не вычисляют. Они создают рисунки на экране, перемещаются в лабиринтах, перетаскивают предметы с места на место. Таких исполнителей принято называть *исполнителями, работающими в обстановке.*

В разделе информатики под названием «Программирование» изучаются методы программного управления работой ЭВМ. Следовательно, в качестве исполнителя выступает компьютер. Компьютер работает с величинами — различными информационными объектами: числами, символами, кодами и т. п. Поэтому алгоритмы, предназначенные для управления компьютером, принято называть *алгоритмами работы с величинами.*

Данные и величины. Совокупность величин, с которыми работает компьютер, принято называть *данными.* По отношению к программе данные делятся на *исходные*, *результаты* (окончательные данные) и *промежуточные* (рис. 1), которые получаются в процессе вычислений.

Например, при решении квадратного уравнения $ax^2 + bx + c = 0$ исходными данными являются коэффициенты a, b, c; результата-

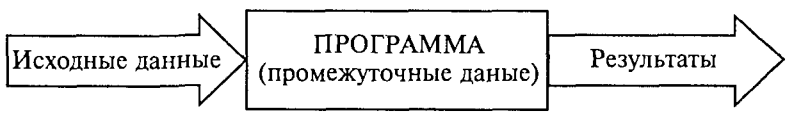


Рис. 1

ми — корни уравнения x_1, x_2; промежуточным данным — дискриминант уравнения $D = b^2 - 4ac$.

Для успешного освоения программирования необходимо усвоить следующее правило: *всякая величина занимает свое определенное место в памяти ЭВМ* (иногда говорят — ячейку памяти). Хотя термин «ячейка» с точки зрения архитектуры современных ЭВМ несколько устарел, однако в учебных целях его удобно использовать.

У всякой величины имеются три основных свойства: *имя*, *значение* и *тип*. На уровне команд процессора величина идентифицируется при помощи адреса ячейки памяти, в которой она хранится. В алгоритмах и языках программирования величины делятся на *константы* и *переменные*. Константа — неизменная величина, и в алгоритме она представляется собственным значением, например: 15, 34.7, *k*, *true* и т.д. Переменные величины могут изменять свои значения в ходе выполнения программы и представляются символическими именами — идентификаторами, например: *X*, *S2*, *cod15*. Любая константа, как и переменная, занимает ячейку памяти, а значение этих величин определяется двоичным кодом в этой ячейке.

Теперь о типах величин — *типах данных*. С понятием типа данных вы уже, возможно, встречались, изучая в курсе информатики базы данных и электронные таблицы. Это понятие является фундаментальным для программирования.

В каждом языке программирования существует своя концепция типов данных, своя система типов. Тем не менее в любой язык входит минимально необходимый набор основных типов данных, к которому относятся: *целый, вещественный, логический* и *символьный* типы. С типом величины связаны три ее характеристики: множество допустимых значений, множество допустимых операций, форма внутреннего представления. В табл. 1.1 представлены эти характеристики основных типов данных.

Таблица 1.1

Тип	Значения	Операции	Внутреннее представление
Целый	Целые положительные и отрицательные числа в некотором диапазоне. Примеры: 23, −12, 387	Арифметические операции с целыми числами: +, −, ·, целое деление и остаток от деления. Операции отношений (<, >, = и др.)	Формат с фиксированной точкой

Окончание табл. 1.1

Тип	Значения	Операции	Внутреннее представление
Вещественный	Любые (целые и дробные) числа в некотором диапазоне. Примеры: 2,5, −0,01, 45,0, $3{,}6 \cdot 10^9$	Арифметические операции: +, −, ·, /. Операции отношений	Формат с плавающей точкой
Логический	True (истина), False (ложь)	Логические операции: И (and), ИЛИ (or), НЕ (not). Операции отношений	1 бит: 1 − true; 0 − false
Символьный	Любые символы компьютерного алфавита. Примеры: '*a*', '5', '+', '$'	Операции отношений	Коды таблицы символьной кодировки. 1 символ − 1 байт

Типы констант определяются по контексту (т.е. по форме записи в тексте), а типы переменных устанавливаются в описаниях переменных.

Есть еще один вариант классификации данных — классификация по структуре. Данные делятся на *простые* и *структурированные*. Для простых величин (их еще называют скалярными) справедливо утверждение: одна величина — одно значение, для структурированных: одна величина — множество значений. К структурированным величинам относятся массивы, строки, множества и т.д.

ЭВМ — исполнитель алгоритмов. Как известно, всякий алгоритм (программа) составляется для конкретного исполнителя в рамках его системы команд. О каком же исполнителе идет речь при обсуждении вопроса о программировании для ЭВМ? Ответ очевиден: исполнителем является компьютер. Точнее говоря, исполнителем является комплекс ЭВМ + Система программирования (СП). Программист составляет программу на том языке, на который ориентирована СП. Иногда в литературе такой комплекс называют виртуальной ЭВМ. Например, компьютер с работающей системой программирования на Бэйсике называют Бэйсик-машиной; компьютер с работающей системой программирования на Паскале называют Паскаль-машиной и т.п. Схематически это изображено на рис. 2.

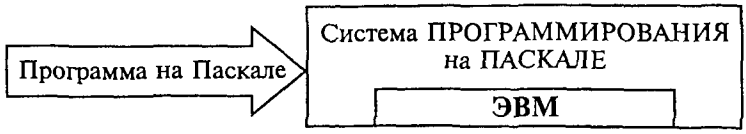


Рис. 2

Входным языком такого исполнителя является язык программирования Паскаль.

Независимо от того, на каком языке программирования будет написана программа, алгоритм решения любой задачи на ЭВМ может быть составлен из команд:

- присваивания;
- ввода;
- вывода;
- обращения к вспомогательному алгоритму;
- цикла;
- ветвления.

Для описания алгоритмов в дальнейшем мы будем использовать блок-схемы и учебный алгоритмический язык, применяемый в школьном курсе информатики.

1.2. Линейные вычислительные алгоритмы

Основным элементарным действием в вычислительных алгоритмах является *присваивание значения переменной величине.* Если значение константы определено видом ее записи, то переменная величина получает конкретное значение только в результате присваивания. Присваивание может осуществляться двумя способами: с помощью команды присваивания и с помощью команды ввода.

Рассмотрим пример. В школьном учебнике математики правила деления обыкновенных дробей описаны так:

1. Числитель первой дроби умножить на знаменатель второй дроби.

2. Знаменатель первой дроби умножить на числитель второй дроби.

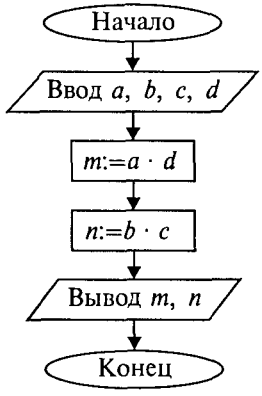
3. Записать дробь, числитель которой есть результат выполнения пункта 1, а знаменатель — результат выполнения пункта 2.

В алгебраической форме это выглядит следующим образом:

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c} = \frac{m}{n}.$$

Построим алгоритм деления дробей для ЭВМ. В этом алгоритме сохраним те же обозначения для переменных, которые использованы в записанной выше формуле. Исходными данными являются

целочисленные переменные *a*, *b*, *c*, *d*. Результатом — также целые величины *m* и *n*. Блок-схема и текст алгоритма на учебном алгоритмическом языке приведены ниже (в дальнейшем для краткости будем обозначать учебный алгоритмический язык буквами АЯ).




```
алг Деление дробей
нач
  цел a,b,c,d,m,n
  ввод a,b,c,d
  m:=a*d
  n:=b*c
  вывод m,n
кон
```

Формат команды присваивания следующий:

```
переменная:=выражение
```

Знак «:=» нужно читать как «присвоить».

Команда присваивания обозначает следующие действия, выполняемые компьютером:

1. Вычисляется *выражение*.
2. Полученное значение присваивается *переменной*.

В приведенном выше алгоритме присутствуют две команды присваивания. В блок-схемах команда присваивания записывается в прямоугольнике. Такой блок называется вычислительным блоком.

В описаниях алгоритмов необязательно соблюдать строгие правила в записи выражений. Их можно писать в обычной математической форме. Это еще не язык программирования со строгим синтаксисом.

В приведенном алгоритме присутствует команда ввода:

```
ввод a,b,c,d
```

В блок-схеме команда ввода записывается в параллелограмме — блоке ввода-вывода. При выполнении данной команды процессор прерывает работу и ожидает действий пользователя. Пользователь должен набрать на устройстве ввода (клавиатуре) значения вводимых переменных и нажать на клавишу ввода **Enter**. Значения следует вводить в том же порядке, в каком соответствующие переменные расположены в списке ввода. Обычно с помощью команды ввода присваиваются значения исходных данных, а команда присваивания используется для получения промежуточных и конечных величин.

Полученные компьютером результаты решения задачи должны быть сообщены пользователю. Для этих целей предназначена команда вывода:

```
вывод m,n
```

С помощью этой команды результаты выводятся на экран или на устройство печати на бумагу.

Поскольку присваивание является важнейшей операцией в вычислительных алгоритмах, обсудим ее более подробно.

Рассмотрим последовательное выполнение четырех команд присваивания, в которых участвуют две переменные величины a и b.

В приведенной ниже таблице напротив каждой команды присваивания указываются значения переменных, которые устанавливаются после ее выполнения.

Команда	a	b
$a := 1$	1	–
$b := 2 \cdot a$	1	2
$a := b$	2	2
$b := a + b$	2	4

Этот пример иллюстрирует три основных свойства команды присваивания:

- пока переменной не присвоено значение, она остается неопределенной;
- значение, присвоенное переменной, сохраняется в ней вплоть до выполнения следующей команды присваивания этой переменной;
- новое значение, присваиваемое переменной, заменяет ее предыдущее значение.

Рассмотрим один очень полезный алгоритм, который приходится часто использовать при программировании. Даны две величины: X и Y. Требуется произвести между ними обмен значениями. Например, если первоначально было $X = 1$, $Y = 2$, то после обмена должно стать: $X = 2$, $Y = 1$.

Хорошей моделью для решения этой задачи является следующая ситуация: имеются два стакана — один с молоком, другой с водой. Требуется произвести обмен их содержимым. Всякому ясно, что в этом случае нужен дополнительный третий пустой стакан. Последовательность действий будет следующей: 1) перелить из первого стакана в третий; 2) перелить из второго в первый; 3) перелить из третьего во второй. Цель достигнута!

По аналогии для обмена значениями двух переменных нужна третья дополнительная переменная. Назовем ее Z. Тогда задача

обмена решается последовательным выполнением трех команд присваивания:

Команда	X	Y	Z
ввод X, Y	1	2	–
$Z:=X$	1	2	1
$X:=Y$	2	2	1
$Y:=Z$	2	1	1

Аналогия со стаканами не совсем точна в том смысле, что при переливании из одного стакана в другой первый становится пустым. В результате же присваивания ($X:=Y$) переменная, стоящая справа (Y), сохраняет свое значение.

Алгоритм для деления дробей имеет линейную структуру. В нем все команды выполняются в строго однозначной последовательности, каждая по одному разу. Линейный алгоритм составляется из команд присваивания, ввода, вывода и обращения к вспомогательным алгоритмам (об этом позже).

При описании алгоритмов в блок-схемах типы, как правило, не указываются (но подразумеваются). В алгоритмах на АЯ для всех переменных типы указываются явно. Описание типов переменных производится сразу после заголовка алгоритма. В них используются следующие обозначения типов: **цел** — целый тип, **вещ** — вещественный тип, **лит** — символьный (литерный) тип, **лог** — логический тип. В алгоритме для деления дробей для всех переменных указан целый тип.

1.3. Ветвления и циклы в вычислительных алгоритмах

Составим алгоритм решения квадратного уравнения

$$ax^2 + bx + c = 0.$$

Задача хорошо знакома из математики. Исходными данными здесь являются коэффициенты a, b, c. Решением в общем случае будут два корня x_1 и x_2, которые вычисляются по формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Все используемые в этой программе величины вещественного типа.

```
алг корни квадратного уравнения
вещ a,b,c,x1,x2,d
нач ввод a,b,c
  d:=b² - 4ac
  x1:=(-b+√d)/(2a)
  x2:=(-b-√d)/(2a)
  вывод x1,x2
кон
```

Слабость такого алгоритма видна невооруженным глазом. Он не обладает важнейшим свойством, предъявляемым к качественным алгоритмам, — универсальностью по отношению к исходным данным. *Какими бы ни были значения исходных данных, алгоритм должен приводить к определенному результату и завершать работу.* Результатом может быть число, но может быть и сообщение о том, что при определенных данных задача решения не имеет. Недопустимы остановки в середине алгоритма из-за невозможности выполнить какую-то операцию. Упомянутое свойство в литературе по программированию называют результативностью алгоритма (в любом случае должен быть получен какой-то результат).

Чтобы построить универсальный алгоритм, сначала требуется тщательно проанализировать математическое содержание задачи.

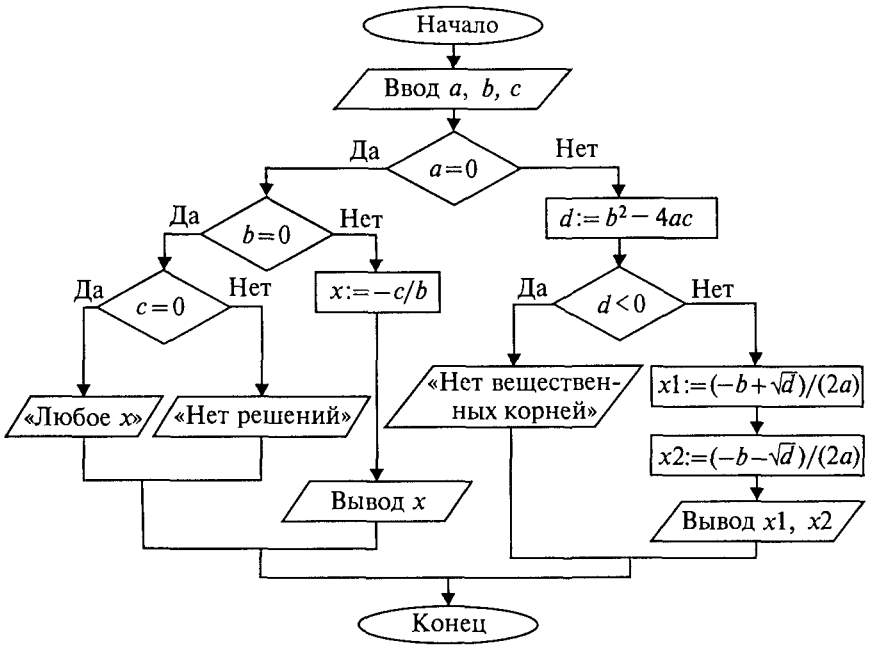


Рис. 3

Решение уравнения зависит от значений коэффициентов a, b, c. Вот анализ рассмотренной выше задачи (ограничиваемся только поиском вещественных корней):

если $a = 0$, $b = 0$, $c = 0$, то любое x — решение уравнения;

если $a = 0$, $b = 0$, $c \neq 0$, то уравнение действительных решений не имеет;

если $a = 0$, $b \neq 0$, то это линейное уравнение, которое имеет одно решение $x = -c/b$;

если $a \neq 0$ и $d = b^2 - 4ac \geq 0$, то уравнение имеет два вещественных корня (формулы приведены выше);

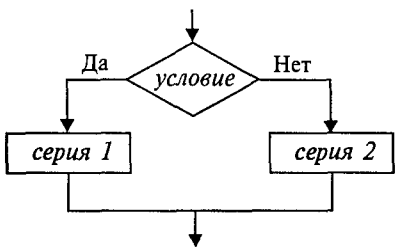
если $a \neq 0$ и $d < 0$, то уравнение не имеет вещественных корней.

Блок-схема алгоритма приведена на рис. 3.

Этот же алгоритм на алгоритмическом языке:

```
алг корни квадратного уравнения
вещ a,b,c,d,x1,x2
нач ввод a,b,c
  если a=0
  то   если b=0
       то   если c=0
            то вывод «Любое x — решение»
            иначе вывод «Нет решений»
            кв
       иначе  x:=-c/b
            вывод x
    кв
  иначе d:=b²-4ac
  если d<0
  то вывод «Нет вещественных корней»
  иначе x1:=(-b+√d)/(2a);x2:=(-b-√d)/(2a)
            вывод «x1=»,x1,«x2=»,x2
  кв
  кв
кон
```

В этом алгоритме многократно использована *структурная команда ветвления*. Общий вид команды ветвления в блок-схемах и на алгоритмическом языке следующий:




```
если условие
то серия 1
иначе серия 2
кв
```

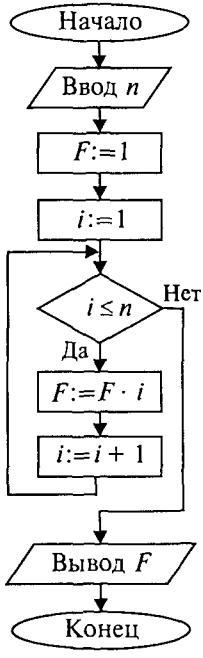
Вначале проверяется *условие* (вычисляется отношение, логическое выражение). Если условие истинно, то выполняется *серия 1* — последовательность команд, на которую указывает стрелка с надписью «да» (положительная ветвь). В противном случае выполняется *серия 2* (отрицательная ветвь). В АЯ условие записывается после служебного слова **если**, положительная ветвь — после слова **то**, отрицательная — после слова **иначе**. Буквы **кв** обозначают конец ветвления.

Если на ветвях одного ветвления содержатся другие ветвления, то такой алгоритм имеет структуру *вложенных ветвлений*. Именно такую структуру имеет алгоритм «Корни квадратного уравнения».

Рассмотрим следующую задачу: дано целое положительное число n. Требуется вычислить $n!$ (n-факториал). Вспомним определение факториала:

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \cdot 2 \ldots n, & \text{если } n \geq 1. \end{cases}$$

На рис. 4 приведена блок-схема алгоритма. В нем используются три переменные целого типа: n — аргумент; i — промежуточная переменная; F — результат. Для проверки правильности алгоритма построена трассировочная таблица. В такой таблице для конк-



Шаг	n	F	i	Условие
1	3			
2		1		
3			1	
4				1≤3, да
5		1		
6			2	
7				2≤3, да
8		2		
9			3	
10				3≤3, да
11		6		
12			4	
13				4≤3, нет
14		вывод		

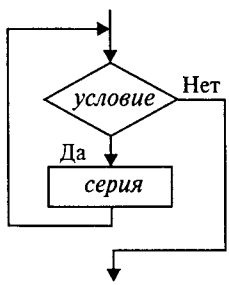
Рис. 4

ретных значений исходных данных по шагам прослеживается изменение переменных, входящих в алгоритм. Данная таблица составлена для случая $n = 3$.

Трассировка доказывает правильность алгоритма. Теперь запишем этот алгоритм на алгоритмическом языке.

```
алг Факториал
цел n,i,F
нач ввод n
    F:=1;i:=1
    пока i≤n, повторять
    нц F:=F*i
       i:=i+1
    кц
    вывод F
кон
```

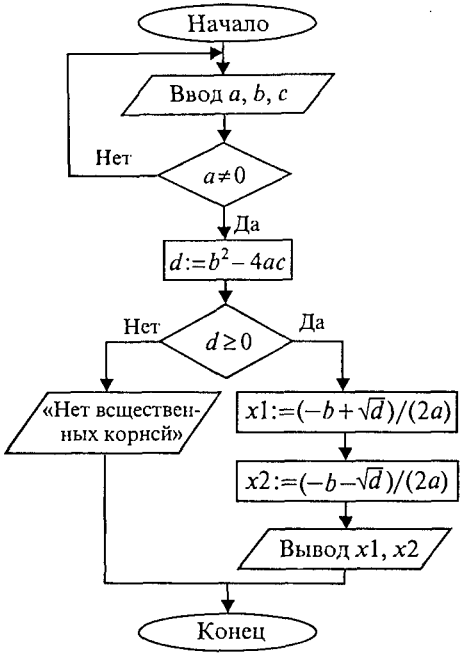
Этот алгоритм имеет циклическую структуру. В алгоритме использована структурная команда **цикл-пока**, или **цикл с предусловием**. Общий вид команды **цикл-пока** в блок-схемах и в АЯ следующий:




```
пока условие, повторять
нц
  серия
кц
```

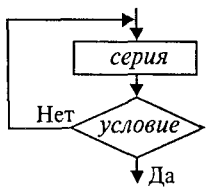
Выполнение серии команд (тела цикла) повторяется, пока условие цикла истинно. Когда условие становится ложным, цикл заканчивает выполнение. Служебные слова **нц** и **кц** обозначают начало цикла и конец цикла соответственно.

Цикл с предусловием — это основная, но не единственная форма организации циклических алгоритмов. Другим вариантом является цикл с постусловием. Вернемся к алгоритму решения квадратного уравнения. К нему можно подойти с такой позиции: если $a = 0$, то это уже не квадратное уравнение и его можно не рассматривать. В таком случае будем считать, что пользователь ошибся при вводе данных, и следует предложить ему повторить ввод. Иначе говоря, в алгоритме будет предусмотрен контроль достоверности исходных данных с предоставлением пользователю возможности исправить ошибку. Наличие такого контроля — еще один признак хорошего качества программы.




```
алг квадратное
уравнение
вещ a,b,c,d,x1,x2
нач
  повторять
     ввод a,b,c
  до a≠0
  d:=b²-4ac
  если d≥0
  то x1:=(-b+√d)/(2a)
     x2:=(-b-√d)/(2a)
     вывод x1,x2
  иначе
     вывод «Нет
     вещественных
     корней»
  кв
кон
```

В общем виде структурная команда **цикл с постусловием** или **цикл — до** представляется так:




```
повторять
   серия
до условие
```

Здесь используется условие окончания цикла. Когда оно становится истинным, цикл заканчивает работу.

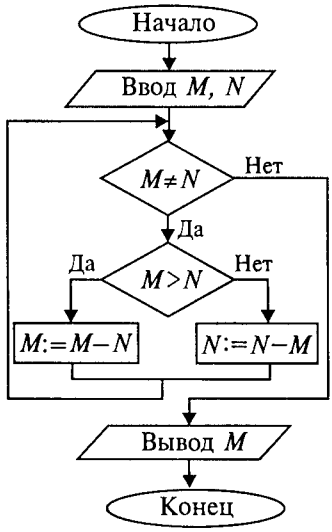
Составим алгоритм решения следующей задачи: даны два натуральных числа M и N. Требуется вычислить их наибольший общий делитель — НОД(M, N).

Эта задача решается с помощью метода, известного под названием алгоритма Евклида. Его идея основана на том свойстве, что если $M > N$, то НОД(M, N) = НОД($M - N$, N). Попробуйте самостоятельно доказать это свойство. Другой факт, лежащий в основе алгоритма, тривиален — НОД(M, M) = M. Для «ручного» выполнения этот алгоритм можно описать в форме следующей инструкции:

1. Если числа равны, то взять их общее значение в качестве ответа; в противном случае продолжить выполнение алгоритма.

2. Определить большее из чисел.
3. Заменить большее число разностью большего и меньшего значений.
4. Вернуться к выполнению пункта 1.

Блок-схема и алгоритм на АЯ будут следующими:




```
алг Евклид
цел M,N
нач ввод M,N
пока M≠N, повторять
нц если  M>N
   то M:=M-N
   иначе N:=N-M
   кв
кц
кон
```

Алгоритм имеет структуру цикла с вложенным ветвлением. Проделайте самостоятельно трассировку этого алгоритма для случая $M = 18$, $N = 12$. В результате получится НОД = 6, что, очевидно, верно.

1.4. Вспомогательные алгоритмы и процедуры

В теории алгоритмов известно понятие *вспомогательного* алгоритма. Вспомогательным называется алгоритм решения некоторой подзадачи из основной решаемой задачи. В таком случае алгоритм решения исходной задачи называется *основным* алгоритмом.

В качестве примера рассмотрим следующую задачу: требуется составить алгоритм вычисления степенной функции с целым показателем $y = x^k$, где k — целое число, $x \neq 0$. В алгебре такая функция определена следующим образом:

$$x^n = \begin{cases} 1 & n = 0, \\ \dfrac{1}{x^{-n}} & n < 0, \\ x^n & n > 0. \end{cases}$$

Для данной задачи в качестве подзадачи можно рассматривать возведение числа в целую положительную степень.

Учитывая, что $1/x^{-n} = (1/x)^{-n}$, запишем основной алгоритм решения этой задачи.

```
алг Степенная функция
цел n; вещ x,y
нач ввод x,n
  если n=0
  то  y:=1
  иначе  если   n>0
         то  СТЕПЕНЬ(x,n,y)
         иначе  СТЕПЕНЬ(1/x,-n,y)
         кв
кв
вывод y
кон
```

Здесь дважды присутствует команда обращения к вспомогательному алгоритму с именем СТЕПЕНЬ. Это алгоритм возведения вещественного основания в целую положительную степень путем его многократного перемножения. Величины, стоящие в скобках в команде обращения к вспомогательному алгоритму, называются *фактическими параметрами*.

В учебном алгоритмическом языке вспомогательные алгоритмы оформляются в виде процедур. Запишем на АЯ процедуру СТЕПЕНЬ.

```
процедура СТЕПЕНЬ(вещ a,цел k,вещ z)
цел i
нач z:=1; i:=1
    пока i≤k, повторять
    нц z:=z*a
       i:=i+1
    кц
кон
```

Заголовок вспомогательного алгоритма начинается со слова «процедура», после которого следует имя процедуры и в скобках — список формальных параметров. В этом списке перечисляются переменные-аргументы и переменные-результаты с указанием их типов. Здесь a и k — формальные параметры-аргументы, z — параметр-результат. Следовательно, процедура СТЕПЕНЬ производит вычисления по формуле $z = a^k$. В основном алгоритме «Степенная функция» обращение к процедуре производится путем указания ее имени с последующим в скобках списком *фактических* параметров. Между формальными и фактическими параметрами процедуры должны выполняться следующие правила соответствия:

• по количеству (сколько формальных, столько и фактических параметров);

• по последовательности (первому формальному соответствует первый фактический параметр, второму — второй и т.д.);

• по типам (типы соответствующих формальных и фактических параметров должны совпадать).

Фактические параметры-аргументы могут быть выражениями соответствующего типа.

Обращение к процедуре инициирует следующие действия:

1. Значения параметров-аргументов присваиваются соответствующим формальным параметрам.

2. Выполняется тело процедуры (команды внутри процедуры).

3. Значение результата передается соответствующему фактическому параметру, и происходит переход к выполнению следующей команды основного алгоритма.

В процедуре СТЕПЕНЬ нет команд ввода исходных данных и вывода результатов. Здесь присваивание начальных значений аргументам (a, n) производится через передачу параметров-аргументов. А присваивание результата переменной (y) происходит через передачу параметра-результата (z). Таким образом, передача значений параметров процедур — это третий способ присваивания (наряду с командой присваивания и командой ввода).

Использование процедур позволяет строить сложные алгоритмы методом *последовательной детализации.*

Упражнения

1. Даны декартовы координаты трех вершин треугольника на плоскости. Составить алгоритм определения площади треугольника.

2. Дана скорость ракеты при выходе за пределы атмосферы Земли. Составить алгоритм определения того, как будет двигаться ракета после выключения двигателей. (Напомним величины трех космических скоростей: 7,5 км/с; 11,2 км/с; 16,4 км/с.)

3. Даны три положительных числа. Составить алгоритм, определяющий, могут ли они быть длинами сторон треугольника.

4. Пусть компьютер способен выполнять только две арифметические операции — сложение и вычитание. Составить алгоритмы:

а) умножения двух целых чисел;

б) целочисленного деления двух чисел;

в) получения остатка от целочисленного деления двух чисел.

5. Построить алгоритм решения биквадратного уравнения, используя как вспомогательный алгоритм решения квадратного уравнения.

6. Составить алгоритм нахождения НОД трех натуральных чисел, используя вспомогательный алгоритм нахождения НОД двух чисел.

ГЛАВА 2. ВВЕДЕНИЕ В ЯЗЫКИ ПРОГРАММИРОВАНИЯ

2.1. История и классификация языков программирования

Язык программирования — это способ записи программ решения различных задач на ЭВМ в понятной для компьютера форме. Процессор компьютера непосредственно понимает язык машинных команд (ЯМК). Программы на ЯМК программисты писали лишь для самых первых ламповых машин — ЭВМ первого поколения. Программирование на ЯМК — дело непростое. Программист должен знать числовые коды всех машинных команд, должен сам распределять память под команды программы и данные.

В 1950-х гг. появляются первые средства автоматизации программирования — языки Автокоды. Позднее для языков этого уровня стало применяться название «Ассемблеры». Появление языков типа Автокод-Ассемблер облегчило участь программистов. Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические (словесные) обозначения, которые легче запомнить. Язык программирования стал понятнее для человека, но при этом удалился от языка машинных команд. Чтобы компьютер мог исполнять программы на Автокоде, потребовался специальный переводчик — транслятор. Транслятор — это системная программа, переводящая текст программы на Автокоде в текст эквивалентной программы на ЯМК.

Компьютер, оснащенный транслятором с Автокода, понимает Автокод. В этом случае можно говорить о псевдо-ЭВМ (аппаратура плюс транслятор с Автокода), языком которой является Автокод. Языки типа Автокод-Ассемблер являются машинно-ориентированными, т.е. они настроены на структуру машинных команд конкретного компьютера. Разные компьютеры с разными типами процессоров имеют разный Ассемблер. Языки программирования высокого уровня (ЯПВУ) являются машинно-независимыми языками. Одна и та же программа на таком языке может быть выполнена на ЭВМ разных типов, оснащенных соответствующим транслятором. Форма записи программ на ЯПВУ по сравнению с Автокодом еще ближе к традиционной математической форме, к естественному языку. Очень скоро вы увидите, что, например, на языке Паскаль она почти такая же, как на школьном Алгоритмическом языке. ЯПВУ легко изучаются, хорошо поддерживают структурную методику программирования.

Первыми популярными языками высокого уровня, появившимися в 1950-х гг., были Фортран, Кобол (в США) и Алгол (в Европе). Языки Фортран и Алгол были ориентированы на научно-технические расчеты математического характера. Кобол — язык для программирования экономических задач. В Коболе по сравнению с двумя другими названными языками слабее развиты математические средства, но зато хорошо развиты средства обработки текстов, организация вывода данных в форме требуемого документа. Для первых ЯПВУ предметная ориентация языков была характерной чертой.

Большое количество языков программирования появилось в 1960—1970-х гг. А за всю историю ЭВМ их было создано более тысячи. Но распространились, выдержали испытание временем немногие. В 1965 г. в Дартмутском университете был разработан язык Бейсик. По замыслу авторов это простой язык, легко изучаемый, предназначенный для программирования несложных расчетных задач. Наибольшее распространение Бейсик получил на микроЭВМ и персональных компьютерах. На некоторых моделях школьных компьютеров программировать можно только на Бейсикс. Однако Бейсик — неструктурный язык, и потому он плохо подходит для обучения качественному программированию. Справедливости ради следует заметить, что последние версии Бейсика для ПК (например, QBasic) стали более структурными и по своим изобразительным возможностям приближаются к таким языкам, как Паскаль.

В эпоху ЭВМ третьего поколения получил большое распространение язык PL/1 (*Program Language One*), разработанный фирмой IBM. Это был первый язык, претендовавший на универсальность, т.е. на возможность решать любые задачи: вычислительные, обработки текстов, накопления и поиска информации. Однако PL/1 оказался слишком сложным языком. Для машин типа IBM 360/370 транслятор с него не мог считаться оптимальным, содержал ряд невыявленных ошибок. На ЭВМ класса мини и микро он вообще не получил распространения. Однако тенденция к универсализации языков оказалась перспективной. Старые языки были модернизированы в универсальные варианты — Алгол-68, Фортран-77.

Значительным событием в истории языков программирования стало создание в 1971 г. языка Паскаль. Его автор — швейцарский профессор Н. Вирт — разрабатывал Паскаль как учебный язык структурного программирования.

Наибольший успех в распространении этого языка обеспечили персональные компьютеры. Фирма *Borland International, Inc* (США) разработала систему программирования Турбо Паскаль для ПК. Турбо Паскаль — это не только язык и транслятор с него, но еще и операционная оболочка, обеспечивающая пользователю удобство работы. Турбо Паскаль вышел за рамки учебного предназначения и стал языком профессионального программирования с

универсальными возможностями. Транслятор с Турбо Паскаля по оптимальности создаваемых им программ близок наиболее удачному в этом отношении транслятору — транслятору с Фортрана. В силу названных достоинств Паскаль стал основой многих современных языков программирования, например, таких как Ада, Модула-2 и др.

Язык программирования Си (английское название — *C*) создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ. Так же как и Паскаль, Си — это язык структурного программирования, но, в отличие от Паскаля, в нем заложены возможности непосредственного обращения к некоторым машинным командам, к определенным участкам памяти компьютера. Дальнейшее развитие Си привело к созданию языка объектно-ориентированного программирования Си++.

Модула-2 — это еще один язык, предложенный Н.Виртом, основанный на языке Паскаль и содержащий средства для создания больших программ.

ЭВМ будущего, пятого поколения называют машинами «искусственного интеллекта». Но прототипы языков для этих машин были созданы существенно раньше их физического появления. Это языки ЛИСП и Пролог.

ЛИСП появился в 1965 г. Язык ЛИСП основан на понятии рекурсивно определенных функций. А поскольку доказано, что любой алгоритм может быть описан с помощью некоторого набора рекурсивных функций, то ЛИСП, по сути, является универсальным языком. С его помощью на ЭВМ можно моделировать достаточно сложные процессы, в частности интеллектуальную деятельность людей.

Язык Пролог разработан во Франции в 1972 г. также для решения проблемы «искусственного интеллекта». Пролог позволяет в формальном виде описывать различные утверждения, логику рассуждений и заставляет ЭВМ давать ответы на заданные вопросы.

Реализовать тот или иной язык программирования на ЭВМ — это значит создать транслятор с этого языка для данной ЭВМ. Существуют два принципиально различных метода трансляции. Они называются соответственно *компиляция* и *интерпретация*. Для объяснения их различия можно предложить следующую аналогию: лектор должен выступить перед аудиторией на незнакомом ей языке. Перевод можно организовать двумя способами:

- полный предварительный перевод — лектор заранее передает текст выступления переводчику, тот записывает перевод, размножает его и раздает слушателям (после чего лектор может и не выступать);
- синхронный перевод — лектор читает доклад, переводчик одновременно с ним слово в слово переводит выступление.

Компиляция является аналогом полного предварительного перевода; интерпретация — аналогом синхронного перевода. Транслятор, работающий по принципу компиляции, называется компилятором; транслятор, работающий методом интерпретации, — интерпретатором.

При компиляции в память ЭВМ загружается программа-компилятор. Она воспринимает текст программы на ЯПВУ как исходную информацию. После завершения компиляции получается программа на языке машинных команд. Затем в памяти остается только программа на ЯМК, которая выполняется, и получаются требуемые результаты.

Интерпретатор в течение всего времени работы программы находится во внутренней памяти. В ОЗУ помещается и программа на ЯПВУ. Интерпретатор в последовательности выполнения алгоритма «читает» очередной оператор программы, переводит его в команды и тут же выполняет эти команды. Затем переходит к переводу и выполнению следующего оператора. При этом результаты предыдущих переводов в памяти не сохраняются. При повторном выполнении одной и той же команды она снова будет транслироваться. При компиляции исполнение программы разбивается на два этапа: трансляцию и выполнение. При интерпретации, поскольку трансляция и выполнение совмещены, программа на ЭВМ проходит в один этап. Однако откомпилированная программа выполняется быстрее, чем интерпретируемая. Поэтому использование компиляторов удобнее для больших программ, требующих быстрого счета. Программы на Паскале, Си, Фортране всегда компилируются. Бейсик чаще всего реализован через интерпретатор.

2.2. Структура и способы описания языков программирования высокого уровня

Во всяком языке программирования определены способы организации данных и способы организации действий над данными. Кроме того, существует понятие «элементы языка», включающее в себя множество символов (алфавит), лексемы и другие изобразительные средства языка программирования. Несмотря на разнообразие указанных языков, их изучение происходит приблизительно по одной схеме. Это связано с общностью структуры различных языков программирования высокого уровня, которая схематически отражена на рис. 5.

Изложение языков Паскаль и Си в данном учебном пособии будет соответствовать этой схеме.

Надо сказать, что в изучении естественных языков и языков программирования есть сходные моменты. Во-первых, для того чтобы читать и писать на иностранном языке, нужно знать *алфа-*

Рис. 5

вит этого языка. Во-вторых, следует знать правописание слов и правила записи предложений, т.е. то, что называется *синтаксисом* языка. В-третьих, важно понимать смысл слов и фраз, чтобы адекватно реагировать на них: ведь из грамотно написанных слов можно составить абсолютно бессмысленную фразу. Например, в салоне самолета засветилось табло, на котором написано: *Fasten belts!* (Пристегните ремни!). Зная правила чтения английского языка, вы, к зависти соседа, правильно прочитаете эту фразу. Однако смысл ее вам может быть непонятен, и поэтому соответствующих действий вы не предпримете, за что получите замечание от стюардессы. Смысловое содержание языковой конструкции называется *семантикой.*

Всякий язык программирования имеет три основные составляющие: *алфавит, синтаксис* и *семантику.*

Соблюдение правил в языке программирования должно быть более строгим, чем в разговорном языке. Человеческая речь содержит значительное количество избыточной информации. Не расслышав какое-то слово, можно понять смысл фразы в целом. Слушающий или читающий человек может додумать, дополнить, исправить ошибки в воспринимаемом тексте.

Компьютер же — автомат, воспринимающий все «всерьез». В текстах программ нет избыточности, компьютер сам не исправит даже очевидной (с точки зрения человека) ошибки. Он может лишь указать на место, которое «не понял», и вывести замечание о предполагаемом характере ошибки. Исправить же ошибку должен программист.

Для описания синтаксиса языка программирования тоже нужен какой-то язык. В этом случае речь идет о *метаязыке* («надъ-

языке»), предназначенном для описания других языков. Наиболее распространенными метаязыками в литературе по программированию являются металингвистические *формулы Бекуса—Наура* (язык БНФ) и *синтаксические диаграммы*. В дальнейшем мы чаще всего будем использовать язык синтаксических диаграмм. Они более наглядны, легче воспринимаются. В некоторых случаях для удобства мы будем обращаться к отдельным элементам языка БНФ.

В БНФ всякое синтаксическое понятие описывается в виде формулы, состоящей из правой и левой части, соединенных знаком ::=, смысл которого эквивалентен словам «по определению есть». Слева от знака ::= записывается имя определяемого понятия (метапеременная), которое заключается в угловые скобки < >, а в правой части записывается формула или диаграмма, определяющая все множество значений, которые может принимать метапеременная.

Синтаксис языка описывается путем последовательного усложнения понятий: сначала определяются простейшие (базовые), затем все более сложные, включающие в себя предыдущие понятия в качестве составляющих.

В такой последовательности, очевидно, конечным определяемым понятием должно быть понятие программы.

В записях метаформул приняты определенные соглашения. Например, формула БНФ, определяющая понятие «двоичная цифра», выглядит следующим образом:

```
<двоичная цифра>::=0|1
```

Значок | эквивалентен слову «или». Это определение можно представить на языке синтаксических диаграмм (рис. 6).

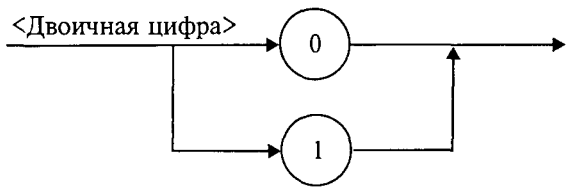


Рис. 6

В диаграммах стрелки указывают на последовательность расположения элементов синтаксической конструкции; кружками обводятся символы, присутствующие в конструкции.

Понятие «двоичный код» как непустую последовательность двоичных цифр БНФ описывает так:

```
<двоичный код>::=<двоичная цифра>|<двоичный
                 код><двоичная цифра>
```

Определение, в котором некоторое понятие определяется само через себя, называется *рекурсивным*. Рекурсивные определения характерны для БНФ.

Синтаксическая диаграмма двоичного кода представлена на рис. 7.

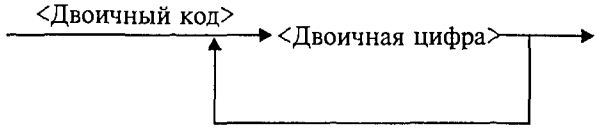


Рис. 7

Возвратная стрелка обозначает возможность многократного повторения. Очевидно, что диаграмма более наглядна, чем БНФ.

Синтаксические диаграммы были введены Н. Виртом и использованы для описания созданного им языка Паскаль. В следующем разделе, посвященном Паскалю, мы также будем пользоваться синтаксическими диаграммами. В литературе по языку Си использование синтаксических диаграмм не принято. Поэтому в гл. 4 будет применяться традиционный для этого языка способ описания правил синтаксиса.

ГЛАВА 3. ПРОГРАММИРОВАНИЕ НА ПАСКАЛЕ

3.1. Первое знакомство с Паскалем

Структура программы на Паскале. По определению стандартного Паскаля программа состоит из *заголовка* программы и *тела* программы (блока), за которым следует *точка* — признак конца программы. В свою очередь, блок содержит разделы *описаний* и раздел *операторов*.

```
Program <имя программы>;
Label <раздел меток>;
Const <раздел констант>;
Type <раздел типов>;
Var <раздел переменных>;
Procedure (Function) <раздел подпрограмм>;
Begin
<раздел операторов>
End.
```

Раздел операторов имеется в любой программе и является основным. Предшествующие разделы носят характер описаний и не все обязательно присутствуют в каждой программе.

В Турбо Паскале, в отличие от стандарта, возможно следующее:

- отсутствие заголовка программы;
- разделы `Const, Type, Var, Label` могут следовать друг за другом в любом порядке и встречаться в разделе описаний сколько угодно раз.

Примеры программ. Уже было сказано, что Паскаль разрабатывался Н. Виртом как учебный язык. Основной принцип, заложенный в нем, — это поддержка структурной методики программирования. Этот же принцип лежит в основе псевдокода, который мы здесь называем Алгоритмическим языком (АЯ). По сути дела, расхождение между АЯ и Паскалем заключается в следующем: АЯ — русскоязычный, Паскаль — англоязычный; синтаксис Паскаля определен строго и однозначно в отличие от сравнительно свободного синтаксиса АЯ.

Запись программы на Паскале похожа на английский перевод алгоритма, записанного на Алгоритмическом языке. Сравните алгоритм деления простых дробей, записанный на АЯ, с соответствующей программой на Паскале.

```
алг деление дробей;
цел a,b,c,d,m,n;
нач ввод(a,b,c,d);
    m:=a*d;
    n:=b*c;
    вывод(m,n)
кон.
```

```
Program Division;
Var a,b,c,d,m,n: Integer;
Begin ReadLn(a,b,c,d);
      m:=a*d;
      n:=b*c;
      WriteLn(m,n)
End.
```

Здесь использовано следующее равенство:

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c}.$$

Даже не заглядывая в учебник по Паскалю, в данной программе можно все понять. Для этого достаточно знать английский язык.

Заголовок программы начинается со слова `Program` (программа), за которым следует произвольное имя, придуманное программистом (*division* — деление). Раздел описания переменных начинается со слова `Var` (*variables* — переменные), за которым следует список переменных. Тип указывается после двоеточия словом `Integer` — целый. Начало и конец раздела операторов программы отмечаются словами `Begin` (начало) и `End` (конец). В конце программы *обязательно ставится точка*.

Ввод исходных данных с клавиатуры производится с помощью процедуры `ReadLn` (*read line* — читать строку). На клавиатуре набирают четыре числа, отделяемые друг от друга пробелами, которые отражаются строкой на экране дисплея. После набора чисел нажимают на клавишу ввода.

Операторы присваивания в Паскале записываются так же, как в АЯ. Знак умножения — * (звездочка).

Вывод результатов на экран дисплея производится с помощью процедуры `WriteLn` (*write line* — писать в строку). В рассмотренном примере два целых числа *m* и *n* выведутся в строчку, курсор на экране перейдет в начало следующей свободной строки и работа программы завершится.

Необходимо строгое соблюдение правил правописания (синтаксиса) программы. В частности, в Паскале однозначно определено назначение знаков пунктуации. Точка с запятой (`;`) ставится в конце заголовка программы, в конце раздела описания переменных, после каждого оператора. Перед словом `End` точку с запятой можно не ставить. Запятая (`,`) является разделителем элементов во всевозможных списках: списке переменных в разделе описания, списке вводимых и выводимых величин.

Строгий синтаксис в языке программирования необходим прежде всего для транслятора. Транслятор — это программа, которая исполняется формально. Если, допустим, разделителем в списке перемен-

ных должна быть запятая, то любой другой знак будет восприниматься как ошибка. Если точка с запятой является разделителем операторов, то транслятор в качестве оператора воспринимает всю часть текста программы от одной точки с запятой до другой. Если вы забыли поставить этот знак между какими-то двумя операторами, то транслятор будет принимать их за один, что неизбежно приведет к ошибке.

Основное назначение синтаксических правил — придать однозначный смысл языковым конструкциям. Если какая-то конструкция может трактоваться двусмысленно, значит, в ней обязательно содержится ошибка. Лучше не полагаться на интуицию, а выучить правила языка.

В дальнейшем мы строго опишем синтаксические правила Паскаля, а пока для получения первоначального представления о языке обратимся еще к нескольким примерам программирования несложных алгоритмов.

«Оттранслируем» алгоритм вычисления факториала натурального числа ($N!$) на Паскале.

```
алг ФАКТОРИАЛ;
цел N,I,F;
нач ввод(N);
       F:=1;
       I:=1;
       пока I<=N
       нц F:=F*I;
          I:=I+1
       кц;
       вывод(F)
кон.
```

```
Program FACTORIAL;
Var N,I,F: Integer;
Begin ReadLn(N);
      F:=1;
      I:=1;
      While I<=N Do
      Begin F:=F*I;
            I:=I+1
      End;
      WriteLn(F)
End.
```

Из этого примера, во-первых, видно, как записывается на Паскале оператор *цикла с предусловием* (цикл-пока):

While <условие выполнения> **Do** <тело цикла>

(`While` — пока, `Do` — делать). Если тело цикла содержит последовательность операторов, то говорят, что оно образует *составной оператор*, в начале и в конце которого надо писать `Begin` и `End`. Служебные слова `Begin` и `End` часто называют *операторными скобками*, которые объединяют несколько операторов в один составной. Если же тело цикла — один оператор (не составной), то операторных скобок не требуется. Тогда транслятор считает, что тело цикла заканчивается на ближайшем знаке «`;`».

Во-вторых, из примера видно, что в Паскале нет специальных слов для обозначения начала цикла (`нц`) и конца цикла (`кц`). На все случаи есть универсальные слова `Begin` и `End`.

Рассмотрим еще один пример программы — решение квадратного уравнения.

```
алг КОРНИ;
вещ a,b,c,d,x1,x2;
нач
 повторять
  вывод(«введите a,b,c;
          a≠0»);
  ввод(a,b,c);
  до a≠0;
  d:=b^2-4*a*c;
  если d≥0
  то
  x1:=(-b+sqrt(d))/(2*a);
  x2:=(-b-sqrt(d))/(2*a);
  вывод(x1,x2)
  иначе вывод(«нет вещ.
           корней»);
  кв
кон.
```

```
Program ROOTS;
Var a,b,c,d,x1,x2: Real;
Begin {ввод данных с
        контролем}
   Repeat
     WriteLn('введите a,
          b,c;a<>0');
     ReadLn(a,b,c)
   Until a=0;
{вычисление дискриминанта}
   d:=b*b-4*a*c;
   If d>=0
   Then Begin {есть корни}
   x1:=(-b+sqrt(d))/2*a);
   x2:=(-b-sqrt(d))/2*a);
    WriteLn('x1=',x1,
            'x2=',x2);
        End
   Else WriteLn('нет вещ.
                 корней')
End.
```

В этой программе по сравнению с предыдущими появилось много новых элементов. Имя вещественного типа в Паскале — `real`.

Цикл с постусловием (цикл-до) программируется оператором

Repeat <тело цикла> **Until** <условие окончания>

(здесь `Repeat` — повторять, `Until` — до). Тело цикла может быть как одиночным, так и составным оператором, однако употребления `Begin` и `End` не требуется, поскольку сами слова `Repeat` и `Until` выполняют роль операторных скобок.

Знак *не равно* в Паскале пишется так: <>, знак *больше или равно*: >=.

Правила записи арифметических выражений мы подробно рассмотрим немного позже. В формулах вычисления корней используется стандартная функция квадратного корня ($\sqrt{x}$), которая в Паскале записывается так: `sqrt(x)`. Порядок выполнения операций в выражении определяется скобками и старшинством операций. Старшинство операций такое же, как и в алгебре. Операции одинакового старшинства выполняются в порядке их записи (слева направо).

Ветвление в Паскале программируется с помощью *условного оператора*, который имеет следующую форму:

If <условие> **Then** <оператор 1> **Else** <оператор 2>

(здесь If — если, Then — то, Else — иначе). Операторы 1 и 2 могут быть как простыми, так и составными. Составной оператор следует заключать в операторные скобки Begin и End.

Так же, как и в Алгоритмическом языке, возможно использование *неполной формы условного оператора*:

if <условие> **then** <оператор>

Характерной чертой данной программы является использование в тексте комментариев. *Комментарий* — это любая последовательность символов, заключенных в фигурные скобки {...}. Можно употреблять также следующие ограничители комментариев (*...*). Комментарий не определяет никаких действий программы и является лишь пояснительным текстом. Он может присутствовать в любом месте программы, где можно поставить пробел. Программист пишет комментарии не для компьютера, а для себя. Комментарий придает тексту программы бóльшую ясность. Хорошо откомментированные программы называют самодокументированными. Во многих подобных программах объем комментариев превышает объем вычислительных операторов.

Удачное использование комментариев — признак хорошего стиля программирования.

Чтобы выполнить программу на ЭВМ, ее нужно *ввести в память, оттранслировать* и *исполнить*. Для того чтобы проделать всю эту работу, на компьютере должны быть специальные средства программного обеспечения. На ПК они составляют *систему Турбо Паскаль*.

Упражнения

«Оттранслируйте» с Алгоритмического языка на Паскаль следующие алгоритмы:

а) алгоритм Евклида;

б) алгоритм выбора наибольшего значения из трех;

в) алгоритм определения существования треугольника с данными длинами сторон;

г) алгоритм умножения двух целых чисел с использованием только операций сложения и вычитания;

д) алгоритм вычисления частного и остатка от целочисленного деления.

3.2. Некоторые сведения о системе Турбо Паскаль

Название Турбо Паскаль обычно воспринимается в двух смыслах:

• как диалект языка Паскаль, представляющий собой расширение стандартного Паскаля;

• как система программирования Турбо Паскаль, являющаяся совокупностью системных программ, предназначенных для создания, отладки и выполнения Паскаль-программ.

В дальнейшем мы будем рассматривать именно Турбо Паскаль, так как он реализован на основных типах персональных компьютеров (IBM PC и совместимых с ними).

Чтобы не было терминологической путаницы, договоримся, что название Турбо Паскаль обозначает язык программирования. Стандартный Паскаль входит в Турбо Паскаль как подмножество. Далее везде, где говорится о расширенных возможностях Турбо-варианта по сравнению со стандартом, это будет оговариваться.

Систему программирования Турбо Паскаль назовем кратко Турбо-системой. Турбо-система обеспечивает удобную операционную обстановку для работы программиста. Но ее назначение не общее, как, например, у оболочки операционной системы MS DOS Norton Commander, а специализированное — предоставлять пользователю необходимые средства работы с Паскаль-программой.

Турбо-система опирается в своей работе на возможности операционной системы. Поэтому каждая конкретная Турбо-система может работать с определенной операционной системой, ее конкретными версиями. Например, Турбо-система на IBM PC работает в среде MS DOS, причем более развитые версии Турбо-системы требуют и более высокоразвитых версий DOS.

Турбо Паскаль (как язык программирования и как операционная оболочка) значительно изменился за историю своего существования. Первый вариант Турбо Паскаля фирма Borland выпустила в середине 1980-х гг. К сегодняшнему дню этой фирмой созданы шесть модификаций системы, известных как версии 3.0, 4.0, 5.0, 5.5, 6.0, 7.0. Каждая из них представляет собой усовершенствование предыдущей версии. Все они создавались для семейства машин IBM PC и совершенствовались вместе с компьютерами.

Версия 3.0 ориентирована на ПК малой мощности (IBM PC/XT). Разрабатываемые на ней программы имеют ограничение на длину (не более 64 Кбайт); в этой версии нет средств раздельной компиляции взаимосвязанных программ; операционная среда весьма несовершенна.

Большие изменения были внесены в версию 4.0. Появились современная диалоговая среда, средства раздельной компиляции программных модулей, мощная графическая библиотека.

Версия 5.0 отличается в основном дальнейшими усовершенствованиями среды, к которой добавлен встроенный отладчик. В версию 5.5 были впервые включены средства поддержки объектно-ориентированного программирования — современной технологии создания программ.

Главные отличия версии 6.0: новая среда, ориентированная на работу с устройством ввода — мышью и использующая много-

оконный режим работы; объектно-ориентированная библиотека Turbo-Vision, а также возможность включать в текст программы команды Ассемблера.

Версия 7.0 не содержит каких-то принципиальных новшеств по сравнению с 6.0. Введены некоторые расширения языка программирования, а также дополнительные сервисные возможности системной оболочки.

Программа на Турбо Паскале проходит три этапа обработки:

- создание текста программы;
- компиляция;
- исполнение откомпилированной программы.

В соответствии с этими функциями Турбо-система включает в себя три главные компоненты:

- редактор текстов;
- компилятор;
- исполнительную систему.

С помощью встроенного в систему текстового редактора можно формировать в памяти любые тексты, не только программы на Паскале. В частности, это могут быть исходные данные решаемой задачи в текстовой форме. Текст программы, созданный редактором, можно сохранить на диске в виде файла с именем следующего формата

```
<имя файла>.pas
```

где `pas` — это стандартное расширение имени файла, созданного системным редактором. Имя файла задается пользователем.

Обращение к текстовому редактору происходит по команде **Edit**.

Компилятор переводит программу с языка Паскаль на язык машинных команд. При этом проверяется соответствие программы правилам языка программирования (синтаксический и семантический контроль). При обнаружении ошибки компьютер выдает сообщение о ней пользователю и прекращает работу. Программа, полученная в результате компиляции, может быть сохранена на диске в файле с именем

```
<имя файла>.exe
```

Работа компилятора инициируется системной командой **Compile**.

Исполнение откомпилированной программы производится по команде **Run**. При этом исполнение программы остается под контролем Турбо-системы. В частности, Турбо-система помогает обнаружить ошибку в программе, если при исполнении произошел сбой. Пользователю сообщается причина сбоя и указывается место, где он случился в Паскаль-программе. Происходит автоматический возврат в режим редактирования.

В старших версиях Турбо Паскаля имеется система отладки (**Debug**). С ее помощью можно просмотреть на экране значение любой пере-

менной, найти значение любого выражения, установить новое значение переменной. Можно прервать выполнение программы в указанных местах, которые называются контрольными точками. Система отладки существенно облегчает программисту поиск ошибок.

Подробные сведения о работе с конкретными версиями Турбо Паскаля можно найти в специальной литературе (см. список литературы).

3.3. Элементы языка Турбо Паскаль

Алфавит. Алфавит языка состоит из множества символов, включающих в себя *буквы*, *цифры* и *специальные символы*.

Латинские буквы: от *A* до *Z* (прописные) и от *a* до *z* (строчные).

Цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Шестнадцатеричные цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Специальные символы: + — * / = < > [ ] . , ( ) : ; { } ^ @ $ #.

Следующие комбинации специальных символов являются едиными символами (*их нельзя разделять пробелами*):

:=	знак присваивания;	<=	меньше или равно;
>=	больше или равно;	(* *)	ограничители комментариев (используются наряду с { });
<>	не равно;	(. .)	эквивалент [ ].

Пробелы — символ пробела (ASCII-32) и все управляющие символы кода ASCII (от 0 до 31).

К спецсимволам относятся *служебные слова*, смысл которых определен однозначно. Служебные слова не могут быть использованы для других целей. С точки зрения языка это единые символы. Вот список служебных слов Турбо Паскаля:

absolute	**end**	**inline**	**procedure**	**type**
and	**external**	**interface**	**program**	**unit**
array	**file**	**interrupt**	**record**	**until**
begin	**for**	**label**	**repeat**	**uses**
case	**forward**	**mod**	**set**	**var**
const	**function**	**nil**	**shl**	**while**
div	**goto**	**not**	**shr**	**with**
do	**if**	**of**	**string**	**xor**
downto	**implementation**	**or**	**then**	
else	**in**	**packed**	**to**	

Последние версии языка содержат еще ряд служебных слов, относящихся к работе с объектами и встроенным ассемблером.

Идентификаторы. Идентификатором называется символическое имя определенного программного объекта. Такими объектами яв-

ляются имена констант, переменных, типов данных, процедур и функций, программ. С помощью синтаксической диаграммы идентификатор можно определить, как показано на рис. 8.

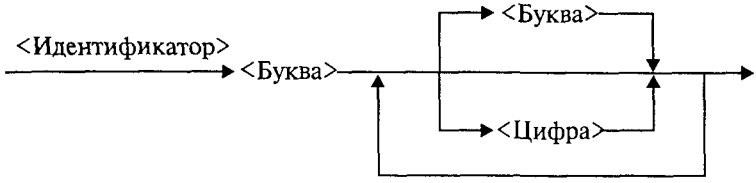


Рис. 8

Расшифровать это можно так: идентификатор — это любая последовательность букв и цифр, начинающаяся с буквы. В Турбо Паскале к буквам приравнивается также знак подчеркивания. Строчные и прописные буквы в идентификаторах и служебных словах не различаются. Например: `max`, `MAX`, `MaX` и `mAx` — одно и то же имя.

Длина идентификатора может быть произвольной, но значащими являются только первые 63 символа.

Комментарии. Следующие конструкции представляют собой комментарии и поэтому игнорируются компилятором:

```
{любой текст, не содержащий символ «}» }
(* любой текст, не содержащий символы «*)»*)
```

Буквы русского алфавита употребляются только в комментариях, в литерных и текстовых константах.

Строка, начинающаяся с символов {$ или (*$, является *директивой компилятора*. За этими символами следует мнемоника команды компилятора.

3.4. Типы данных

Концепция типов данных является одной из центральных в любом языке программирования. *С типом величины связаны три ее свойства: форма внутреннего представления, множество принимаемых значений и множество допустимых операций.* Турбо Паскаль характеризуется большим разнообразием типов данных, отраженном на рис. 9.

В стандартном Паскале отсутствует строковый тип. Кроме того, в Турбо Паскале целые и вещественные — это группы типов. В старших версиях Турбо Паскаля существует процедурный тип и тип объект.

Каждый тип имеет свой идентификатор.

В табл. 3.1 представлена информация о простых типах данных, определенных в Турбо Паскале. Для вещественных типов в скобках указано количество сохраняемых значащих цифр мантиссы в десятичном представлении числа.

Рис. 9

Таблица 3.1

Идентификатор	Длина, байт	Диапазон (множество) значений
Целые типы		
`Integer`	2	−32768 ... 32767
`Byte`	1	0...255
`Word`	2	0 ... 65535
`Shortint`	1	−128 ... 127
`Longint`	4	−2147483648 ... 2147483647
Вещественные типы		
`Real`	6	$2{,}9\cdot10^{-39}$ ... $1{,}7\cdot10^{38}$ (11 — 12)
`Single`	4	$1{,}5\cdot10^{-45}$ ... $3{,}4\cdot10^{38}$ (7 — 8)
`Double`	8	$5\cdot10^{-324}$ ... $1{,}7\cdot10^{308}$ (15 —16)
`Extended`	10	$3{,}4\cdot10^{-4932}$ ... $1{,}1\cdot10^{4932}$ (19 — 20)
Логический тип		
`Boolean`	1	true, false
Символьный тип		
`Char`	1	все символы кода ASCII

В стандарте Паскаля из вещественных типов определен только тип `Real`; из целых типов — `Integer`.

Типы `Single`, `Double`, `Extended` употребляются в Паскаль-программах только в том случае, если ПК снабжен сопроцессором «плавающей арифметики» (для процессоров IBM PC, начиная с Intel-80486 и старше, это условие всегда выполняется).

Тип данных называется *порядковым*, если он состоит из счетного количества значений, которые можно пронумеровать. Отсюда следует, что на этом множестве значений существуют понятия «следующий» и «предыдущий».

Описание переменных. Для всех переменных величин, используемых в программе, должны быть указаны их типы. Это делается в *разделе переменных* программы. Структура раздела переменных показана на рис. 10.

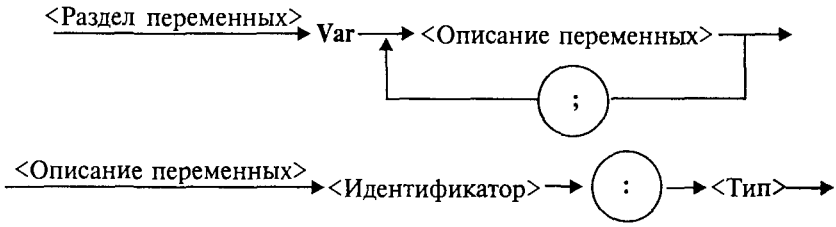


Рис. 10

Пример раздела переменных программы:

```
Var m,n,k: Integer;
    x,y,z: Real;
    Symbol: Char;
```

Константы. Тип константы определяется по контексту, т.е. по форме ее записи в программе.

Целые десятичные константы записываются в обычной форме целого числа со знаком или без знака, например 25, –24712, 376.

Целые шестнадцатеричные константы записываются с префиксом $. Они должны находиться в диапазоне от $00000000 до $FFFFFFFF.

Вещественные константы *с фиксированной точкой* записываются в обычной форме десятичного числа с дробной частью. Разделитель целой и дробной части — точка, например: 56.346, 0.000055, –345678.0.

Вещественные константы *с плавающей точкой* имеют форму:

<мантисса>E<порядок>

Здесь мантисса — целое или вещественное число с фиксированной точкой, порядок — целое число со знаком или без, например 7E–2 ($7\cdot10^{-2}$), 12.25E6 ($12{,}25\cdot10^{6}$), 1E–25 (10^{-25}).

Символьная константа — любой символ алфавита, заключенный в апострофы, например, 'W', '!', '9'.

Логическая константа — одно из двух слов: *true, false.*

Строковая константа — строка символов, заключенная в апострофы, например 'Turbo Pascal', 'Ответ:', '35-45-79'. Максимальная длина — 255 символов.

Константе может быть поставлено в соответствие определенное имя. Назначение имени константе производится в *разделе констант* программы. Структура раздела констант показана на рис. 11.

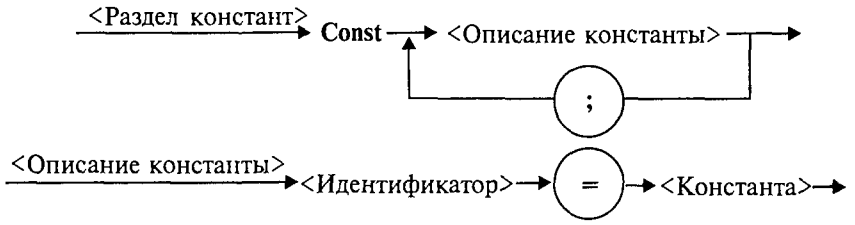


Рис. 11

Пример:

```
Const
     Max=1000;
     G=9.81;
     Cod='ошибка';
```

В дополнение к сказанному заметим, что в Турбо Паскале допустимо употребление *типизированных* констант. Типизированная константа аналогична переменной, которой задается начальное значение. Причем происходит это на этапе компиляции. Описание типизированной константы приведено на рис. 12.

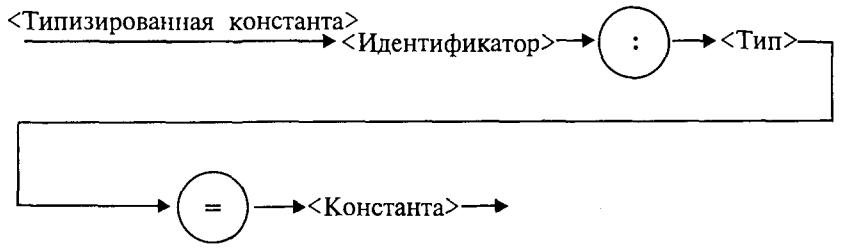


Рис. 12

Пример:

```
Const NumberCard:Integer=1267;
      Size:Real=12.67;
      Symbol:Char = '*';
```

В Турбо Паскале имеется ряд имен, зарезервированных за определенными значениями констант. Ими можно пользоваться без предварительного определения в программе (табл. 3.2).

Таблица 3.2

Идентификатор	Тип	Значение
True	boolean	истина
False	boolean	ложь
MaxInt	integer	32767

Типы пользователя. Один из принципиальных моментов состоит в том, что пользователю разрешается определять свои типы данных. Типы пользователя всегда базируются на стандартных типах данных Паскаля.

Для описания типов пользователя в Паскале существует *раздел типов*, структура которого представлена на рис. 13.

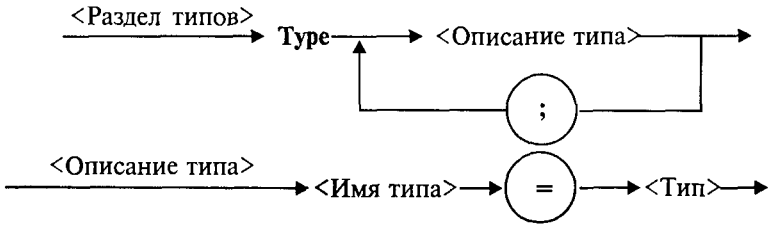


Рис. 13

Перечисляемый тип (рис. 14) задается непосредственно перечислением всех значений, которые может принимать переменная данного типа.

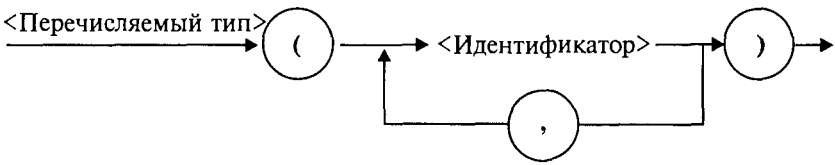


Рис. 14

Определенное имя типа затем используется для описания переменных. Например:

```
Type Gaz=(C,O,N,F);
        Metal=(Fe,Co,Na,Cu,Zn);
Var G1,G2,G3:Gaz;
```

```
Met1,Met2:Metall;
Day:(Sun,Mon,Tue,Wed,Thu,Fri,Sat)
```

Здесь `Gaz` и `Metal` — имена перечисляемых типов, которые ставятся в соответствие переменным `G1`, `G2`, `G3` и `Met1`, `Met2`. Переменной `Day` назначается перечисляемый тип, которому не присвоено имя.

Значения, входящие в перечисляемый тип, являются *константами*. Действия над ними подчиняются правилам, применимым к константам. Каждое значение в перечисляемом типе занимает в памяти 2 байта. Поэтому число элементов не должно превышать 65535.

Перечисляемый тип — упорядоченное множество. Его элементы пронумерованы начиная от 0 в порядке следования в описании.

В программе, в которой присутствует данное выше описание, возможен такой фрагмент:

```
if Day=Sun then WriteLn('Ура!Сегодня выходной!');
```

Интервальный тип (рис. 15) задается как упорядоченное ограниченное подмножество некоторого порядкового типа.

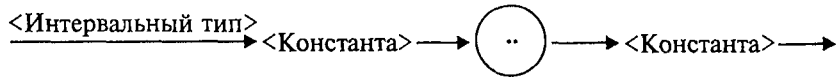


Рис. 15

Порядковый номер первой константы не должен превышать номера второй константы в соответствующем базовом типе.

При исполнении программы автоматически контролируется принадлежность значений переменной интервального типа установленному диапазону. При выходе из диапазона исполнение программы прерывается.

Пример:

```
Type Numbers=1..31;
        Alf='A'..'Z';
Var  Data:Numbers;
        Bukva:Alf;
```

3.5. Арифметические операции, функции, выражения. Арифметический оператор присваивания

К арифметическим типам данных относятся группы вещественных и целых типов. К ним применимы арифметические операции и операции отношений.

Операции над данными бывают *унарными* (применимые к одному операнду) и *бинарными* (применимые к двум операндам). Унарная арифметическая операция одна. Это операция изменения знака. Ее формат:

```
-<величина>
```

Бинарные арифметические операции стандартного Паскаля описаны в табл. 3.3. В ней *I* обозначает целые типы, *R* — вещественные типы.

Таблица 3.3

Знак	Выражение	Типы операндов	Тип результатов	Операция
+	A + B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Сложение
-	A - B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Вычитание
*	A * B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Умножение
/	A / B	*R, R* *I, I* *I, R; R, I*	*R* *R* *R*	Вещественное деление
div	A div B	*I, I*	*I*	Целое деление
mod	A mod B	*I, I*	*I*	Остаток от целого деления

К арифметическим величинам могут быть применены стандартные функции Паскаля. Структура обращения к функции представлена на рис. 16.

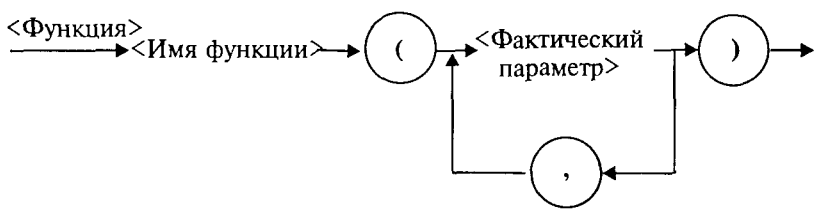


Рис. 16

Функция выступает как операнд в выражении. Например, в следующем операторе присваивания

```
X:=2*Sin(A)/Ln(3.5)+Cos(C-D)
```

операндами являются три функции: sin, ln, cos. Их запись такая же, как в математике. Аргументы называются фактическими параметрами и являются в общем случае выражениями арифметического типа. Аргументы записываются в круглых скобках. Результат вычисления функции — величина соответствующего типа.

Табл. 3.4 содержит описания математических стандартных функций Турбо Паскаля.

Таблица 3.4

Обращение	Тип аргумента	Тип результата	Функция
`Pi`	—	*R*	Число π = 3.1415926536E+00
`Abs(x)`	*I, R*	*I, R*	Модуль аргумента *x*
`Arctan(x)`	*I, R*	*R*	Арктангенс *x* (радианы)
`Cos(x)`	*I, R*	*R*	Косинус *x* (*x* в радианах)
`Exp(x)`	*I, R*	*R*	e^x — экспонента
`Frac(x)`	*I, R*	*R*	Дробная часть *x*
`Int(x)`	*I, R*	*R*	Целая часть *x*
`Ln(x)`	*I, R*	*R*	Натуральный логарифм *x*
`Random`		*R*	Псевдослучайное число в интервале [0, 1)
`Random(x)`	*I*	*I*	Псевдослучайное число в интервале [0, *x*)
`Round(x)`	*R*	*I*	Округление до ближайшего целого
`Sin(x)`	*I, R*	*R*	Синус *x* (*x* в радианах)
`Sqr(x)`	*I, R*	*I, R*	Квадрат *x*
`Sqrt(x)`	*I, R*	*R*	Корень квадратный из *x*
`Trunc(x)`	*R*	*I*	Ближайшее целое, не превышающее *x* по модулю

Арифметическое выражение задает порядок выполнения действий над числовыми величинами. Арифметические выражения содержат арифметические операции, функции, операнды, круглые скобки. Одна константа или одна переменная — простейшая форма арифметического выражения.

Например, запишем по правилам Паскаля следующее математическое выражение:

$$\frac{2a+\sqrt{0,5\sin(x+y)}}{0,2c-\ln(x-y)}.$$

На Паскале это выглядит так:

```
(2*a+Sqrt(0.5*Sin(x+y)))/(0.2*C-Ln(x-y))
```

Для того чтобы правильно записывать арифметические выражения, нужно соблюдать следующие правила:

1. Все символы пишутся в строчку на одном уровне. Проставляются все знаки операций (нельзя пропускать знак умножения).

2. Не допускаются два следующих подряд знака операций (нельзя **A+-B**; можно **A+(-B)**).

3. Операции с более высоким приоритетом выполняются раньше операций с меньшим приоритетом. Порядок убывания приоритетов:

- вычисление функции;
- унарная операция смены знака (–);
- *, /, div, mod;
- +, –.

4. Несколько записанных подряд операций одинакового приоритета выполняются последовательно слева направо.

5. Часть выражения, заключенная в скобки, вычисляется в первую очередь. (Например, **(A+B)*(C-D)** — умножение производится после сложения и вычитания.)

Не следует записывать выражений, не имеющих математического смысла. Например, деление на нуль, логарифм отрицательного числа и т.п.

Пример. Цифрами сверху указан порядок выполнения операций:

```
   1  7  4  5   3      6   2     12   11 10    8     9
(1+y)*(2*x+sqrt(y)-(x+y))/(y+1/(sqr(x)-4))
```

Данное арифметическое выражение соответствует следующей математической формуле:

$$(1+y)\frac{2x+\sqrt{y}-(x+y)}{y+\dfrac{1}{x^2-4}}.$$

В Паскале нет операции или стандартной функции возведения числа в произвольную степень. Для вычисления x^y рекомендуется поступать следующим образом:

• если y — целое значение, то степень вычисляется через умножение; например, $x^3 \to x \cdot x \cdot x$; большие степени следует вычислять умножением в цикле;

• если y — вещественное значение, то используется следующая математическая формула: $x^y = e^{y\ln(x)}$.

На Паскале это будет выглядеть так:

```
Exp(y*Ln(x))
```

Очевидно, что при вещественном y не допускается нулевое или отрицательное значение x. Для целого y такого ограничения нет.

Например,

$$\sqrt[3]{a+1} = (a+1)^{\frac{1}{3}}.$$

На Паскале это будет так:

```
Exp(1/3*Ln(a+1))
```

Выражение имеет целый тип, если в результате его вычисления получается величина целого типа. Выражение имеет вещественный тип, если результатом его вычисления является вещественная величина.

Арифметический оператор присваивания имеет структуру, представленную на рис. 17.

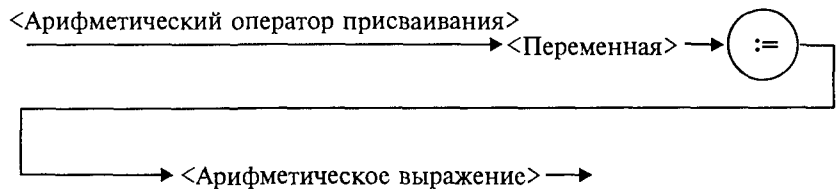


Рис. 17

Например:

```
Y:=(F*N-4.5)/Cos(X)
```

Порядок выполнения оператора присваивания нами уже рассматривался. Следует обратить особое внимание на следующее правило: типы переменной и выражения должны быть одинаковыми. Исключение составляет случай, когда выражение имеет целый тип, а переменная — вещественный.

Упражнения

1. Для следующих формул записать соответствующие арифметические выражения на Паскале:

а) $a + bx + cyz$; б) $[(ax - b)x + c]x - d$;

в) $\frac{a+b}{c}+\frac{c}{ab}$;

г) $\frac{x+y}{a_1}\cdot\frac{a_2}{x-y}$;

д) $10^4\alpha-3\frac{1}{5}\beta$;

е) $\left(1+\frac{x}{2!}+\frac{y}{3!}\right)\Big/\left(1+\frac{2}{3+xy}\right)$.

2. Записать математические формулы, соответствующие следующим выражениям на Паскале:

а) `(p+q)/(r+s)-p*q/(r*s);`
б) `1E3+beta/(x-gamma*delta);`
в) `a/b*(c+d)-(a-b)/b/c+1E-8.`

3. Почему в Паскале аргумент функции всегда записывают в скобках (например, пишут ln(5), а не ln5)?

4. Для следующих формул записать соответствующие арифметические выражения на Паскале:

а) $(1+x)^2$; б) $\sqrt{1+x^2}$; в) $\cos^2 x^2$; г) $\log_2\frac{x}{5}$;

д) $\arcsin x$; е) $\frac{e^x+e^{-x}}{2}$; ж) $x^{\sqrt{2}}$; з) $\sqrt[3]{1+x}$;

и) $\sqrt{x^8+8^x}$; к) $\frac{xyz-3{,}3\left|x+\sqrt[4]{y}\right|}{10^7+\ln 4!}$; л) $\frac{\beta+\sin^2\pi^4}{\cos 2+\left|\operatorname{ctg}\gamma\right|}$.

5. Вычислить значения выражений:

а) `trunc(6.9);`
б) `trunc(6.2);`
в) `20 div 6;`
г) `2 div 5;`
д) `round(6.9);`
е) `round(6.2);`
ж) `20 mod 6;`
з) `2 mod 5;`
и) `3*7 div 2 mod 7/3-trunc(sin(1)).`

6. Определить тип выражения:

а) `1+0.0;` б) `20/4;` в) `sqr(4);`
г) `sqrt(16);` д) `sin(0);` е) `trunc(-3.14).`

7. Если y — вещественная переменная, а n — целая, то какие из следующих операторов присваивания правильные, а какие нет:

а) `y:=n+1;`
б) `n:=y-1;`
в) `n:=4.0;`
г) `y:=trunc(y);`
д) `y:=n div 2;`
е) `y:=y div 2;`
ж) `n:=n/2;`
з) `n:=sqr(sqrt(n))`?

8. Поменять местами значения целых переменных x и y, не используя дополнительные переменные. Найдя такой алгоритм, оп-

ределить, в чем его недостаток по сравнению с методом обмена через третью переменную. Можно ли его применять для вещественных чисел?

9. Присвоить целой переменной h значение цифры, стоящей в разряде сотен в записи положительного целого числа k (например, если $k = 28796$, то $h = 7$).

10. Целой переменной S присвоить значение суммы цифр трехзначного целого числа k.

11. Какую задачу решает следующая программа?

```
Program Test;
Type Natur=1..MaxInt;
Var N: Natur;
    X: Real;
Begin ReadLn(N);
    X:=0;
      While N>0 Do
      Begin
         X:=X+1.0;
         N:=N-1
      End;
    WriteLn(X)
End.
```

Можно ли того же самого результата достичь более простым способом?

3.6. Ввод с клавиатуры и вывод на экран

Ввод данных — это передача информации от внешних устройств в оперативную память. Вводятся, как правило, исходные данные решаемой задачи. *Вывод* — обратный процесс, когда данные передаются из оперативной памяти на внешние носители (принтер, дисплей, магнитные устройства и т.д.). Результаты решения всякой задачи должны быть выведены на один из этих носителей.

Основными устройствами ввода-вывода у персонального компьютера являются клавиатура и дисплей (экран монитора). Именно через эти устройства главным образом осуществляется диалог между человеком и ПК.

Процедура ввода *с клавиатуры* имеет следующий формат:

```
Read(<список ввода>)
```

где `<список ввода>` — это последовательность имен переменных, разделенных запятыми. Слово *read* переводится как *читать*. (Точнее говоря, `Read` — это оператор обращения к стандартной процедуре ввода.)

Например,

```
Read(a,b,c,d)
```

При выполнении этого оператора происходит прерывание работы компьютера, после чего пользователь должен набрать на клавиатуре значения переменных *a*, *b*, *c*, *d*, отделяя их друг от друга пробелами. При этом вводимые значения высвечиваются на экране. В конце нажимают клавишу Enter. Значения должны вводиться в строгом соответствии с синтаксисом Паскаля.

Пример:

```
Var T: Real;
    J: Integer;
    K: Char;
Begin
    Read(T,J,K);
```

Набираем на клавиатуре:

```
253.98   100   G (Enter)
```

Если в программе имеется несколько операторов `Read`, то данные для них вводятся потоком, т.е. после считывания значений переменных для одного оператора `Read` данные для следующего оператора читаются из той же строки на экране, что и для предыдущего до окончания строки, затем происходит переход на следующую строку.

Пример:

```
Var A,B: Integer;
    C,D: Real;
Begin
    Read(A,B);
    Read(C,D);
```

Набираем на клавиатуре:

```
18758 34(Enter) 2.62E-02 1.54E+01(Enter)
```

Другой вариант оператора ввода с клавиатуры имеет вид:

```
ReadLn(<список ввода>)
```

Здесь слово `ReadLn` означает *read line* — читать строку. Этот оператор отличается от `Read` только тем, что после считывания последнего в списке значения для одного оператора `ReadLn` данные для следующего оператора будут считываться с начала новой строки. Если в предыдущем примере заменить операторы `Read` на `ReadLn`:

```
ReadLn(A,B);
ReadLn(C,D);
```

то ввод значений будет происходить из двух строк:

```
18758 34 (Enter)
2.62E-02 1.54E+01 (Enter)
```

Оператор *вывода на экран* (обращение к стандартной процедуре вывода) имеет следующий формат:

```
Write(<список вывода>)
```

Здесь элементами списка вывода могут быть выражения различных типов (в частности, константы и переменные).

Пример:

```
Write(234);    {выводится целая константа}
Write(A+B-2); {выводится результат вычисления
               выражения}
Write(X,Summa,Arg1,Arg2); {выводятся значения
                           переменных}
```

При выводе на экран нескольких чисел в строку они не отделяются друг от друга пробелами. Программист сам должен позаботиться о таком разделении. Пусть, например, $I = 1$; $J = 2$; $K = 3$. Тогда, написав в программе

```
Write(I,'',J,'',K);
```

получим на экране строку: `1 2 3`. После вывода последнего символа курсор остается в той же строке. Следующий вывод на экран будет начинаться с этой позиции курсора.

Второй вариант процедуры вывода на экран:

```
WriteLn(<список вывода>)
```

Слово `WriteLn` — *write line* — означает *писать строку*. Его действие отличается от оператора `Write` тем, что после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор `WriteLn`, записанный без параметров, вызывает перевод строки.

Форматы вывода. В списке вывода могут присутствовать указатели форматов вывода (форматы). Формат определяет представление выводимого значения на экране. Он отделяется от соответствующего ему элемента двоеточием. Если указатель формата отсутствует, то машина выводит значение по определенному правилу, предусмотренному по умолчанию.

Ниже кратко, в справочной форме, приводятся правила и примеры бесформатного и форматированного вывода величин различных типов. Для представления списка вывода здесь будут использованы следующие обозначения:

I, P, Q — целочисленные выражения;

R — выражение вещественного типа;

B — выражение булевского типа;
Ch — символьная величина;
S — строковое выражение;
— цифра;
* — знак «+» или «—»;
_ — пробел.

Форматы процедуры `Write`

I — выводится десятичное представление величины *I*, начиная с позиции расположения курсора:

Значение *I*	Оператор	Результат
`134`	`Write(I)`	`134`
`287`	`Write(I,I,I)`	`287287287`

I:*P* — выводится десятичное представление величины *I* в крайние правые позиции поля шириной *P*:

Значение *I*	Оператор	Результат
`134`	`Write(I:6)`	`___134`
`312`	`Write((I+I):7)`	`____624`

R — в поле шириной 18 символов выводится десятичное представление величины *R* в формате с плавающей точкой. Если $R \geq 0{,}0$, используется формат `_#.##########E*##`. Если $R < 0{,}0$, то формат имеет вид `_-#.##########E*##`:

Значение *R*	Оператор	Результат
`715.432`	`Write(R)`	`_7.1543200000E+02`
`-1.919E+01`	`Write(R)`	`_-1.9190000000E+01`

R:*P* — в крайние правые позиции поля шириной *P* символов выводится десятичное представление значения *R* в нормализованном формате с плавающей точкой. Минимальная длина поля вывода для положительных чисел составляет 7 символов, для отрицательных — 8 символов. После точки выводится по крайней мере одна цифра:

Значение *R*	Оператор	Результат
`511.04`	`Write(R:15)`	`5.110400000E+02`
`46.78`	`Write(-R:12)`	`-4.67800E+01`

R:*P*:*Q* — в крайние правые позиции поля шириной *P* символов выводится десятичное представление значения *R* в формате с фиксированной точкой, причем после десятичной точки выводится *Q* цифр ($0 \leq Q \leq 24$), представляющих дробную часть числа. Если $Q = 0$, то ни дробная часть, ни десятичная точка не выводятся. Если $Q > 24$, то при выводе используется формат с плавающей точкой:

Значение *R*	Оператор	Результат
`511.04`	`Write(R:8:4)`	`511.0400`
`-46.78`	`Write(R:7:2)`	`_-46.78`

Ch:*P* — в крайнюю правую позицию поля шириной *P* выводится значение *Ch*:

Значение *Ch*	Оператор	Результат
`'X'`	`Write(Ch:3)`	`__X`
`'!'`	`Write(Ch:2,Ch:4)`	`_!___!`

S — начиная с позиции курсора выводится значение *S*:

Значение *S*	Оператор	Результат
`'Day N'`	`Write(S)`	`Day N`
`'RRDD'`	`Write(S,S)`	`RRDDRRDD`

S:*P* — значение *S* выводится в крайние правые позиции поля шириной *P* символов:

Значение *S*	Оператор	Результат
`'Day N'`	`Write(S:10)`	`_____Day N`
`'RRDD'`	`Write(S:5,S:5)`	`_RRDD_RRDD`

B — выводится результат выражения *B*: *true* или *false*, начиная с текущей позиции курсора:

Значение *B*	Оператор	Результат
`True`	`Write(B)`	`True`
`False`	`Write(B,Not B)`	`FalseTrue`

B:*P* — *в крайние правые позиции поля шириной P символов* выводится результат булевского выражения:

Значение *B*	Оператор	Результат
`True`	`Write(B:6)`	`--True`
`False`	`Write(B:6,Not B:7)`	`_False___True`

3.7. Управление символьным выводом на экран

Использование для вывода на экран только процедур `Write` и `WriteLn` дает программисту очень слабые возможности для управления расположением на экране выводимого текста. Печать текста может производиться только сверху вниз, слева направо. Невозможны возврат к предыдущим строкам, стирание напечатанного текста, изменение цвета символов и т.д.

Дополнительные возможности управления выводом на экран дают процедуры и функции модуля CRT.

Для установления связи пользовательской программы с модулем перед разделами описаний должна быть поставлена строка

```
Uses CRT
```

Для работы с модулем CRT необходимо познакомиться со следующими понятиями: режимы экрана, координаты на экране, текстовое окно, цвет фона и цвет символа.

Режимы экрана. Вывод на экран может происходить в текстовом или графическом виде (на графических дисплеях). Мы здесь будем говорить только о текстовом выводе.

Дисплеи бывают монохроматические (черно-белые) и цветные. Монохроматические дисплеи могут работать только в черно-белом режиме; цветные — как в черно-белом, так и в цветном. Кроме того, текстовые режимы различаются по количеству символьных строк и столбцов, умещающихся на экране.

В модуле CRT каждый режим имеет определенный номер, за которым закреплено символическое имя (описанная константа). Для установки режима экрана используется процедура

```
TextMode(<номер режима>)
```

При обращении к процедуре номер режима может задаваться как числом, так и именем соответствующей константы. Например, два оператора

```
TextMode(1);
TextMode(CO40);
```

эквивалентны.

Как правило, исходный режим экрана, устанавливаемый по умолчанию, — CO80 (на цветных дисплеях).

Координаты позиции. Каждая символьная позиция на текстовом экране определена двумя координатами (X, Y). Координата X — позиция в строке. Для крайней левой позиции в строке $X = 1$. Координата Y — номер строки, в которой находится символ. Строки нумеруются сверху вниз.

Например, в режиме 80×25 символ в верхнем левом углу имеет координаты (1; 1); символ в нижнем правом углу — (80; 25); символ в середине экрана — (40; 13).

Для установления курсора на экране в позицию с координатами (X, Y) в модуле CRT существует процедура:

```
GoToXY(X,Y)
```

Здесь координаты курсора задаются выражениями типа `Byte`.

Вот пример программы, которая очищает экран и выставляет в центре экрана символ *:

```
Uses CRT;
Begin
```

```
  ClrScr;
  GoToXY(40,13);
  Write('*')
End.
```

Используемая здесь процедура `ClrScr` производит очистку экрана.

Текстовое окно. Прямоугольное пространство на экране, в которое производится вывод символов, называется текстовым окном. Положение окна определяется координатами верхнего левого угла и нижнего правого угла прямоугольника. Если окно занимает весь экран, то в режиме 80×25 его координаты (1; 1) — (80; 25). Таким является исходное окно. Изменить положение и размер текстового окна можно с помощью процедуры

```
Window(X1,Y1,X2,Y2)
```

Здесь аргументы — величины типа `Byte`; `(X1,Y1)` — координаты верхнего левого угла, `(X2,Y2)` — координаты правого нижнего угла окна. После определения окна попытки вывода символов за его пределы оказываются безрезультатными. Повторное обращение к процедуре `Window` с новыми параметрами отменяет предыдущее назначение.

Управление цветом. На современных цветных дисплеях типа EGA, VGA, SVGA в текстовом режиме экрана можно использовать 16 цветов.

В модуле CRT объявлены константы, имена которых представляют собой английские названия цветов, а соответствующие им значения — порядковые номера этих цветов.

Процедура назначения цвета фона:

```
TextBackGround(Color)
```

Здесь аргумент — величина типа `Byte`, задающая номер цвета.

Процедура назначения цвета символа:

```
TextColor(Color)
```

Если цвет фона назначается до очистки текстового окна, то после очистки окно заливается этим цветом. Если фон устанавливается после очистки экрана, то чистое окно будет иметь черный цвет (по умолчанию), а назначенный цвет фона будет устанавливаться в тех позициях, в которые выводятся символы.

Вот пример программы, в которой по очереди откроются четыре окна, и каждое из них будет залито своим фоновым цветом:

```
Uses CRT;
Begin
  Window(1,1,40,12);
  TextBackGround(White); ClrScr;
```

```
  Window(41,1,80,12);
  TextBackGround(Red); ClrScr;
  Window(1,13,40,25);
  TextBackGround(LightRed); ClrScr;
  Window(41,13,80,25);
  TextBackGround(Green); ClrScr;
End.
```

По следующей программе на белом фоне в середине экрана будут выведены номера первых пятнадцати цветов. Каждый номер будет того цвета, который он обозначает.

```
Uses CRT;
Var I: Byte;
Begin
    TextBackGround(White);
    ClrScr;
    GoToXY(1,12);
    For I:=0 To 14 Do
    Begin
        TextColor(I);
        Write(I:5);
    End
End.
```

Кратко опишем еще несколько процедур управления текстовым экраном из модуля CRT. Все эти процедуры не имеют параметров.

Процедура `ClrEOL`. Стирает часть строки от текущей позиции курсора до конца этой строки в окне. При этом положение курсора не меняется.

Процедура `DelLine`. Уничтожает всю строку с курсором. Нижние строки сдвигаются на одну вверх.

Процедура `InsLine`. Вставляет пустую строку перед строкой, в которой стоит курсор.

Процедуры `LowVideo, NormVideo, HighVideo`. Устанавливают режимы пониженной, нормальной и повышенной яркости символов соответственно.

Весьма полезной является функция `KeyPressed` из модуля CRT. При исполнении этой функции происходит опрос клавиатуры и определяется, не нажата ли какая-нибудь клавиша. В результате функция выдает логическое значение `True`, если нажата любая клавиша, и значение `False` в противном случае. Часто эту функцию используют для организации задержки окна результатов на экране (после выполнения программы Турбо Паскаль вызывает на экран окно редактора). Перед концом программы записывается следующий оператор:

```
Repeat Until KeyPressed;
```

Это пустой цикл, который «крутится на месте» до нажатия какой-либо клавиши. В это время на экране окно результатов. После нажатия на клавишу значение KeyPressed станет равно True, цикл завершится, будет выполнен переход на метку **End** и на экран вернется окно редактора. Этот прием можно использовать для задержки выполнения программы в любом ее месте.

В приведенную выше программу получения на экране четырех разноцветных окон внесем следующее дополнение: после установки четырехцветного экрана выполнение программы останавливается и изображение сохраняется; затем после нажатия на любую клавишу экран возвращается в исходный режим (80×25, черный фон, белые символы). Для этого перед концом программы нужно добавить следующее:

```
Repeat Until KeyPressed;
Window(1,1,80,25);
TextBackGround(Black);
ClrScr;
```

О других процедурах и функциях модуля CRT читайте в книгах по Турбо Паскалю.

Упражнения

1. Что будет напечатано в результате работы программы

```
Program Roots;
Var B,C,D: Real;
Begin
    Read(B,C);
    D:=Sqrt(Sqr(B)-4*C);
    WriteLn('x1=',(-B+D)/2,
            'x2=',(-B-D)/2)
End.
```

если в качестве исходных данных заданы числа 10 и -20?

2. Что будет напечатано в результате работы программы

```
Program Less;
Var X:Real; T:Boolean;
Begin
   Read(X);
   T:=X<Round(X);
   Read(X);
   T:=T And (X<Trunc(X));
   WriteLn(T)
End.
```

если последовательно вводятся два значения: 34, 79?

3. Что будет напечатано в результате работы программы

```
Program ABC;
Var A,B: Integer;
Begin
   Read(A,B,A);
   WriteLn(A,B:2,A:5)
End.
```

если последовательно вводятся три числа: 36, −6, 2345?

4. Составьте программу вычисления суммы двух целых чисел, которая будет вести диалог с пользователем в следующем виде (на месте многоточий — вводимые и выводимые числа):

```
ВВЕДИТЕ ДВА СЛАГАЕМЫХ
a=...........
b=...........
РЕЗУЛЬТАТ ВЫЧИСЛЕНИЙ:
a+b=...........
```

3.8. Логические величины, операции, выражения. Логический оператор присваивания

Прямое отношение к программированию имеет дисциплина, которая называется *математической логикой*. Основу математической логики составляет *алгебра логики*, или исчисление высказываний. Под высказыванием понимается любое утверждение, в отношении которого можно однозначно сказать, истинно оно или ложно. Например, «Луна — спутник Земли» — истинно; «5 > 3» — истинно; «Москва — столица Китая» — ложно; «1 = 0» — ложно. Истина или ложь являются логическими величинами. Логические значения приведенных выше высказываний однозначно определены; другими словами, их значения являются *логическими константами*.

Логическое значение неравенства $x < 0$, где x — переменная, является переменной величиной. В зависимости от значения x оно может быть либо истиной, либо ложью. В связи с этим возникает понятие логической переменной.

Основы формального аппарата математической логики создал в середине XIX в. английский математик Джордж Буль. В его честь исчисление высказываний называют булевой алгеброй, а логические величины — булевскими.

Одиночные высказывания могут быть объединены в составные логические формулы с помощью логических операций.

Имеются три основные логические операции: *отрицание*, *конъюнкция* (логическое умножение) и *дизъюнкция* (логическое сложение).

Операция отрицания обозначается в математической логике значком ¬ и читается как частица *не*. Это одноместная операция.

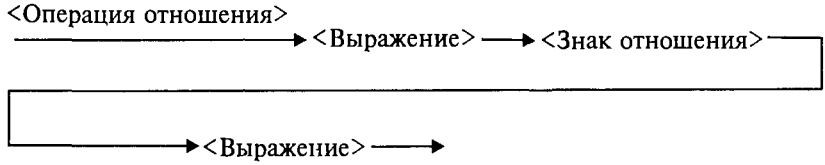
Например, $\neg(x = y)$ читается «не (x равно y)». В результате получится истина, если x не равно y, и ложь, если x равно y. Отрицание изменяет значение логической величины на противоположное.

Операция конъюнкции обозначается значком & и читается как частица *и*. Это двухместная операция. Например, $(x > 0) \& (x < 1)$ читается «x больше 0 и x меньше 1». Данная логическая формула примет значение истина, если $x \in (0, 1)$, и ложь — в противном случае. Следовательно, результат конъюнкции — истина, если истинны оба операнда. Знак операции дизъюнкции $\vee$ читается как частица *или*. Например, $(x = 0) \vee (x = 1)$ читается «x равно 0 или x равно 1». Формула дает истину, если x — двоичная цифра (0 или 1). Следовательно, дизъюнкция дает в результате истину, если хотя бы один операнд — истина.

В Паскале логические значения обозначаются служебными словами `false` (ложь) и `true` (истина), а идентификатор логического типа — `boolean`.

Кроме величин (констант и переменных) типа `boolean` логические значения `false`, `true` принимают результаты операций *отношения*.

Операции отношения (рис. 18) осуществляют сравнение двух операндов и определяют, истинно или ложно соответствующее отношение между ними.




```
<знак отношения>::=(равно)|<>(не равно)|>(боль-
ше)|<(меньше)|>=(больше или равно)|<=(меньше или
равно).
```

Рис. 18

Примеры записи отношений: `x < y; a + b >= c / d; abs(m - n) <= 1`.
Примеры вычисления значений отношений:

Отношение	Результат
`12 >= 12`	`true`
`56 > 10`	`true`
`11 <= 6`	`false`

Логические операции выполняются над операндами булева типа. Имеются четыре логические операции: `Not` — отрицание; `And` — логическое умножение (конъюнкция); `Or` — логическое сложение (дизъюнкция). Кроме этих трех обязательных операций в Турбо Паскале имеется еще операция — *исключающее ИЛИ*. Ее знак — служебное слово `Xor`. Это двухместная операция, которая в результате дает значение истина, если оба операнда имеют разные логические значения.

Операции перечислены в порядке убывания приоритетов. Результаты логических операций для различных значений операндов приведены в табл. 3.5.

Таблица 3.5

A	*B*	Not *A*	*A* And *B*	*A* Or *B*	*A* Xor *B*
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T

Операции отношения имеют самый низкий приоритет. Поэтому если операндами логической операции являются отношения, то их следует заключать в круглые скобки. Например, математическому неравенству $1 \le x \le 50$ соответствует следующее логическое выражение:

```
(1<=X) And (X<=50)
```

Логическое выражение есть логическая формула, записанная на языке программирования. Логическое выражение состоит из логических операндов, связанных логическими операциями и круглыми скобками. Результатом вычисления логического выражения является булева величина (`false` или `true`). Логическими операндами могут быть логические константы, переменные, функции, операции отношения. Один отдельный логический операнд является простейшей формой логического выражения.

Примеры логических выражений (здесь *d*, *b*, *c* — логические переменные; *x*, *y* — вещественные переменные; *k* — целая переменная):

1) `x<2*y;` 2) `true;` 3) `d;`
4) `odd(k);` 5) `not not d;` 6) `not (x>y/2);`
7) `d and (x <> y) and b;` 8) `(c or d) and (x = y) or not b.`

Если `d=true; b=false; c=true; x=3.0; y=0.5; k=5`, то результаты вычисления будут следующими:

1) `false`; 2) `true`; 3) `true`; 4) `true`;
5) `true`; 6) `false`; 7) `false`; 8) `true`.

В примере использована логическая функция `odd(k)`. Это функция от целого аргумента *k*, которая принимает значение `true`, если значение *k* нечетное, и `false`, если *k* четное.

Логический оператор присваивания имеет структуру, представленную на рис. 19.

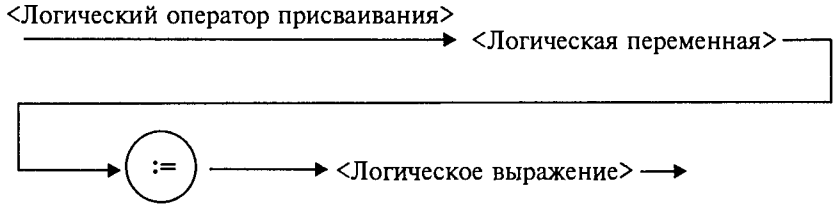


Рис. 19

Примеры логических операторов присваивания:

```
1) d:=true;
2) b:=(x>y) and (k<>0);
3) c:=d or b and not(odd(k) and d).
```

3.9. Функции, связывающие различные типы данных

В табл. 3.6 приводится список стандартных функций, обеспечивающих связь между различными типами данных.

Таблица 3.6

Обращение	Тип аргумента	Тип результата	Действие
`ord(x)`	любой порядковый	*I*	Порядковый номер значения *x* в его типе
`pred(x)`	любой порядковый	тот же, что для *x*	Предыдущее относительно *x* значение в его типе
`succ(x)`	любой порядковый	тот же, что для *x*	Следующее относительно *x* значение в его типе
`chr(x)`	`byte`	`char`	Символ с порядковым номером *x*
`odd(x)`	*I*	`boolean`	*True*, если *x* нечетное; *False*, если *x* четное

Функции `ord`, `pred` и `succ` применимы только к порядковым типам. Из простых типов это все, кроме вещественного.

Функция `ord`, применяемая к целому числу, дает его собственное значение. Например,

```
ord(-35)=-35; ord(128)=128
```

Если аргумент целый, то, например, оператор `y:=pred(x)` эквивалентен `y:=x-1`, а `y:=succ(x)` эквивалентен `y:=x+1`.

Для символьного типа аргумента эти функции дают соответственно предыдущий и следующий символ в таблице внутренней кодировки. Поскольку латинский алфавит всегда упорядочен по кодам, т.е.

```
ord('a')<ord('b')<...<ord('z'),
```

то, например,

```
pred('b')='a', а  succ('b')='c'
```

То же относится и к цифровым литерам:

```
pred('5')='4'; succ('5')='6'
```

Функция `chr(x)` является обратной к функции `ord(x)`, если `x` — символьная величина.

Это можно выразить формулой

```
chr(ord(x))=x,
```

где `x` — символьная величина.

Например, для кода ASCII справедливо

```
ord('a')=97; chr(97)='a'
```

В некоторых случаях возникает задача преобразования символьного представления числа в числовой тип. Например, нужно получить из литеры `'5'` целое число 5. Это делается так:

```
N:=ord('5')-ord('0'),
```

где `N` — целая переменная. Здесь использован тот факт, что код литеры `'5'` на пять единиц больше кода `'0'`.

Булевский тип также является порядковым. Порядок расположения двух его значений таков: `false`, `true`. Отсюда справедливы следующие отношения:

```
ord(false)=0, succ(false)=true,
ord(true)=1, pred(true)=false
```

Вот интересный пример. Пусть x, y, z — вещественные переменные. Как вы думаете, какую задачу решает следующий оператор:

```
z:=x*ord(x>=y)+y*ord(y>x)
```

Ответ такой: $z = \max(x, y)$. Как видите, эту задачу можно решить, не используя условного оператора `if...then...else`.

Упражнения

1. Вычислить значения логических выражений:

а) `K mod 7=K div 5-1 при K=15;`
б) `odd(trunc(10*P)) при P=0.182;`
в) `not odd(n) при n=0;`
г) `t and (P mod 3=0) при t=true, P=10101;`
д) `(x*y<>0) and (y>x) при x=2, y=1;`
е) `a or not b при a=false, b=true.`

2. Если `a=true` и `x=1`, то какое значение получит логическая переменная `d` после выполнения оператора присваивания:

а) `d:=x<2;` б) `d:=not a or odd(x);` в) `d:=ord(a)<>x`

3. Написать оператор присваивания, в результате выполнения которого логическая переменная `t` получит значение `true`, если высказывание истинно, и значение `false` — в противном случае, для следующих высказываний:

а) из чисел x, y, z только два равны между собой;
б) x — положительное число;
в) каждое из чисел x, y, z положительно;
г) только одно из чисел x, y, z положительно;
д) p делится нацело на q;
е) цифра 5 входит в десятичную запись трехзначного целого числа k.

3.10. Логические выражения в управляющих операторах

Алгоритмическая структура ветвления программируется в Паскале с помощью *условного оператора*. Раньше мы его описывали в таком виде:

If <условие> **Then** <оператор 1> **Else** <оператор 2>;

Кроме того, возможно использование неполной формы условного оператора:

If <условие> **Then** <оператор>;

Теперь дадим строгое описание условного оператора в форме синтаксической диаграммы (рис. 20).

То, что мы раньше называли условием, есть *логическое выражение*, которое вычисляется в первую очередь. Если его значение равно *true*, то будет выполняться <оператор 1> (после `Then`), если

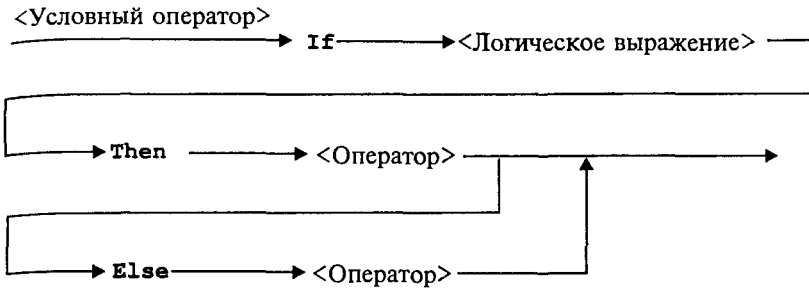


Рис. 20

false, то <оператор 2> (после `Else`) для полной формы или сразу следующий оператор после условного для неполной формы (без `Else`).

Пример 1. По длинам трех сторон треугольника a, b, c вычислить его площадь.

Для решения задачи используется формула Герона

$$\sqrt{p(p-a)(p-b)(p-c)},$$

где $p = (a + b + c)/2$ — полупериметр треугольника. Исходные данные должны удовлетворять основному соотношению для сторон треугольника: длина каждой стороны должна быть меньше длин двух других сторон.

Имея возможность в одном условном операторе записывать достаточно сложные логические выражения, мы можем сразу «отфильтровать» все варианты неверных исходных данных.

```
Program Geron;
Var A,B,C,P,S: Real;
Begin
   WriteLn('Введите длины сторон треугольника:');
   Write('a='); ReadLn(A);
   Write('b='); ReadLn(B);
   Write('c='); ReadLn(C);
   If (A>0) And (B>0) And (C>0) And (A+B>C)
       And (B+C>A) And (A+C>B)
  Then Begin
         P:=(A+B+C)/2;
         S:=Sqrt(P*(P-A)*(P-B)*(P-C));
         WriteLn('Площадь=',S)
       End
  Else WriteLn('Неверные исходные данные')
End.
```

Теперь рассмотрим синтаксическую диаграмму оператора **цикл-пока**, или **цикл с предусловием** (рис. 21).

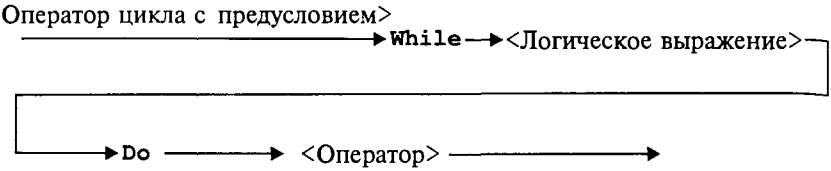


Рис. 21

Сначала вычисляется <Логическое выражение>. Пока его значение равно `true`, выполняется <Оператор> — тело цикла. Здесь <Оператор> может быть как простым, так и составным.

Пример 2. В следующем фрагменте программы на Паскале вычисляется сумма конечного числа членов гармонического ряда

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{i}+\ldots .$$

Суммирование прекращается, когда очередное слагаемое становится меньше ε или целая переменная *i* достигает значения `MaxInt`.

```
S:=0;
I:=1;
While (1/I>=Eps) And (I<MaxInt) Do
Begin
   S:=S+1/I;
   I:=I+1
End;
```

Синтаксическая диаграмма оператора **цикл-до, или цикл с постусловием**, представлена на рис. 22.

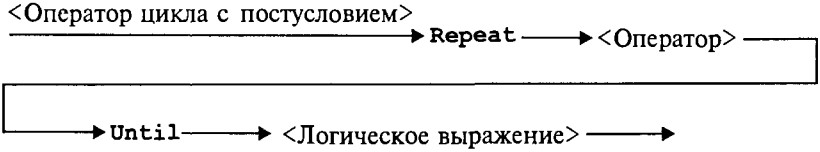


Рис. 22

Исполнение цикла повторяется до того момента, когда <Логическое выражение> станет равным *true*.

Предыдущая задача с использованием цикла с постусловием решается так:

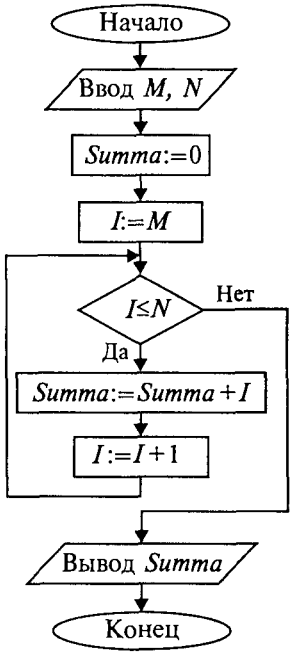
```
S:=0;
I:=1;
Repeat
   S:=S+1/I; I:=I+1
Until (1/I<Eps) Or (I>=MaxInt);
```

3.11. Цикл по параметру

Рассмотрим следующую простую задачу: требуется вычислить сумму целых чисел от M до N путем прямого суммирования. Здесь M и N — целые числа. Задачу можно сформулировать так:

$$Summa = \begin{cases} \sum_{i=M}^{N} i, & \text{если } M \le N, \\ 0, & \text{если } M > N. \end{cases}$$

Алгоритм и программа решения этой задачи с использованием структуры цикл-пока представлены на рис. 23.

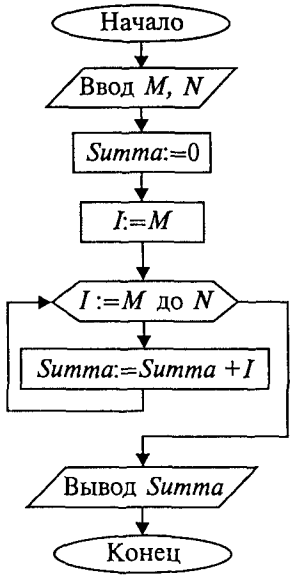



```
Program Adding;
Var I,M,N,Summa: Integer;
  Begin  Write('M=');
    ReadLn(M);
    Write('N=');
    ReadLn(N);
    Summa:=0;
    I:=M;
    While I<=N Do
    Begin
        Summa:=Summa+I;
        I:=Succ(I)
  End;
  WriteLn('Сумма равна',
          Summa)
End.
```

Рис. 23

А теперь введем новый тип циклической структуры, который будет называться **цикл по параметру**, или **цикл-для**. Блок-схема и программа на Паскале для решения рассматриваемой задачи с использованием этой структуры приведены на рис. 24.

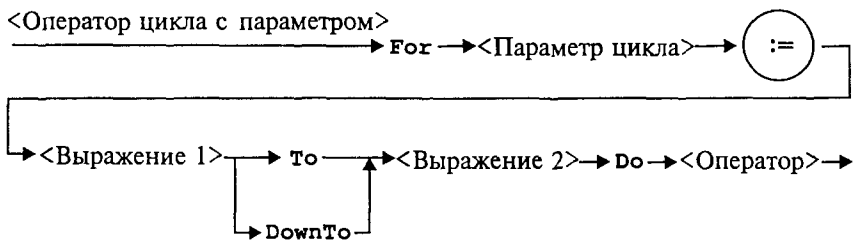
Здесь целая переменная I последовательно принимает значения в диапазоне от M до N. При каждом значении I выполняется тело цикла. После последнего выполнения цикла при $I = N$ происходит выход из цикла на продолжение алгоритма. Цикл выполняется хотя бы один раз, если $M \le N$, и не выполняется ни разу при $M > N$.




```
Program Summering_2;
Var I,M,N,Summa: Integer;
Begin
  Write('M=');
  ReadLn(M);
  Write('N=');
      ReadLn(N);
      Summa:=0;
  For I:=M To N Do
      Summa:=Summa+I;
  WriteLn('Сумма равна',
          Summa)
End.
```

Рис. 24

В программе используется оператор цикла **For**, синтаксическая диаграмма которого представлена на рис. 25.



Здесь <параметр цикла>::=<имя простой переменной порядкового типа>

Рис. 25

Выполнение оператора **For** в первом варианте (**To**) происходит по следующей схеме:

1. Вычисляются значения <Выражения 1> и <Выражения 2>. Это делается только один раз при входе в цикл.
2. Параметру цикла присваивается значение <Выражения 1>.
3. Значение параметра цикла сравнивается со значением <Выражения 2>. Если параметр цикла меньше или равен этому значению, то выполняется тело цикла, в противном случае выполнение цикла заканчивается.

4. Значение параметра цикла изменяется на следующее значение в его типе (для целых чисел — увеличивается на единицу); происходит возврат к пункту 3.

Оператор цикла **For** объединяет в себе действия, которые при использовании цикла **While** выполняют различные операторы: присваивание параметру начального значения, сравнение с конечным значением, изменение на следующее.

Как известно, результат суммирования целых чисел не зависит от порядка суммирования. Например, в рассматриваемой задаче числа можно складывать и в обратном порядке, т.е. от N до M ($N \geq M$). Для этого можно использовать второй вариант оператора цикла **For**:

```
Summa:=0;
For I:=N DownTo M Do
Summa:=Summa+I;
```

Слово **DownTo** буквально можно перевести как «вниз до». В таком случае параметр цикла изменяется по убыванию, т.е. при каждом повторении цикла параметр изменяет свое значение на предыдущее (равносильно `i:=pred(i)`). Тогда ясно, что цикл не выполняется ни разу, если $N < M$.

Работая с оператором **For**, учитывайте следующие правила:

- параметр цикла не может иметь тип `Real`;
- в теле цикла нельзя изменять переменную «параметр цикла»;
- при выходе из цикла значение переменной-параметра является неопределенным.

В следующем примере в качестве параметра цикла **For** используется символьная переменная. Пусть требуется получить на экране десятичные коды букв латинского алфавита. Как известно, латинские буквы в таблице кодировки упорядочены по алфавиту. Вот фрагмент такой программы:

```
For C:='a' To 'z' Do
Write (C,'-',Ord(C));
```

Здесь переменная `C` имеет тип `Char`.

А теперь подумайте сами, как вывести кодировку латинского алфавита в обратном порядке (от 'z' до 'a').

Упражнения

1. Составить программу полного решения квадратного уравнения (алгоритм см. в разд. 1.3).
2. Используя операторы цикла **While**, **Repeat** и **For**, составить три варианта программы вычисления $N!$.
3. Составить программу, по которой будет вводиться последовательность символов до тех пор, пока не встретится маленькая

или большая латинская буква *z*. Подсчитать, сколько раз среди вводимых символов встречалась буква *W*.

4. Вычислить сумму квадратов всех целых чисел, попадающих в интервал ($\ln x$, e^x), $x > 1$.

5. Вычислить количество точек с целочисленными координатами, попадающих в круг радиуса R ($R > 0$) с центром в начале координат.

6. Напечатать таблицу значений функции $\sin x$ и $\cos x$ на отрезке [0, 1] с шагом 0,1 в следующем виде:

x	$\sin x$	$\cos x$
0.0000	0.0000	1.0000
0.1000	0.0998	0.9950
...	...	...
1.0000	0.8415	0.5403

7. Напечатать в возрастающем порядке все трехзначные числа, в десятичной записи которых нет одинаковых цифр.

8. Дано целое $n > 2$. Напечатать все простые числа из диапазона [2, n].

3.12. Особенности целочисленной и вещественной арифметики

Числовые расчеты могут производиться на множестве целых чисел или на множестве вещественных чисел. С математической точки зрения целые числа являются подмножеством множества вещественных чисел. Поэтому, казалось бы, можно было бы и не разделять числа на целые и вещественные и иметь дело только с вещественным числовым типом данных.

Однако целочисленная арифметика на ЭВМ имеет три очень существенных преимущества по сравнению с вещественной арифметикой:

- целые числа всегда представимы своими точными значениями;
- операции целочисленной арифметики дают точные результаты;
- операции целочисленной арифметики выполняются быстрее, чем операции вещественной («плавающей») арифметики.

Недостатком целого типа данных является сравнительно узкий диапазон допустимых значений (для типа `Integer` — от –32768 до 32767). При исполнении программы автоматически не контролируется выход значения целой величины за эти границы. В этом случае получается ошибочный результат. Если такая опасность существует, то программист должен сам предусматривать в своей программе предупреждение целочисленного переполнения. Чаще всего целый тип используется для представления счетчиков, номеров, индексов и других целочисленных величин.

Вам уже известно, что целый тип данных является порядковым. Вспомним, что это значит:

- величины этого типа принимают конечное множество значений, которые могут быть пронумерованы;
- на множестве значений данного типа работают понятия: «предыдущий элемент», «последующий элемент».

Почему же вещественный тип данных не является упорядоченным? Вещественные числа в памяти ЭВМ представляются в формате с плавающей точкой, т.е. в виде совокупности пары чисел — целого порядка и нормализованной мантиссы. Поскольку размер ячейки памяти ограничен, в большинстве случаев мантисса оказывается «обрезанной», иными словами, приближенной. Точное представление в памяти имеет лишь дискретное конечное множество вещественных значений. Поэтому множество вещественных чисел в машинном представлении (рис. 26) есть дискретное, конечное множество, хотя оно и является отражением континуума действительных чисел.

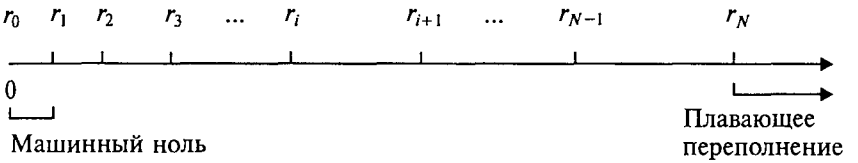


Рис. 26

На рисунке изображена положительная часть действительной числовой оси, на которой штрихами отмечены значения, точно представимые в вещественном типе данных. Эта картина симметрично отражается на отрицательную полуось.

С ростом абсолютного значения интервал между соседними точками растет. Он равен (при двоично-нормализованной форме с плавающей точкой) $2^{-t} \times 2^{p} = 2^{p-t}$, где p — порядок числа, а t — количество двоичных разрядов в мантиссе. Ясно, что с ростом абсолютной величины числа его порядок (p) растет и, следовательно, растет шаг между двумя соседними значениями. Минимальный шаг

$$\Delta r_{\min} = |r_1 - r_0| = 2^{p_{\min} - t};$$

максимальный шаг

$$\Delta r_{\max} = |r_N - r_{N-1}| = 2^{p_{\max} - t}.$$

Например, если $p_{\min} = -64$; $p_{\max} = 63$; $t = 24$, то имеем $\Delta r_{\min} = 2^{-88}$; $\Delta r_{\max} = 2^{39}$.

Казалось бы, значения множества точно представимых вещественных чисел можно пронумеровать и таким образом опреде-

лить на нем понятия «следующий», «предыдущий». Однако расстояние между двумя последовательными значениями на этом множестве оказывается величиной субъективной, в частности, зависящей от размера ячейки памяти, в которой хранится число. Например, если под мантиссу выделяется 3 байта, то следующее значение получается путем прибавления к мантиссе единицы в 24-м разряде; если 5 байт — единицы в 40-м разряде.

Бесконечное количество действительных чисел вообще непредставимо точно в памяти ЭВМ. Если вещественное значение X попадает между двумя точно представимыми значениями r_i и r_{i+1}, то оно заменяется на значение меньшего по модулю из этой пары чисел (некоторые типы процессоров выполняют «правильное» округление). Следовательно, в общем случае вещественные числа хранятся в памяти приближенно, т.е. несут в себе погрешность, которая называется *погрешностью машинного округления*.

Из сказанного следует, что если два числа X и Y удовлетворяют условиям $r_i < X < r_{i+1}$; $r_i < Y < r_{i+1}$, но $X \neq Y$, то в машинном представлении они неразличимы.

Разность между вещественной единицей и ближайшим к ней числом, представимым в памяти машины, называется *машинным эпсилон* ε. Иначе говоря, если $r_i = 1$, то $r_{i+1} = 1 + \varepsilon$. Легко понять, что величина машинного ε связана только с разрядностью мантиссы в представлении вещественных чисел на данной ЭВМ.

Для определения величины машинного ε можно использовать следующую программу:

```
Program Epsilon;
Var Eps: Real;
Begin Eps:=1/2;
    While 1.0+Eps>1.0 Do
          Eps:=Eps/2;
    WriteLn('Машинный эпсилон=',Eps)
End.
```

Упражнения

1. С помощью приведенной выше программы определите на вашем компьютере величину машинного ε.

2. Если на вашем компьютере это возможно, определите машинное ε для удвоенной или повышенной точности представления вещественных чисел.

3. Составьте программу определения машинного нуля. Проведите численный эксперимент на вашем компьютере.

4. Результат выполнения следующей программы доказывает некорректность сравнения на равенство двух вещественных чисел.

```
Program Test;
Var X,StepX: Real;
```

```
Begin X:=0;
    StepX:=0.1;
    Repeat
        X:=X+StepX
    Until X=1E3
End.
```

Проведите численный эксперимент, объясните результат.

5. Попробуйте экспериментально доказать, что в «плавающей» арифметике на ЭВМ не всегда выполняются законы ассоциативности: $(x + y) + z = x + (y + z)$ и дистрибутивности: $(x + y) \cdot z = xz + yz$.

6. Проведите эксперимент на исследование целочисленного переполнения, т.е. установите, что происходит с целой переменной, когда ее значение превышает `MaxInt`.

3.13. Подпрограммы

С понятием вспомогательного алгоритма вы уже знакомы (см. разд. 1.4). В языках программирования вспомогательные алгоритмы называются *подпрограммами*. В Паскале различаются две разновидности подпрограмм: *процедуры* и *функции*. Рассмотрим этот вопрос на примере следующей задачи: даны два натуральных числа a и b. Требуется определить наибольший общий делитель трех величин: $a + b$, $|a - b|$, $a \cdot b$. Запишем это так: НОД$(a + b, |a - b|, a \cdot b)$.

Идея решения состоит в следующем математическом факте: если x, y, z — три натуральных числа, то НОД$(x, y, z) =$ $=$ НОД(НОД$(x, y), z)$. Иначе говоря, нужно найти НОД двух величин, а затем НОД полученного значения и третьего числа (попробуйте это доказать).

Очевидно, что вспомогательным алгоритмом для решения поставленной задачи является алгоритм получения наибольшего общего делителя двух чисел. Эта задача решается с помощью известного алгоритма Евклида (см. раздел 1.3). Запишем его в форме процедуры на алгоритмическом языке.

```
Процедура Евклид(цел M,N,K);
нач
  пока M<>N
  нц
    если M>N
    то M:=M-N
    иначе N:=N-M
    кв
  кц;
  K:=M
кон
```

Здесь M и N являются формальными параметрами процедуры. M и N параметры-аргументы, K — параметр-результат. Основной алгоритм, решающий исходную задачу, будет следующим:

```
алг задача;
цел a,b,c;
нач ввод(a,b);
    Евклид(a+b,|a-b|,c);
    Евклид(c,a*b,c);
    вывод(c)
кон.
```

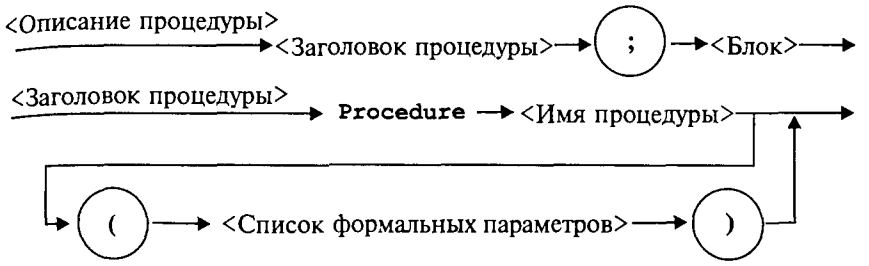
Процедуры в Паскале. Основное отличие процедур в Паскале от процедур в Алгоритмическом языке (АЯ) состоит в том, что процедуры в Паскале описываются в разделе описания подпрограмм, а в АЯ процедура является внешней по отношению к вызывающей программе. Теперь посмотрим, как решение поставленной задачи программируется на Турбо Паскале.

```
Program NOD1;
Var A,B,C: Integer;
Procedure Evklid(M,N: Integer; Var K:
          Integer);
Begin
    While M<>N Do
    If M>N
    Then M:=M-N
    Else N:=N-M;
    K:=M
End;
Begin
    Write('a=');
    ReadLn(A);
    Write('b=');
    ReadLn(B);
    Evklid(A+B,Abs(A-B),C);
    Evklid(C,A*B,C);
    WriteLn('НОД=',C)
End.
```

В данном примере *обмен* аргументами и результатами между основной программой и процедурой производится через *параметры* (*формальные* и *фактические*). Существует и другой механизм обмена — через глобальные переменные. Но об этом чуть позже. А сейчас рассмотрим синтаксическую диаграмму описания процедуры (рис. 27).

Из диаграммы видно, что процедура может иметь параметры, а может быть и без них. Чаще всего аргументы представляются как па-

раметры-значения (хотя могут быть и параметрами-переменными). А для передачи результатов используются параметры-переменные.




```
<формальные параметры>::=<параметры-значения>
                        |<параметры-переменные>
```

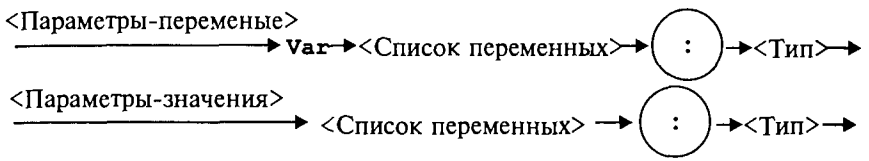


Рис. 27

Процедура в качестве результата может передавать в вызывающую программу множество значений (в частном случае — одно), а может и ни одного. Теперь рассмотрим правила обращения к процедуре. Обращение к процедуре производится в форме *оператора процедуры* (рис. 28).

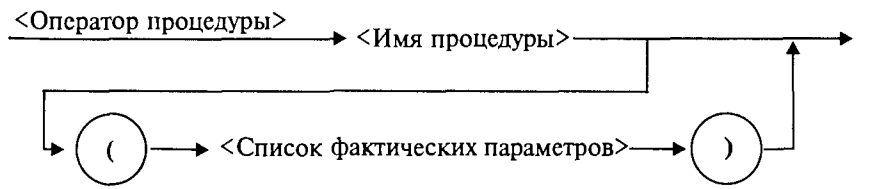


Рис. 28

Если описана процедура с формальными параметрами, то и обращение к ней производится оператором процедуры с фактическими параметрами. Правила соответствия между формальными и фактическими параметрами: соответствие *по количеству,* соответствие *по последовательности* и соответствие *по типам.*

Первый вариант взаимодействия формальных и фактических параметров называется передачей *по значению*: вычисляется значение фактического параметра (выражения) и это значение при-

сваивается соответствующему формальному параметру. Второй вариант взаимодействия называется передачей *по имени*: при выполнении процедуры имя формальной переменной заменяется на имя соответствующей фактической переменной (в откомпилированной программе имени переменной соответствует адрес ячейки памяти).

В рассмотренном нами примере формальные параметры M и N являются параметрами-значениями. Это аргументы процедуры. При обращении к ней первый раз им соответствуют значения выражений $a + b$ и $\text{abs}(a - b)$; второй раз — c и $a \cdot b$. Параметр K является параметром-переменной. В ней получается результат работы процедуры. В обоих обращениях к процедуре соответствующим фактическим параметром является переменная c. Через эту переменную основная программа получает результат.

Теперь рассмотрим другой вариант программы, решающей ту же задачу. В ней используется процедура *без параметров*.

```
Program NOD2;
Var A,B,K,M,N: Integer;
Procedure Evklid;
Begin
    While M<>N Do
    If M>N
    Then M:=M-N
    Else N:=N-M;
    K:=M
End;
Begin
    Write('a=');
    ReadLn(A);
    Write('b=');
    ReadLn(B);
    M:=A+B;
    N:=Abs(A-B);
    Evklid;
    M:=K;
    N:=A*B;
    Evklid;
    WriteLn('НОД равен',K)
End.
```

Чтобы разобраться в этом примере, требуется объяснить новое для нас понятие: *область действия описания*.

Областью действия описания любого программного объекта (переменной, типа, константы и т.д.) является тот блок, в котором расположено это описание. Если данный блок вложен в другой (подпрограмма), то присутствующие в нем описания являют-

ся локальными. Они действуют только в пределах внутреннего блока. Описания же, стоящие во внешнем блоке, называются глобальными по отношению к внутреннему блоку. Если глобально описанный объект используется во внутреннем блоке, то на него распространяется внешнее (глобальное) описание.

В программе `NOD1` переменные *M*, *N*, *K* — локальные внутри процедуры; переменные *a*, *b*, *c* — глобальные. Однако внутри процедуры переменные *a*, *b*, *c* не используются. Связь между внешним блоком и процедурой осуществляется через параметры.

В программе `NOD2` все переменные являются глобальными. В процедуре `Evklid` нет ни одной локальной переменной (нет и параметров). Поэтому переменные *M* и *N*, используемые в процедуре, получают свои значения через оператор присваивания в основном блоке программы. Результат получается в глобальной переменной *K*, значение которой выводится на экран.

Использование механизма передачи через параметры делает процедуру более универсальной, независимой от основной программы. Однако в некоторых случаях оказывается удобнее использовать передачу через глобальные переменные. Чаще такое бывает с процедурами, работающими с большими объемами информации. В этой ситуации глобальное взаимодействие экономит память ЭВМ.

Функции. Теперь выясним, что такое подпрограмма-функция. Обычно функция используется в том случае, если результатом подпрограммы должна быть скалярная (простая) величина. Тип результата называется типом функции. В Турбо Паскале допускаются функции строкового типа. Синтаксическая диаграмма описания функции представлена на рис. 29.

Как и у процедуры, у функции в списке формальных параметров могут присутствовать параметры-переменные и параметры-значения. Все это аргументы функции. Параметры вообще могут отсутствовать (если аргументы передаются глобально).

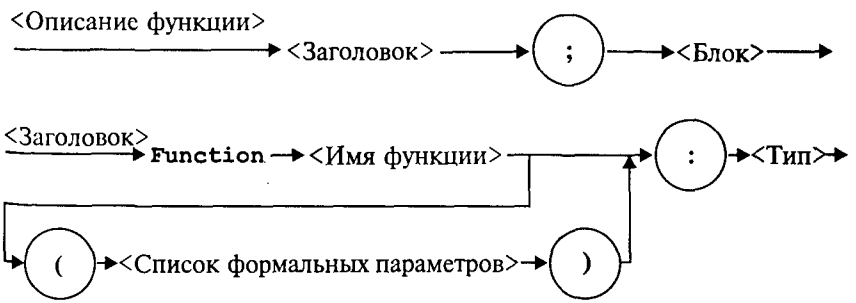


Рис. 29

Программа решения рассмотренной выше задачи с использованием функции будет выглядеть следующим образом:

```
Program NOD3;
Var A,B,Rez:Integer;
Function Evklid(M,N:Integer):Integer;
  Begin
     While M<>N Do
     If M>N
     Then M:=M-N
     Else N:=N-M;
     Evklid:=M
  End;
     Begin
     Write('a=');
     ReadLn(A);
     Write('b=');
     ReadLn(B);
     Rez:=Evklid(Evklid(A+B,
          Abs(A-B)),A*B);
     WriteLn('NOD равен',Rez)
End.
```

Из примера видно, что тело функции отличается от тела процедуры только тем, что в функции *результат присваивается переменной с тем же именем, что и функция.*

Обращение к функции является операндом в выражении. Оно записывается в следующей форме:

```
<Имя функции> (<Список фактических параметров>)
```

Правила соответствия между формальными и фактическими параметрами все те же. Сравнивая приведенные выше программы, можно сделать вывод, что программа NOD3 имеет определенные преимущества перед другими. Функция позволяет получить результат путем выполнения одного оператора присваивания. Здесь иллюстрируется возможность того, что фактическим аргументом при обращении к функции может быть эта же функция.

По правилам стандарта Паскаля возврат в вызывающую программу из подпрограммы происходит, когда выполнение подпрограммы доходит до ее конца (последний **End**). Однако в Турбо Паскале есть средство, позволяющее выйти из подпрограммы в любом ее месте. Это оператор-процедура Exit. Например, функцию определения наибольшего из двух данных вещественных чиссл можно описать так:

```
Function Max(X,Y: Real): Real;
Begin
   Max:=X;
```

```
    If X>Y Then Exit Else Max:=Y
End;
```

Еще раз об области действия описаний. В Паскале неукоснительно действует следующее правило: *любой программный объект (константа, переменная, тип и т.п.) должен быть описан перед использованием в программе.* Иначе говоря, описание объекта должно предшествовать его первому появлению в других фрагментах программы. Это правило относится и к подпрограммам.

На рис. 30 схематически показана структура взаимного расположения описаний подпрограмм в некоторой условной программе. Попробуем, используя эту схему, разобраться в вопросе об области действия описаний подпрограмм.

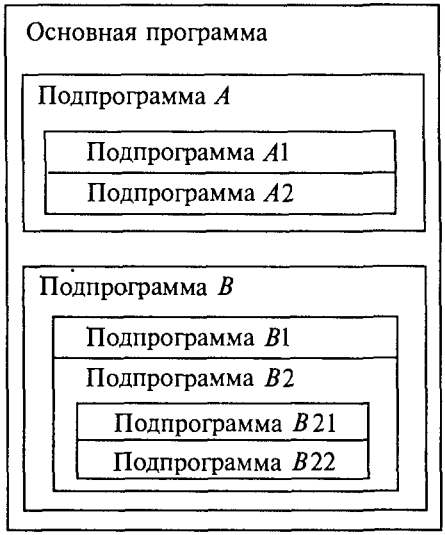


Рис. 30

Любая подпрограмма может использоваться лишь в пределах области действия ее описания. Например, область действия подпрограмм *A* и *B* — основная программа. Поэтому из основной программы можно обратиться к подпрограммам *A* и *B*. В свою очередь, в подпрограмме *B* могут быть обращения к подпрограмме *A*; а из *A* нельзя обратиться к *B*, поскольку описание *A* предшествует описанию *B*. Подпрограммы *A*1 и *A*2 локализованы в подпрограмме *A* и могут использоваться только в ней; из *A*2 можно обратиться к *A*1, но не наоборот.

Из подпрограммы *B*1 можно обратиться к *A*, поскольку ее описание является глобальным по отношению к *B*1, но нельзя обратиться к *A*1, поскольку область действия описания *A*1 не распространяется на блок подпрограммы *B*.

Из подпрограммы *B*22 можно обратиться только к *B*21, *B*1, *A*. Объясните сами почему.

Все понятно? Если нет, то прочитайте еще раз этот раздел. Очень важно в нем разобраться.

Если одно и то же имя описано во внешнем блоке (глобально) и во внутреннем блоке (локально), то последнее описание (локальное) перекрывает первое в пределах внутреннего блока. Рассмотрим следующий пример:

```
Program Example1;
Var X: Integer;
Procedure P;
Var X: Integer;
Begin WriteLn('x=',X)
End;
Begin X:=1;
 P
End.
```

```
Program Example2;
Var X: Integer;
Procedure P;
Begin
     WriteLn('x=',X)
End;
Begin X:=1;
P
End.
```

Что выведется на экран в результате работы программы Example1 и Example2? Первая программа выдаст результат:

```
x=...
```

На месте многоточия будет какое-то произвольное значение, соответствующее неопределенной величине *x*. Вторая программа в результате даст

```
x=1
```

В первом случае переменная с именем *x* описана как глобально, так и локально. Но процедура выводит значение локальной переменной, которой ничего не присвоено. В этом примере идентификатором *x* обозначены две совершенно разные величины, им соответствуют две разные ячейки памяти.

Во втором примере переменная *x* одна на всю программу. Она описана глобально. Поэтому значение 1, присвоенное ей в основной программе, передается и в подпрограмму.

Далее разговор пойдет о ситуации на первый взгляд совершенно парадоксальной. Оказывается, подпрограмма в своем описании может содержать обращение к самой себе. Такая подпрограмма называется рекурсивной.

Рекурсивные подпрограммы. В математике рекурсивным называется определение любого понятия через самое себя. Классическим примером является определение факториала целого числа, большего или равного нулю:

$$n! = \begin{cases} 1, & \text{если } n = 0, \\ n(n-1)!, & \text{если } n > 0. \end{cases}$$

Здесь функция факториала определена через факториал. Нетрудно понять справедливость такого определения. Для $n > 0$

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \ldots (n-1) \cdot n = n \cdot \prod_{i=1}^{n-1} i = n(n-1)!$$

Вариант 0!=1 является тривиальным. Но это «опорное» значение, от которого начинается раскручивание всех последующих значений факториала:

$$1! = 1 \times 0! = 1 \times 1 = 1;\ 2! = 2 \times 1 = 2;\ 3! = 3 \times 2! = 6 \text{ и т.д.}$$

Рассмотрим подпрограмму-функцию, использующую в своем описании приведенную выше рекурсивную формулу.

```
Function Factor(N: PozInt): PozInt;
Begin
     If N=0
     Then Factor:=1
     Else Factor:=N*Factor(N-1)
End;
```

Предполагается, что тип `PozInt` объявлен глобально следующим образом:

```
Type PozInt=0..MaxInt;
```

Пусть в основной программе для вычисления в целой переменной *x* значения 3! используется оператор

```
X:=Factor(3);
```

При вычислении функции с аргументом 3 произойдет повторное обращение к функции `Factor(2)`. Это обращение потребует вычисления `Factor(1)`. И наконец, при вычислении `Factor(0)` будет получен числовой результат 1. Затем цепочка вычислений раскрутится в обратном порядке:

```
Factor(1)=1*Factor(0)=1
Factor(2)=2*Factor(1)=2
Factor(3)=3*Factor(2)=6.
```

Последовательность рекурсивных обращений к функции должна обязательно выходить на определенное значение. А весь маршрут последовательных вхождений машина запоминает в специальной области памяти, называемой *стеком*. Таким образом, выполнение рекурсивной функции происходит в два этапа: прямой ход — заполнение стека; обратный ход — цепочка вычислений по обратному маршруту, сохраненному в стеке.

Использование рекурсивных функций — красивый прием с точки зрения программистской эстетики. Однако этот путь не всегда самый рациональный. Рассмотренную задачу с *n*! можно решить так:

```
F:=1;
For I:=1 To N Do
F:=F*I;
```

Очевидно, что такой вариант программы будет работать быстрее, чем рекурсивный. И в том случае, когда важнейшим является сокращение времени выполнения программы, следует отдать предпочтение последнему варианту.

В каждой конкретной реализации Паскаля имеется ограничение на количество рекурсивных обращений к подпрограмме (глубина рекурсии). Это связано с ограничением на размер стека. По этой причине можно попасть в ситуацию, когда рекурсивной подпрограммой вообще не удастся воспользоваться.

Рекурсивно определена может быть не только функция, но и процедура. Примеры использования рекурсивно-определенных процедур рассматриваются в пятой главе.

Упражнения

1. Составить программу вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя подпрограмму вычисления площади круга (2 варианта: с процедурой и с функцией).

2. По координатам вершин треугольника вычислить его периметр, используя подпрограмму вычисления длины отрезка, соединяющего две точки.

3. Даны три целых числа. Определить, сумма цифр которого из них больше. Подсчет суммы цифр организовать через подпрограмму.

4. Определить площадь выпуклого четырехугольника по заданным координатам вершин. Использовать подпрограмму-функцию вычисления длины отрезка и подпрограмму-процедуру вычисления площади треугольника по формуле Герона.

5. Описать рекурсивную функцию $pow(x, n)$ от вещественного x ($x \neq 0$) и целого n, которая вычисляет величину x^n по формуле

$$x^n = \begin{cases} 1 & \text{при } n = 0, \\ \dfrac{1}{x^{-n}} & \text{при } n < 0, \\ x \cdot x^{n-1} & \text{при } n > 0. \end{cases}$$

6. Даны натуральные числа n и m; найти НОД (n, m). Использовать программу, включающую в себя рекурсивную процедуру вычисления НОД, и основанную на соотношении НОД (n, m) = = НОД (m, r), где r — остаток от деления n на m.

3.14. Вычисление рекуррентных последовательностей

Рекуррентная последовательность. Из курса математики известно понятие *рекуррентной последовательности.* Это понятие вводится так: пусть известно k чисел $a_1, \ldots, a_k$. Эти числа являются первыми числами числовой последовательности. Следующие элементы данной последовательности вычисляются так:

$$a_{k+1} = F(a_1, \ldots, a_k);\ a_{k+2} = F(a_2, \ldots, a_{k+1});\ a_{k+3} = F(a_3, \ldots, a_{k+2});\ \ldots$$

Здесь F — функция от k аргументов. Формула вида

$$a_i = F(a_{i-1}, a_{i-2}, \ldots, a_{i-k})$$

называется *рекуррентной формулой.* Величина k называется *глубиной рекурсии.*

Другими словами, можно сказать, что *рекуррентная последовательность — это бесконечный ряд чисел, каждое из которых, за исключением k начальных, выражается через предыдущие.*

Примерами рекуррентных последовательностей являются арифметическая (1) и геометрическая (2) прогрессии:

$$a_1 = 1,\ a_2 = 3,\ a_3 = 5,\ a_4 = 7,\ a_5 = 9, \ldots \quad (1)$$
$$a_1 = 1,\ a_2 = 2,\ a_3 = 4,\ a_4 = 8,\ a_5 = 16, \ldots \quad (2)$$

Рекуррентная формула для указанной арифметической прогрессии:

$$a_i = a_{i-1} + 2.$$

Рекуррентная формула для данной геометрической прогрессии:

$$a_i = 2 \cdot a_{i-1}.$$

Глубина рекурсии в обоих случаях равна единице (такую зависимость еще называют одношаговой рекурсией). В целом рекуррентная последовательность описывается совокупностью начальных значений и рекуррентной формулы. Все это можно объединить в одну ветвящуюся формулу. Для арифметической прогрессии:

$$a_i = \begin{cases} 1, & \text{если } i = 1, \\ a_{i-1} + 2, & \text{если } i > 1. \end{cases}$$

Для геометрической прогрессии:

$$a_i = \begin{cases} 1, & \text{если } i = 1, \\ 2 \cdot a_{i-1}, & \text{если } i > 1. \end{cases}$$

Следующая числовая последовательность известна в математике под названием чисел Фибоначчи:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Начиная с третьего элемента каждое число равно сумме значений двух предыдущих, т.е. это рекуррентная последовательность с глубиной равной 2 (двухшаговая рекурсия). Опишем ее в ветвящейся форме:

$$a_i = \begin{cases} 1, & i = 1,\ i = 2, \\ f_{i-1} + f_{i-2}, & i > 2. \end{cases}$$

Введение представления о рекуррентных последовательностях позволяет по-новому взглянуть на некоторые уже известные нам задачи. Например, факториал целого числа $n!$ можно рассматривать как значение n-го элемента следующего ряда чисел:

$$a_0 = 1,\ a_1 = 1!,\ a_2 = 2!,\ a_3 = 3!,\ a_4 = 4!,\ \ldots$$

Рекуррентное описание такой последовательности выглядит следующим образом:

$$a_i = \begin{cases} 1, & i = 0, \\ a_{i-1} \cdot i, & i > 0. \end{cases}$$

Программирование вычислений рекуррентных последовательностей. С рекуррентными последовательностями связаны задачи такого рода:

1) вычислить заданный (n-й) элемент последовательности;

2) математически обработать определенную часть последовательности (например, вычислить сумму или произведение первых n членов);

3) подсчитать количество элементов на заданном отрезке последовательности, удовлетворяющих определенным свойствам;

4) определить номер первого элемента, удовлетворяющего определенному условию;

5) вычислить и сохранить в памяти заданное количество элементов последовательности.

Данный перечень задач не претендует на полноту, но наиболее часто встречающиеся типы он охватывает. В четырех первых задачах не требуется одновременно хранить в памяти множество элементов числового ряда. В таком случае *его элементы могут получаться последовательно в одной переменной, сменяя друг друга.*

Пример 1. Вычислить n-й элемент арифметической прогрессии (1).

```
Var N,I: 0..MaxInt;
A: Real;
Begin
  Write('N=');
  ReadLn(N);
  A:=1;
  For I:=2 To N Do
```

```
    A:=A+2;
    WriteLn('A(',N:1,')=',A:6:0)
End.
```

Рекуррентная формула $a_i = a_{i-1} + 2$ перешла в оператор $A := A + 2$.

Пример 2. Просуммировать первые n элементов геометрической прогрессии (2) (не пользуясь формулой для суммы первых n членов прогрессии).

```
Var N,I: 0..MaxInt;
A,S: Real;
Begin
   Write('N='); ReadLn(N);
   A:=1;
   S:=A;
   For I:=2 To N Do
   Begin
      A:=2*A;
      S:=S+A
   End;
   WriteLn('Сумма равна',S:6:0)
End.
```

При вычислении рекуррентной последовательности с глубиной 2 уже нельзя обойтись одной переменной. Это видно из следующего примера.

Пример 3. Вывести на печать первые n ($n \geq 3$) чисел Фибоначчи. Подсчитать, сколько среди них четных чисел.

```
Var N,I,K,F,F1,F2: 0..MaxInt;
Begin
   F1:=1; F2:=1;
   K:=0;
   WriteLn('F(1)=',F1,'F(2)=',F2);
   For I:=3 To N Do
   Begin
      F:=F1+F2;
      WriteLn('F(',I:1,')=',F);
      If Not Odd(F) Then K:=K+1;
      F1:=F2; F2:=F
   End;
   WriteLn('Количество четных чисел в
   последовательности равно',K)
End.
```

Понадобились три переменные для последовательного вычисления двухшаговой рекурсии, поскольку для нахождения очередного элемента необходимо помнить значения двух предыдущих.

Пример 4. Для заданного вещественного x и малой величины ε (например, $\varepsilon = 0{,}000001$) вычислить сумму ряда

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,$$

включив в нее только слагаемые, превышающие ε. Известно, что сумма такого бесконечного ряда имеет конечное значение, равное e^x, где $e = 2{,}71828\ldots$ — основание натурального логарифма. Поскольку элементы этого ряда представляют собой убывающую последовательность чисел, стремящуюся к нулю, то суммирование нужно производить до первого слагаемого, по абсолютной величине не превышающего ε.

Если слагаемые в этом выражении обозначить следующим образом:

$$a_0 = 1,\ a_1 = x,\ a_2 = \frac{x^2}{2!},\ a_3 = \frac{x^3}{3!},\ \ldots,$$

то обобщенная формула для i-го элемента будет следующей:

$$a_i = \frac{x_i}{i!}.$$

Нетрудно увидеть, что между элементами данной последовательности имеется рекуррентная зависимость. Ее можно найти интуитивно, но можно и вывести формально. Правда, для этого нужно догадаться, что рекурсия — одношаговая, и что каждый следующий элемент получается путем умножения предыдущего на некоторый множитель, т.е.

$$a_i = K \cdot a_{i-1}.$$

Используя обобщенную формулу, имеем:

$$\frac{x_i}{i!} = K \cdot \frac{x_{i-1}}{(i-1)!}.$$

Отсюда:

$$K = \frac{x_i / i!}{x_{i-1} / (i-1)!} = \frac{x}{i}.$$

Действительно:

$$a_0 = 1,\ a_1 = a_0 \cdot \frac{x}{2},\ a_3 = a_2 \cdot \frac{x}{3} \text{ и т.д.}$$

Следовательно, данная рекуррентная последовательность может быть описана следующим образом:

$$a_i = \begin{cases} 1, & \text{если } i = 0, \\ a_{i-1} \cdot \dfrac{x}{i}, & \text{если } i > 0. \end{cases}$$

И наконец, приведем программу, решающую поставленную задачу.

```
Var A,X,S,Eps: Real;
    I: Integer;
Begin
    Write('X ='); ReadLn(X);
    Write('Epsilon ='); ReadLn(Eps);
    A:=1; S:=0; I:=0;
    While Abs(A)>Eps Do
    Begin
        S:=S+A;
        I:=I+1;
        A:=A*X/I
    End;
    WriteLn('Сумма ряда равна', S:10:4)
End.
```

Как и прежде, значения одношаговой рекуррентной последовательности вычисляются в одной переменной.

Каждое повторное выполнение цикла в этой программе приближает значение `S` к искомому (уточняет значащие цифры в его записи). Такой вычислительный процесс в математике называется *итерационным процессом*. Соответственно, циклы, реализующие итерационный вычислительный процесс, называются *итерационными циклами*. Для их организации используются операторы `While` или `Repeat`.

Пример 5. Для заданного натурального N и вещественного x ($x > 0$) вычислить значение выражения:

$$\sqrt{x+\sqrt{x+\ldots+\sqrt{x}}}.$$

В этом случае рекуррентность не столь очевидна. Попробуем найти ее методом индукции. Будем считать, что искомое выражение есть N-й элемент последовательности следующего вида:

$$a_1 = \sqrt{x},\ a_2 = \sqrt{x+\sqrt{x}},\ a_3 = \sqrt{x+\sqrt{x+\sqrt{x}}},\ldots$$

Отсюда видна связь:

$$a_2 = \sqrt{x+a_1},\ a_3 = \sqrt{x+a_2},\ \ldots,\ a_i = \sqrt{x+a_{i-1}}.$$

Теперь поставленная задача решается очень просто:

```
Var A,X: Real; I,N: Integer;
Begin
   Write('X='); ReadLn(X);
   Write('N='); ReadLn(N);
```

```
    A:= Sqrt(X);
    For I:= 2 To N Do
       A:=Sqrt(X+A);
    WriteLn('Ответ:',A)
End.
```

К решению всех перечисленных выше задач можно подойти иначе.

Вспомним о рекурсивно определенных подпрограммах. Посмотрите на описание арифметической прогрессии в форме рекуррентной последовательности. Из него непосредственно вытекает способ определения функции для вычисления заданного элемента прогрессии.

Сделаем это для общего случая, определив арифметическую прогрессию с первым членом a_0 и разностью d:

$$a_i = \begin{cases} a_0, & \text{если } i = 1, \\ a_{i-1} + d, & \text{если } i > 1. \end{cases}$$

Соответствующая подпрограмма-функция выглядит так:

```
Function Progres(A0,D: Real;I:
                 Integer): Real;
Begin
   If I=1
   Then Progres:=A0
   Else Progres:=Progres(A0,D,I-1)+D
End;
```

Следующая программа выводит на экран первые 20 чисел Фибоначчи, значения которых вычисляет рекурсивная функция Fibon.

```
Var K: Byte;
Function Fibon(N: Integer): Integer;
Begin
   If (N=1) Or (N=2)
   Then Fibon:=1
   Else Fibon:=Fibon(N-1)+Fibon(N-2)
End;
Begin
   For K:=1 To 20 Do WriteLn(Fibon(K))
End.
```

Необходимо отметить, что использование рекурсивных функций ведет к замедлению счета. Кроме того, можно столкнуться с проблемой нехватки длины стека, в котором запоминается «маршрут» рекурсивных обращений.

Рекуррентные последовательности часто используются для решения разного рода эволюционных задач, т.е. задач, в которых прослеживается какой-то процесс, развивающийся во времени. Рассмотрим такую задачу.

Пример 6. В ходе лечебного голодания масса пациента за 30 дней снизилась с 96 до 70 кг. Было установлено, что ежедневные потери массы пропорциональны массе тела. Вычислить, чему была равна масса пациента через k дней после начала голодания для $k = 1, 2, ..., 29$.

Обозначим массу пациента в i-й день через p_i ($i = 0, 1, 2, ..., 30$). Из условия задачи известно, что $p_0 = 96$ кг, $p_{30} = 70$ кг.

Пусть K — коэффициент пропорциональности убывания массы за один день.

Тогда

$$p_1 = p_0 - Kp_0 = (1 - K)p_0; \quad p_2 = (1 - K)p_1;$$
$$p_3 = (1 - K)p_2; \quad \ldots; \quad p_i = (1 - K)p_{i-1}.$$

Получаем последовательность, описываемую следующей рекуррентной формулой:

$$p_i = \begin{cases} 96, & \text{если } i = 0, \\ (1-K)\cdot p_{i-1}, & \text{если } 1 \le i \le 30. \end{cases}$$

Однако нам неизвестен коэффициент K. Его можно найти, используя условие p_{30}=70.

Для этого будем делать обратные подстановки:

$$p_{30} = (1 - K)\cdot p_{29} = (1 - K)^2 \cdot p_{28} = (1 - K)^3 \cdot p_{27} = \ldots =$$
$$= (1 - K)^{30} \cdot p_0 = (1 - K)^{30} \cdot 96.$$

Из равенства $(1 - K)^{30} \cdot 96 = 70$ находим: $1 - K = \left(\frac{70}{96}\right)^{\frac{1}{30}}$.

Далее программирование становится тривиальным.

```
Var I: Byte;
    P,Q: Real;
Begin
   P:=96;
   Q:=Exp(1/30*Ln(70/96));
   For I:=1 To 29 Do
   Begin
     P:=Q*P;
     WriteLn(I,'-й день-',P:5:3,'кг')
   End
End.
```

Упражнения

1. Рекуррентная последовательность определена следующим образом:

$$a_i = \begin{cases} 1, & \text{если } i = 0, \\ a_{i-1} \cdot i + \dfrac{1}{i}, & \text{если } i > 0. \end{cases}$$

Для данного натурального n получить значение a_n.

2. Дана последовательность:

$$a_i = \begin{cases} 1, & \text{если } i = 0, \quad i = 1, \\ a_{i-2} + \dfrac{a_{i-1}}{i-1}, & \text{если } i > 1. \end{cases}$$

Вычислить произведение элементов с 1-го по 20-й.

3. Используя рекуррентный подход, вычислить сумму многочлена 10-й степени по формуле Горнера, где x — данное вещественное число:

$$10x^{10} + 9x^9 + 8x^8 + \ldots + 2x^2 + x = ((\ldots((10x + 9)x + 8)x + \ldots + 2)x + 1)x.$$

4. Для данного вещественного x и натурального N вычислить цепную дробь: $x/(1 + x/(2 + x/(3 + x/(\ldots/(N + x))\ldots)$.

5. Вычислить и вывести все члены числового ряда

$$1, \ \frac{1}{2!}, \ \frac{1}{3!}, \ \ldots, \ \frac{1}{N!},$$

превышающие значение 10^{-5}.

6. Функцию $y = \sqrt{x}$ можно вычислить как предельное значение последовательности, определяемой рекуррентной формулой:

$$y_k = \frac{1}{2}\left(y_{k-1} + \frac{x}{y_{k-1}}\right); \ k = 1, 2, \ldots$$

Начальное значение y_0 задается произвольно (желательно ближе к $\sqrt{x}$). За приближенное значение корня с точностью ε берется первое y_k, для которого выполняется условие: $|y_k - y_{k-1}| < \varepsilon$. Составить программу.

3.15. Основные понятия и средства компьютерной графики в Турбо Паскале

До сих пор мы использовали экран компьютера только для вывода символьной информации — чисел, текстов. Однако Турбо Паскаль позволяет выводить на экран рисунки, чертежи, графики

функций, диаграммы и т.п., все то, что принято называть компьютерной графикой.

В стандарте Паскаля графический вывод не предусмотрен. Однако на разных типах компьютеров, в разных реализациях Паскаля существуют различные программные средства графического вывода — специальные наборы данных, функций, процедур. Несмотря на такое разнообразие, имеются общие понятия и средства, свойственные любому варианту реализации графики в любом языке программирования. В данном разделе лекций мы затронем только такие базовые средства.

Начиная с четвертой версии Турбо Паскаля для IBM PC появилась мощная графическая библиотека, организованная в модуль `Graph`. В приложении 2 в справочной форме дано описание основных компонент этого модуля. В рассмотренных ниже примерах программ используется модуль `Graph`. Для его подключения в начале программы необходимо написать строку:

```
Uses Graph;
```

Графические режимы экрана. Для вывода графических изображений необходимо перевести экран в один из графических режимов. В графическом режиме можно из программы управлять состоянием каждого пиксела (точечного элемента) экрана.

Графические режимы отличаются:

- размером графической сетки ($M \times N$, где M — число точек по горизонтали, N — число точек по вертикали);
- цветностью (число воспроизводимых на экране цветов).

Допустимые режимы зависят от типа монитора и соответствующего графического драйвера, используемого на компьютере.

Для установки графического режима экрана существуют соответствующие процедуры. В модуле `Graph` процедура установки графического режима экрана имеет следующий заголовок:

```
Procedure InitGraph(Var Driver,Mode: Integer;
Path: String);
```

Здесь целая переменная `Driver` определяет тип графического драйвера; целая переменная `Mode` задает режим работы графического драйвера; `Path` — выражение типа `String`, содержащее маршрут поиска файла графического драйвера.

Список констант модуля `Graph`, определяющих типы драйверов и режимы, приведен в табл. П2.1 приложения 2.

Вот пример программы, инициализирующей графический режим VGAHi для работы с драйвером VGA (монитор типа VGA).

```
Uses Graph;
Var Driver,Mode: Integer;
Begin
```

```
Driver: = VGA;{драйвер}
Mode: = VGAHi;{режим работы}
InitGraph(Driver,Mode,'C:\TP\BGI');
```

Здесь указывается, что файл *egavga.bgi* с драйвером для VGA-монитора находится в каталоге C:\TP\BGI. Режим VGAHi соответствует графической сетке 640 × 480 с палитрой из 16 цветов.

Возможно также автоматическое определение типа драйвера и установка режима. Этот прием позволяет программе работать с разными типами мониторов, не внося изменений в текст:

```
Driver:=Detect;
InitGraph(Driver,Mode,'C:\TP\BGI');
```

При этом автоматически устанавливается режим с наибольшей разрешающей способностью и цветностью. После окончания работы в графическом режиме следует вернуться в текстовый режим экрана.

В модуле `Graph` процедура возвращения в текстовый режим имеет заголовок:

```
Procedure CloseGraph;
```

Цвет фона и цвет рисунка. На цветном мониторе можно менять окраску экрана. Установленная окраска экрана называется цветом фона. Рисунок на этом фоне наносится с помощью разнообразных линий: прямых, окружностей, прямоугольников, ломаных и т.д. Цвета этих линий также могут меняться.

В табл. П2.2 приложения 2 приведены имена констант, определяющих 16 цветов палитры для мониторов типа EGA, VGA.

Заголовок процедуры установки цвета фона:

```
Procedure SetBkColor(Color: Word);
```

Здесь `Color` — выражение целого типа, определяющее номер цвета фона.

Заголовок процедуры установки цвета линий:

```
Procedure SetColor(Color: Word);
```

Заметим, что если в качестве номера цвета линии указывается 0, то это всегда совпадает с цветом фона (невидимая линия).

Если необходимо очистить графический экран (стереть рисунок), то для этого используется процедура очистки экрана.

Заголовок процедуры очистки экрана:

```
Procedure ClearDevice;
```

В результате выполнения этой процедуры экран заполняется установленным цветом фона.

Графические координаты. Положение каждого пикселя графической сетки однозначно определяется указанием его координат.

Графические оси координат расположены на экране так, как показано на рис. 31.

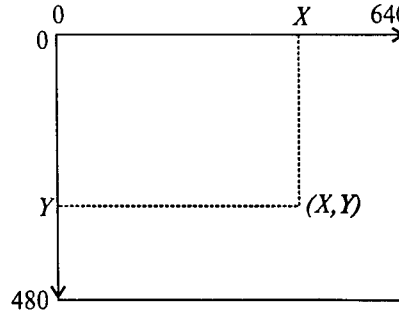


Рис. 31

Горизонтальная ось *X* направлена слева направо, вертикальная ось *Y* — сверху вниз. На рисунке указаны предельные графические координаты, соответствующие режиму VGAHi.

Можно определить максимальные координаты по осям, соответствующие данному драйверу. Это делается с помощью двух целочисленных функций:

```
Function GetMaxX;
Function GetMaxY;
```

Графическое окно. Область вывода изображения может быть ограничена любым прямоугольником в пределах экрана. Такая область называется графическим окном. Существует процедура, устанавливающая положение графического окна на экране.

Заголовок процедуры назначения графического окна:

```
Procedure SetViewPort(X1,Y1,X2,Y2: Integer;
                      Clip: Boolean);
```

Здесь `(X1,Y1)` — координаты левого верхнего угла окна; `(X2,Y2)` — координаты правого нижнего угла окна; `Clip` — ограничитель фигур; если `Clip=True`, то все построения производятся только в пределах окна, в противном случае они могут выходить за его пределы.

После установки окна координаты точек внутри него отсчитываются от верхнего левого угла.

Существует понятие графического курсора (по аналогии с символьным курсором). Но в отличие от символьного курсора графический курсор на экране не виден. Графический курсор указывает на текущую позицию на экране. При входе в графический режим координаты текущей позиции равны (0, 0).

Процедура назначения координат графического курсора:

```
Procedure MoveTo(X,Y: Integer);
```

Здесь `X, Y` — устанавливаемые координаты курсора. Координаты указываются относительно левого верхнего угла окна или, если окно не установлено, экрана.

Процедура *поставить точку* — основная процедура получения изображения, поскольку любой рисунок складывается из точек. Состояние светящейся точки определяется координатами точки на экране и цветом точки.

Заголовок процедуры выставления точки на графическом экране:

```
Procedure PutPixel(X,Y: Integer; Color: Word);
```

Здесь `X`, `Y` — координаты точки, `Color` — цвет точки.

Пример 1. Следующая программа устанавливает по центру экрана графическое окно размером 100×100, заливает его желтым фоном и заполняет синими точками, расположенными через 4 позиции.

```
Uses Graph;
Var Driver,Mode: Integer;
X,Y,X1,Y1,X2,Y2,Xc,Yc: Integer;
Begin
{Инициализация графического режима}
    Driver:=Detect;
    InitGraph(Driver, Mode,'C:\TP\BGI');
{Определение координат центра экрана}
    Xc:=GetMaxX Div 2;
    Yc:=GetMaxY Div 2;
{Определение координат графического окна}
    X1:=Xc-50;
    Y1:=Yc-50;
    X2:=Xc+50;
    Y2:=Yc+50;
{Установка графического окна}
    SetViewPort(X1,Y1,X2,Y2,True);
{Установка цвета фона и очистка экрана}
    SetBkColor(Yellow);
    ClearDevice;
{Расстановка точек в окне}
    For X:=1 To 25 Do
        For Y:=1 To 25 Do
        PutPixel(4*X,4*Y,Blue);
{Задержка изображения на экране до нажатия
<ENTER>}
    ReadLn;
{Выход из графического режима в символьный}
    CloseGraph;
End.
```

Графические примитивы. Хотя любое изображение можно построить из точек, но программировать получение сложного рисунка или чертежа, используя только процедуру *поставить точку*, было бы слишком неудобно и громоздко. В любом графическом пакете существуют процедуры рисования основных геометрических фигур: прямых линий, окружностей, эллипсов, прямоугольников и т. п. Такие фигуры называют *графическими примитивами*.

Рассмотрим несколько основных процедур рисования графических примитивов, имеющихся в модуле Graph.

Линия с заданными координатами концов (X1, Y1) и (X2, Y2):

```
Procedure Line(X1,Y1,X2,Y2: Integer);
```

Линия от текущей точки до точки с координатами X, Y:

```
Procedure LineTo(X,Y: Integer);
```

Линия от текущей точки до точки с заданными приращениями координат DX, DY:

```
Procedure LineRel(DX,DY: Integer);
```

Прямоугольник с заданными координатами верхнего левого угла (X1, Y1) и нижнего правого угла (X2, Y2):

```
Procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

Окружность с центром в точке (X,Y) и радиусом R — в пикселях:

```
Procedure Circle(X,Y: Integer; R: Word);
```

Дуга окружности с центром в точке (X, Y), радиусом R, начальным углом BegA и конечным углом EndA. Углы измеряются в градусах против часовой стрелки от направления оси *X*.

```
Procedure Arc(X,Y: Integer; BegA,EndA,R: Word);
```

Эллипсная дуга с центром в точке X, Y с начальным и конечным углами BegA и EndA, горизонтальным радиусом RX и вертикальным радиусом RY:

```
Procedure Ellipse(X,Y: Integer; BegA,EndA,RX,RY:
                  Word);
```

Пример 2. Составим программу, рисующую голову робота (рис. 32).

Рисунок содержит два прямоугольника, две окружности, две дуги, эллипс, три прямые линии и две красные точки. Заранее определяются все координаты и размеры элементов рисунка.

```
Uses Graph;
Var  Driver,Mode: Integer;
     Begin
     {Инициализация
     графического режима}
        Driver:=Detect;
        InitGraph(Driver,
        Mode,'C:\TP\BGI');
```

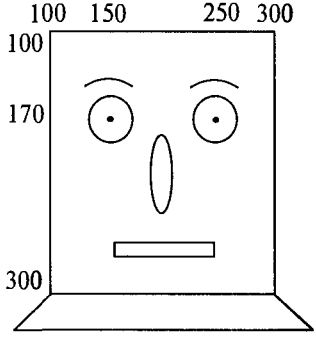


Рис. 32

```
    SetColor(White);{белый цвет рисунка}
    SetBkColor(Black);{черный цвет фона}
    Rectangle(100,100,300,300);{голова}
    Circle(150,170,30);{левый глаз}
    Circle(250,170,30);{правый глаз}
    Arc(150,170,45,135,40);{левая бровь}
    Arc(250,170,45,135,40);{правая бровь}
    Ellipse(200,250,0,359,10,20);{нос}
    Rectangle(130,280,270,290);{рот}
    MoveTo(100,300);{установка вниз влево}
    LineTo(50,350);{три}
    LineTo(350,350);{линии}
    LineTo(300,300);{шеи}
    PutPixel(150,170,Red);{левый зрачок}
    PutPixel(250,170,Red);{правый зрачок}
    ReadLn;{задержка}
    CloseGraph;{выход из графики}
End.
```

В приведенном примере все линии рисуются сплошными и стандартной толщины. Модуль `Graph` позволяет управлять стилем линии (сплошная, пунктирная, точечная и т.п.) и толщиной. Для этого существует процедура `SetLineStile` (см. приложение 2).

Закраски и заполнения. Среди графических примитивов существуют закрашенные области. Цвет закраски определяется процедурой `SetColor`. Кроме того, можно управлять рисунком закраски (типом заполнения). Это может быть сплошная закраска, заполнение редкими точками, крестиками, штрихами и т.п. В табл. П2.3 приложения 2 описаны константы для указания типа заполнения.

Процедура определения типа заполнения (`Fill`) *и цвета заполнения* (`Color`) имеет следующий заголовок:

```
Procedure SetFillStyle(Fill,Color: Word);
```

Заполненная прямоугольная область с заданными координатами углов:

```
Procedure Bar(X1,Y1,X2,Y2: Integer);
```

Обведенный линией (SetLineColor, SetLineStyle) *и закрашенный* (SetFillStyle) *эллипс*:

```
Procedure FillEllips(X,Y,RX,RY: Integer);
```

Обведенный линией и закрашенный эллипсный сектор:

```
Procedure Sector(X,Y: Integer;
                 BegA,EndA,RX,RY: Word);
```

Обведенный линией и закрашенный сектор окружности:

```
procedure PieSlice(X,Y:Integer; BegA,EndA: Word);
```

Наконец, можно закрасить любую область, ограниченную замкнутой линией. Для этого нужно указать какую-нибудь точку внутри этой области (X,Y) и цвет граничной линии (*Border*). Соответствующая процедура выглядит следующим образом:

```
Procedure FloodFill(X,Y: Integer; Border: Word);
```

Модуль `Graph` позволяет выводить на графический экран тексты. Мы не будем сейчас детально обсуждать эту проблему, необходимую информацию можно найти в соответствующей литературе. Приведем лишь пример одной текстовой процедуры, с помощью которой выводится в графическое окно символьная строка (`Txt`), начиная с указанной позиции (X,Y).

```
Procedure OutTextXY(X,Y: Integer; Txt: String);
```

Например, чтобы вывести под нашим рисунком строку «ЭТО РОБОТ», следует в программу добавить оператор

```
OutTextXY(195,400,'ЭТО РОБОТ');
```

Как построить график функции. Одним из приложений компьютерной графики является наглядное представление результатов математических расчетов. Графики функций, диаграммы, линии уровней распределения пространственных зависимостей и т. п. делают результаты расчетов обозримее, нагляднее, понятнее.

Мы рассмотрим лишь один простейший вариант математической графики — построение графика функции.

Требуется составить программу построения на экране дисплея графика функции

$$y = F(x).$$

Решение этой задачи удобно проводить в следующем порядке:

1. Определить границы значений аргумента, в пределах которых будет строиться график. Обозначим их следующим образом: X_{min} — нижняя граница, X_{max} — верхняя граница.

2. Для данной области значений аргумента определить предельные значения функции: Y_{min} и Y_{max}. Эти значения необязательно должны быть точными. Они могут быть оценочными снизу и сверху соответственно.

3. Задать границы графического окна, в пределах которого будет рисоваться график: $[Xg_{min}, Xg_{max}]$, $[Yg_{min}, Yg_{max}]$. Поскольку в графических координатах вертикальная ось направлена вниз, то $Yg_{min} > Yg_{max}$.

Таким образом, мы имеем две системы координат: (X, Y), которую назовем системой математических координат (в литературе

чаще используют термин «мировые координаты»), и (Xg, Yg) – систему графических координат. Нетрудно получить формулу, связывающую графические и математические координаты:

$$Xg = Xg_{\min} + \left[\frac{Xg_{\max} - Xg_{\min}}{X_{\max} - X_{\min}} (X - X_{\min}) \right];$$

$$Yg = Yg_{\min} + \left[\frac{Yg_{\max} - Yg_{\min}}{Y_{\max} - Y_{\min}} (Y - Y_{\min}) \right].$$

Здесь квадратные скобки означают округление до целого значения (функция `Round`).

Построение графика функции может производиться либо точечным методом, либо кусочно-линейным. При первом способе график строится как последовательность точек, расположенных максимально близко. Производится «попикселевый» перебор значений аргумента в интервале $[Xg_{\min}, Xg_{\max}]$ с выставлением точек с соответствующими координатами Y.

При кусочно-линейном методе задается шаг ΔX и рассчитывается последовательность значений (X_i, Y_i):

$$X_i = X_{\max} + i \cdot \Delta X, \quad Y_i = F(X_i),$$

$$i = 0, 1, \ldots, n, \quad n = \frac{X_{\max} - X_{\min}}{\Delta X}.$$

График строится в виде отрезков прямых, проведенных через точки (X_i, Y_i), (X_{i+1}, Y_{i+1}).

Пример 3. Составим программу построения графика функции

$$y = \sin x$$

для $x \in [0; 2\pi]$, используя первый (точечный) метод.

Из условия задачи следует, что $X_{\min} = 0$, $X_{\max} = 2\pi$. В этих пределах функция $\sin x$ меняется от -1 до 1. Поэтому $Y_{\min} = -1$, $Y_{\max} = 1$.

Выберем следующие границы графического окна:

$$Xg_{\min} = 10; \quad Xg_{\max} = 200;$$
$$Yg_{\min} = 140; \quad Yg_{\max} = 40.$$

График строится в виде последовательности точек с математическими координатами

$$X_i = X_{\min} + i \cdot h; \; Y_i = \sin(X_i); \; i = 0, \ldots, 190.$$

Шаг h выбирается минимально возможным, соответствующим шагу графической сетки:

$$h = \frac{X_{\max} - X_{\min}}{Xg_{\max} - Xg_{\min}} = \frac{2\pi}{190} = \frac{\pi}{95}.$$

Приведенные выше формулы перевода математических координат в графические примут вид:

$$Xg = 10 + \left[\frac{200 - 10}{2\pi} X\right] = 10 + \left[\frac{95}{\pi}\right];$$

$$Yg = 140 + \left[\frac{40 - 140}{2}(Y + 1)\right] = 90 - [50Y].$$

Вместе с графиком функции строятся оси координат. Ось X имеет координату $Yg = 90$, ось Y — координату $Xg = 10$.

```
Uses Graph;
Var Driver,Mode: Integer;
    X: Real; Xg,Yg,I: Integer;
Begin
{Инициализация графического режима}
    Driver:=Detect;
    InitGraph(Driver,Mode,'C:\TP\BGI');
    SetColor(White);{белый цвет линий}
    SetBkColor(Black);{черный цвет фона}
    Line(10,90,200,90);{ось X}
    Line(10,20,10,160);{ось Y}
{Построение графика функции желтыми точками}
    X:=0;
    For I:=0 To 190 Do
    Begin Xg:=10+Round(95/Pi*X);
      Yg:=90-Round(50*Sin(X));
      PutPixel(Xg,Yg,Yellow);
      X:=X+Pi/95
    End;
{Разметка осей, запись функции}
    OutTextXY(15,30,'Y');
    OutTextXY(205,90,'X');
    OutTextXY(130,40,'Y=SIN(X)');
    ReadLn;{задержка}
    CloseGraph; {выход из графики}
End.
```

Упражнения

1. Составить программу «Звездное небо»: в черном окне случайным образом появляются белые точки. Работа программы заканчивается по нажатию клавиши.

2. Изменить программу «Звездное небо» так, чтобы наряду с зажиганием новых звезд происходило угасание (закрашивание цветом фона) уже светящихся звезд.

3. В программу «Робот» внести такие изменения, в результате которых робот окажется раскрашенным в разные цвета.

4. Используя линии и другие графические примитивы, составить программу, рисующую дом.

5. Составить программу рисования на экране шахматного поля.

6. Написать универсальную процедуру построения графика функции $y = F(x)$ точечным методом. Процедура должна иметь следующие параметры: X_{min}, X_{max}, Y_{min}, Y_{max}, Xg_{min}, Xg_{max}, Yg_{min}, Yg_{max}. Функция $F(x)$ описывается во внешней подпрограмме-функции.

7. Исследовав область определения и выбрав расположение координатных осей, построить на экране графики функций:

$$1)\ y = \frac{1}{x+1};\quad 2)\ y = \frac{x+3}{x-2};\quad 3)\ y = 1 + \frac{2}{x} + \frac{3}{x^2},$$

используя процедуру из предыдущей задачи.

3.16. Строковый тип данных

Теперь мы познакомимся с типом данных, который относится к числу структурированных. Это строковый тип данных (строка). Следует заметить, что строковый тип данных есть в Турбо Паскале и отсутствует в стандартном Паскале.

Строка —это последовательность символов. Каждый символ занимает 1 байт памяти (код ASCII). Количество символов в строке называется ее длиной. Длина строки может находиться в диапазоне от 0 до 255. Строковые величины могут быть константами и переменными.

Строковая константа есть последовательность символов, заключенная в апострофы. Например:

```
'Язык программирования ПАСКАЛЬ',
'IBM PC — computer',
'33-45-12'.
```

Строковая переменная описывается в разделе описания переменных следующим образом:

```
Var <идентификатор>: String[<максимальная длина строки>]
```

Например:

```
Var Name: String[20]
```

Параметр длины может и не указываться в описании. В таком случае подразумевается, что он равен максимальной величине — 255. Например:

```
Var slovo: String
```

Строковая переменная занимает в памяти на 1 байт больше, чем указанная в описании длина. Дело в том, что один (нулевой) байт содержит значение *текущей длины строки*. Если строковой переменной не присвоено никакого значения, то ее текущая длина равна нулю. По мере заполнения строки символами ее текущая длина возрастает, но она не должна превышать максимальной по описанию величины.

Символы внутри строки индексируются (нумеруются) от единицы. Каждый отдельный символ идентифицируется именем строки с индексом, заключенным в квадратные скобки. Например:

```
Name[5], Name[i], slovo[k+1].
```

Индекс может быть положительной константой, переменной, выражением целого типа. Значение индекса не должно выходить за границы описания.

Тип `String` и стандартный тип `char` совместимы. Строки и символы могут употребляться в одних и тех же выражениях.

Строковые выражения строятся из строковых констант, переменных, функций и знаков операций. Над строковыми данными допустимы операции сцепления и операции отношения.

Операция сцепления (+) применяется для соединения нескольких строк в одну результирующую строку. Сцеплять можно как строковые константы, так и переменные.

Например:

```
'ЭВМ'+'IBM'+'PC'.
```

В результате получится строка:

```
'ЭВМ IBM PC'.
```

Длина результирующей строки не должна превышать 255.

Операции отношения =, <, >, <=, >=, <> производят сравнение двух строк, в результате чего получается логическая величина (`true` или `false`). Операция отношения имеет более низкий приоритет, чем операция сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и больше считается та строка, в которой первый несовпадающий символ имеет больший номер в таблице символьной кодировки.

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки равны, если они полностью совпадают по длине и содержат одни и те же символы.

Пример:

Выражение	Результат
`'cosm1'<'cosm2'`	`True`
`'pascal'>'PASCAL'`	`True`

```
'Ключ_'<>'Ключ'        True
'MS DOS'='MS DOS'      True
```

Функция `Copy (S,Poz,N)` выделяет из строки `S` подстроку длиной в `N` символов, начиная с позиции `Poz`. `N` и `Poz` — целочисленные выражения.

Пример:

Значение `S`	Выражение	Результат
`'ABCDEFG'`	`Copy(S,2,3)`	`'BCD'`
`'ABCDEFG'`	`Copy(S,4,4)`	`'DEFG'`

Функция `Concat(S1,S2,...,SN)` выполняет сцепление (конкатенацию) строк `S1,...,SN` в одну строку.

Пример:

Выражение	Результат
`Concat('AA','XX','Y')`	`'AAXXY'`

Функция `Length (S)` определяет текущую длину строки `S`. Результат — значение целого типа.

Пример:

Значение `S`	Выражение	Результат
`'test-5'`	`Length(S)`	6
`'(A+B)*C'`	`Length(S)`	7

Функция `Pos(S1,S2)` обнаруживает первое появление в строке `S2` подстроки `S1`. Результат — целое число, равное номеру позиции, где находится первый символ подстроки `S1`.

Если в строке `S2` подстроки `S1` не обнаружено, то результат равен `0`.

Пример:

Значение `S2`	Выражение	Результат
`'abcdef'`	`Pos('cd',S2)`	3
`'abcdcdef'`	`Pos('cd',S2)`	3
`'abcdef'`	`Pos('k',S2)`	0

Процедура `Delete(S,Poz,N)` выполняет удаление `N` символов из строки `S`, начиная с позиции `Poz`.

Пример:

Исходное значение `S`	Оператор	Конечное значение `S`
`'abcdefg'`	`Delete(S,3,2)`	`'abefg'`
`'abcdefg'`	`Delete(S,2,6)`	`'a'`

В результате выполнения процедуры уменьшается текущая длина строки в переменной `S`.

Процедура `Insert(S1,S2,Poz)` выполняет вставку строки `S1` в строку `S2`, начиная с позиции `Poz`.

Пример:

Начальное S2	Оператор	Конечное S2
'ЭВМ PC'	Insert('IBM-',S2,5)	'ЭВМ IBM-PC'
'Рис.2'	Insert('N',S2,6)	'Рис.N2'

Пример 1. Следующая программа получает из слова «ВЕЛИЧИНА» слово «НАЛИЧИЕ»:

```
Program Slovo_1;
Var S11,S12: String[10];
Begin
   S11:='ВЕЛИЧИНА';
   S12:=Copy(S11,7,2)+Copy(S11,3,4)+S11[2];
   WriteLn(S12)
End.
```

Пример 2. По следующей программе из слова «СТРОКА» будет получено слово «СЕТКА».

```
Program Slovo_2;
Var S1: String[10];
Begin
   S1:='СТРОКА';
   Delete(S1,3,2);
   Insert('E',S1,2);
   WriteLn(S1)
End.
```

Пример 3. Составим программу, которая формирует символьную строку, состоящую из n звездочек (n — целое число, $1 \le n \le 255$).

```
Program Stars;
Var A: String;
   N,I: Byte;
Begin
   Write('Введите число звездочек');
   ReadLn(N);
   A:='';
   For I:=1 To N Do
   A:=A+'*';
   WriteLn(A)
End.
```

Здесь строковой переменной A вначале присваивается значение пустой строки (''). Затем к ней присоединяются звездочки.

Пример 4. В символьной строке подсчитать количество цифр, предшествующих первому символу !.

```
Program C;
Var S: String;
    K,I: Byte;
Begin
   WriteLn(«Введите строку»);
   ReadLn(S);
   K:=0;
   I:=1;
While (I<=Length(S)) And (S[I]<>'!') Do
   Begin
       If (S[I]>='0') And (S[i]<='9')
       Then K:=K+1;
       I:=I+1
   End;
   WriteLn ('Количество цифр до символа
                «!» равно',K)
End.
```

В этой программе переменная K играет роль счетчика цифр, а переменная I — роль параметра цикла. Цикл закончит выполнение при первом же выходе на символ ! или, если в строке такого символа нет, при выходе на конец строки. Символ S[I] является цифрой, если истинно отношение: 0<S[I]<9.

Пример 5. Дана символьная строка, которая имеет следующий вид:

'a ⊕ b ='

На месте a и b стоят десятичные цифры; значком ⊕ обозначен один из знаков операций: +, -, *. Иначе говоря, записано арифметическое выражение с двумя однозначными числами и знак равенства, например '5 + 7 ='. Нужно, чтобы машина вычислила это выражение и после знака равенства вывела результат. Операция деления не рассматривается для того, чтобы иметь дело только с целыми числами.

Программу решения такой задачи назовем *интерпретатором*. Интерпретатор должен расшифровать содержание строки и выполнить соответствующую арифметическую операцию. От исходной символьной информации он должен перейти к работе с числовой информацией. Если предположить, что строка составлена только из четырех символов в соответствии с указанным форматом, то задача решается довольно просто. Вот программа такого интерпретатора:

```
Program Interpretator;
Var Str: String[4];
    A, B: 0..9;
```

```
C: -100..100;
Begin
   {ввод исходной строки}
   WriteLn('Введите выражение!');
   WriteLn;
   Read(Str);
   {преобразование цифровых символов в числа}
   A:=Ord(Str[1])-Ord('0');
   B:=Ord(Str[3])-Ord('0');
   {выполнение арифметической операции}
   Case Str[2] Of
        '+': C:=A+B;
        '-': C:=A-B;
        '*': C:=A*B
   End;
   {вывод результата}
   WriteLn(C:2)
End.
```

В этой программе появился новый для нас оператор. Он называется *оператором выбора*. Формат этого оператора описывается синтаксической диаграммой (рис. 33).

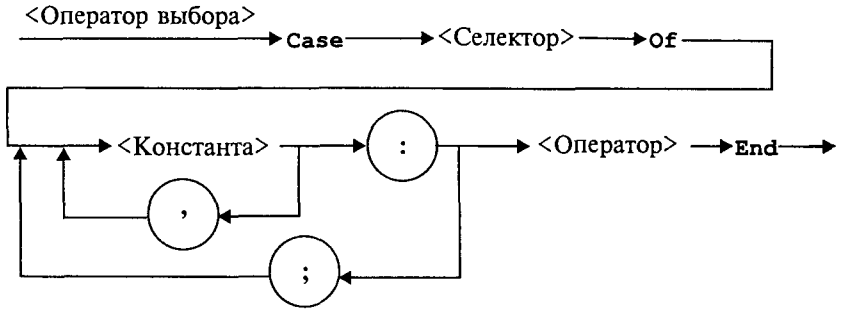


Рис. 33

Здесь <селектор> — это выражение любого *порядкового* типа; <константа> — постоянная величина того же типа, что и селектор; <оператор> — любой простой или составной оператор.

Выполнение оператора выбора происходит так: вычисляется выражение-селектор; затем в списках констант ищется такое значение, которое совпадает с полученным значением селектора; далее исполняется оператор, помеченный данной константой. Если такой константы не найдено, то происходит переход к выполнению оператора, следующего после оператора выбора.

В приведенной выше программе роль селектора играет символьная величина `Str[2]`. Если она равна +, то выполнится оператор

c:=a+b; если равна −, то выполнится оператор c:=a−b; если равна *, выполнится оператор c:=a*b. Для любых других значений Str[2] не выполнится ни один из операторов присваивания, и значение переменной c останется неопределенным.

Приведенное выше описание оператора выбора соответствует стандарту Паскаля. В Турбо Паскале допустимо использование в операторе **Case** альтернативной ветви после служебного слова **Else**. Вот пример для той же задачи. Если мы хотим, чтобы в случае неверного символа в Str[2] выдавалось сообщение об этом, нужно программировать так:

```
Case Str[2] Of
   '+': C:=A+B;
   '-': C:=A-B;
   '*': C:=A*B
Else WriteLn('неверный знак операции')
End;
```

А теперь вернемся к программе Interpretator и разберемся в том, как она будет выполняться.

После ввода строки цифровые символы переводятся в соответствующие десятичные числа. Затем интерпретируется знак операции. В зависимости от знака выполняется одно из трех арифметических действий. Далее результат выводится на экран после символа =.

Упражнения

1. Составить программу получения из слова «дисковод» слова «воск», используя операцию сцепления и функцию Copy.
2. Составить программу получения слова «правило» из слова «операция», используя процедуры Delete, Insert.
3. В данном слове заменить первый и последний символы на *.
4. В данном слове произвести обмен первого и последнего символов.
5. К данному слову присоединить столько !, сколько в нем имеется букв (например, из строки «УРА» получить «УРА!!!»).
6. В данной строке вставить пробел после каждого символа.
7. Удвоить все буквы во введенном слове.
8. Перевернуть введенную строку (например, из «ДИСК» получится «КСИД»).
9. В данной строке удалить все пробелы.
10. Строка представляет собой запись целого числа. Составить программу перевода ее в соответствующую величину целого типа.

3.17. Табличные данные и массивы

В повседневной и научной практике часто приходится встречаться с информацией, представленной в табличной форме. Вот,

например, таблица, содержащая среднемесячные значения температуры, °С, за определенный год:

Месяц	1	2	3	4	5	6	7	8	9	10	11	12
Температура	−21	−18	−7,5	5,6	10	18	22,2	24	17	5,4	−7	−18

Такую таблицу называют линсйной. Она представляет собой последовательность упорядоченных чисел. Если требуется какая-то математическая обработка этих данных, то для их обозначения обычно вводят индексную символику. Например, через T_1 обозначается температура января (первого месяца), T_5 — температура мая и т. д. В общем виде множество значений, содержащихся в таблице, обозначается так:

$$\{T_i\},\ i=1,\ldots,12.$$

Порядковые номера элементов (1, 5, i и т.д.) называются индексами. Индексированные величины удобно использовать для записи их математической обработки. Например, среднегодовая температура выражается следующей формулой:

$$T_{\text{ср}} = \frac{1}{12}\sum_{i=1}^{12} T_i.$$

Теперь представьте, что нам требуется собрать информацию о среднемесячных температурах за 10 лет, например с 1981 по 1990 г. Очевидно, что для этого удобна прямоугольная таблица, в которой столбцы соответствуют годам, а строки — месяцам.

Год	Месяц											
	1	2	3	4	5	6	7	8	9	10	11	12
1981	−23	−17	−8,4	6,5	14	18,6	25	19	12,3	5,6	−4,5	−19
1982	−16	−8,5	−3,2	7,1	8,4	13,8	28,5	21	6,5	2	−13	−20
1983	−9,8	−14	−9,2	4,6	15,6	21	17,8	20	11,2	8,1	−16	−21
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1990	−25	−9,7	−3,8	8,5	13,9	17,8	23,5	17,5	10	5,7	−14	−20

Обратите внимание на то, что уже сама удачно выбранная форма записи данных дает возможность достаточно удобно и быстро получить новую информацию. Например, из таблицы легко узнать, в каком году был самый холодный январь или какой месяц был самым нестабильным за десятилетие.

Для значений, хранящихся в такой таблице, удобно использовать двухиндексные обозначения. Например, $H_{1981,2}$ обозначает температуру в феврале 1981 г. и т.п. А вся совокупность данных, составляющих таблицу, обозначается так:

$\{H_{i,j}\}$, i=1981..1990; j=1..12

Обработка подобных данных производится с использованием двухиндексных величин. Например, средняя температура марта за 10 лет вычисляется по формуле:

$$H_{\text{ср. март}} = \frac{1}{10}\sum_{i=1981}^{1990} H_{i,3}.$$

А значение средней температуры за весь десятилетний период вычисляется так:

$$H_{\text{ср}} = \frac{1}{10}\sum_{i=1981}^{1990}\left[\frac{1}{12}\sum_{j=1}^{12} H_{i,j}\right] = \frac{1}{120}\sum_{i=1981}^{1990}\sum_{j=1}^{12} H_{i,j}.$$

В Паскале аналогом таблиц является структурированный тип данных, который называется *регулярным типом*, или *массивом*. Так же как и таблица, массив представляет собой совокупность пронумерованных однотипных значений, имеющих общее имя. Элементы массива обозначаются переменными с индексами. Индексы записывают в квадратных скобках после имени массива. Например:

`T[1],T[5],T[i],H[1981,9],H[i,j]` и т.п.

Массив, хранящий линейную таблицу, называется одномерным, прямоугольную таблицу — двумерным. В программах могут использоваться массивы и большей размерности.

Тип элементов массива называется его *базовым* типом. Очевидно, что для рассмотренных массивов температур базовым типом являтся вещественный (`Real`).

Описание массивов. Переменная регулярного типа описывается в разделе описания переменных в следующей форме:

Var <идентификатор>: **Array**[<тип индекса>] **Of** <тип компонент>

Чаще всего в качестве типа индекса употребляется интервальный тип. Например, рассмотренный выше одномерный массив среднемесячных температур опишется так:

```
Var T: Array[1..12] Of Real;
```

Описание массива определяет, во-первых, размещение массива в памяти, во-вторых, правила его дальнейшего употребления в программе. Последовательные элементы массива располагаются в последовательных ячейках памяти (`T[1]`, `T[2]` и т.д.), причем значения индекса не должны выходить из диапазона 1 ... 12. В качестве индекса может употребляться любое выражение соответствующего типа. Например, `T[i+j]`, `T[m div 2]`.

Тип индекса может быть любым скалярным порядковым типом, кроме integer. Например, в программе могут присутствовать следующие описания:

```
Var Cod: Array[Char] Of 1..100;
L: Array[Boolean] Of Char;
```

В такой программе допустимы следующие обозначения элементов массивов:

```
cod['x']; L[true]; cod[chr(65)]; L[a>0]
```

В некоторых случаях бывает удобно в качестве индекса использовать перечисляемый тип. Например, данные о количестве учеников в четырех десятых классах одной школы могут храниться в следующем массиве:

```
Type Index=(A,B,C,D);
Var Class_10: Array[Index] Of Byte;
```

И если, например, элемент Class_10[A] равен 35, то это означает, что в 10«А» классе 35 чел. Такое индексирование улучшает наглядность программы.

Часто структурированному типу присваивается имя в разделе типов, которое затем используется в разделе описания переменных.

```
Type Mas1=Array [1..100] Of Integer;
     Mas2=Array [-10..10] Of Char;
Var Num: Mas1; Sim: Mas2;
```

До сих пор речь шла об одномерных массивах, в которых типы элементов скалярные.

Многомерный массив в Паскале трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов). Например, рассмотренную выше прямоугольную таблицу можно хранить в массиве, описанном следующим образом:

```
Var H: Array[1981..1990] Of Array[1..12] Of Real;
```

Вот примеры обозначения некоторых элементов этого массива:

```
H[1981][1]; H[1985][10]; H[1990][12]
```

Однако чаще употребляется другая, эквивалентная форма обозначения элементов двумерного массива:

```
H[1981,1]; H[1985,10]; H[1990,12]
```

Переменная H[1981] обозначает всю первую строку таблицы, т.е. весь массив температур за 1981 г.

Другим вариантом, эквивалентным приведенному выше описанию, является следующий:

```
Type Month=Array[1..12] Of Real;
     Year=Array [1981..1990] Of Month;
Var H: Year;
```

Наиболее краткий вариант описания данного массива такой:

```
Var H: Array[1981..1990,1..12] Of Real;
```

Продолжая по аналогии, можно определить трехмерный массив как одномерный массив, у которого элементами являются двумерные массивы. Вот пример описания трехмерного массива:

```
Var A: Array[1..10,1..20,1..30] Of Integer;
```

Это массив, состоящий из $10 \cdot 20 \cdot 30 = 6000$ целых чисел и занимающий в памяти $6000 \cdot 2 = 12000$ байт. В Паскале нет ограничения сверху на размерность массива. Однако в каждой конкретной реализации Паскаля ограничивается объем памяти, выделяемый под массивы. В Турбо Паскале это ограничение составляет 64 килобайта.

По аналогии с математикой одномерные числовые массивы часто называют векторами, а двумерные — матрицами.

В Паскале не допускается употребление динамических массивов, т.е. таких, размер которых определяется в процессе выполнения. Изменение размеров массива происходит через изменение в тексте программы и повторную компиляцию. Для упрощения таких изменений удобно определять индексные параметры в разделе констант:

```
Const Imax=10; Jmax=20;
Var Mas: Array[1..Imax,1..Jmax] Of Integer;
```

Теперь для изменения размеров массива `Mas` и всех операторов программы, связанных с этими размерами, достаточно отредактировать только одну строку в программе — раздел констант.

Действия над массивом как единым целым. Такие действия допустимы лишь в двух случаях:

- присваивание значений одного массива другому;
- операции отношения «равно», «не равно».

В обоих случаях массивы должны иметь одинаковые типы (тип индексов и тип элементов).

Пример:

```
Var P,Q: Array[1..5,1..10] Of Real;
```

При выполнении операции присваивания `P:=Q` все элементы массива `P` станут равны соответствующим элементам массива `Q`.

Как уже отмечалось, в многомерных массивах переменная с индексом может обозначать целый массив. Например, если в таблице `H` требуется, чтобы данные за 1989 г. были такими же, как за

1981 г. (девятой строке присвоить значение первой строки), то это можно делать так:

```
H[1989]:=H[1981]
```

А если нужно поменять местами значения этих строк, то это делается через третью переменную того же типа:

```
P:=H[1989];H[1989]:=H[1981];H[1981]:=P;
```

где P описана так:

```
Var P: Array[1..12] Of Real;
```

Обработка массивов в программах производится покомпонентно. Вот примеры ввода значений в массивы:

```
For I:=1 To 12 Do
ReadLn(T[I]);
For I:=1 To IMax Do
For J:=1 To JMax Do
    ReadLn(Mas[I,J]);
```

Здесь каждое следующее значение будет вводиться с новой строки. Для построчного ввода используется оператор Read.

Аналогично в цикле по индексной переменной организуется вывод значений массива. Например:

```
For I:=1 To 12 Do Write(T[I]:8:4);
```

Следующий фрагмент программы организует построчный вывод матрицы на экран:

```
For I:=1 To IMax Do
Begin
For J:=1 To JMax Do
    Write(Mas[I,J]:6);
    WriteLn
End;
```

После печати очередной строки матрицы оператор WriteLn без параметров переведет курсор в начало новой строки. Следует заметить, что в последнем примере матрица на экране будет получена в естественной форме прямоугольной таблицы, если JMax не превышает 12 (сами подумайте почему).

Рассмотрим несколько примеров типовых программ обработки массивов.

Пример 1. Вернемся к массиву среднемесячных температур T[1..12]. Требуется вычислить среднегодовую температуру, а также ежемесячные отклонения от этой величины.

```
Program Example;
Const N = 12;
```

```
Type Vec=Array [1..N] Of Real;
Var T,Dt: Vec;
  St: Real;
  I: Integer;
Begin {Ввод исходных данных}
  WriteLn('Вводите таблицу температур');
  For I:=1 To N Do
     Begin
       Write(I: 2,':');
       ReadLn(T[I])
     End;
{Вычисление средней температуры}
     St:=0;
     For I:=1 To N Do
     St:=St+T[I];
     St:=St/N;
{Вычисление таблицы отклонений от среднего}
  For I:=1 To N Do
  Dt[I]:=T[I]-St;
  {Вывод результатов}
  WriteLn('Средняя температура равна',St:6:2);
  WriteLn;
  WriteLn('Отклонения от средней
           температуры:');
  For I:=1 To N Do
     WriteLn(I:1,':',Dt[I]:6:2)
End.
```

По этой программе можно рассчитать среднее значение и вектор отклонений от среднего для любого одномерного вещественного массива. Настройка на размер массива осуществляется только редактированием раздела констант.

Пример 2. Выбор максимального элемента. Пусть из рассмотренного выше массива температур требуется отобрать самую высокую температуру и номер месяца, ей соответствующего. Идея алгоритма решения этой задачи: чтобы получить максимальную температуру в вещественной переменной `TMax`, сначала в нее заносится первое значение массива `T[1]`. Затем поочередно сравнивается значение `TMax` с остальными элементами массива температур, и каждое значение большее, чем `TMax`, присваивается этой переменной. Для получения номера самого теплого месяца в целой переменной `NumMax` в нее следует каждый раз засылать номер элемента массива температур одновременно с занесением в `TMax` его значения.

```
TMax:=T[1];
NumMax:=1;
```

```
For I:=2 To 12 Do
  If T[I]>Tmax
  Then
  Begin
    TMax:=T[I];
    NumMax:=I
 End;
```

Заметим, что если в массиве температур несколько значений, равных максимальному, то в NumMax будет получен первый номер из этих элементов. Чтобы получить последнюю дату, нужно в операторе **If** заменить знак отношения > на >=.

Пример 3. Сортировка массива. В одномерном массиве X из N элсментов требуется произвести перестановку значений так, чтобы они расположились по возрастанию, т.е. $X_1 \le X_2 \le \ldots \le X_N$.

Существует целый класс алгоритмов сортировки. Ниже описан алгоритм, который называется «метод пузырька».

Идея: производится последовательное упорядочивание смежных пар элементов массива: X_1 и X_2, X_2 и X_3, ..., X_{N-1} и X_N. В итоге максимальное значение переместится в X_N. Затем ту же процедуру повторяют до X_{N-1} и т.д., вплоть до цепочки из двух элементов X_1 и X_2. Такой алгоритм будет иметь структуру двух вложенных циклов с внутренним циклом — переменной (сокращающейся) длины.

```
For I:=1 To N-1 Do
For K:=1 To N-I Do
   If X[K]>X [K+1]
   Then
   Begin
      A:=X[K];
      X[K]:=X[K+1];
      X[K+1]:=A
   End;
```

Пример 4. Дан описанный выше двумерный массив среднемесячных температур за 10 лет. Определить, в каком году было самое теплое лето, т.е. в каком году была наибольшая средняя температура летних месяцев.

Идея решения: в векторе S получить средние температуры летних месяцев за 10 лет. Затем найти номер наибольшего элемента в этом векторе, это и будет искомый год.

```
Program Example_2;
Type Month=Array[1..12] Of Real;
     Year=Array[1981..1990] Of Month;
Var H: Year;
    S: Array[1981..1990] Of Real;
    I,J,K: Integer;
```

```
Begin {Ввод данных с клавиатуры}
   For I:=1981 To 1990 Do
     For J:=1 To 12 Do
       Begin
        Write(J:2,'.',I:4,':');
        ReadLn(H[I,J])
       End;
{Вычисление вектора средних летних
температур}
   For I:=1981 To 1990 Do
      Begin
       S[I]:=0;
       For J:=6 To 8 Do
          S[I]:=S[I]+H[I,J];
       S[I]:=S[I]/3
      End;
{Определение года с самым теплым летом}
   K:=1981;
   For I:=1982 To 1990 Do
     If S[I]>S[K] Then K:=I;
     WriteLn('Самое теплое лето было в',
              K,'-м году')
End.
```

Упражнения

1. Дан вектор $\{z_i\}$, $i = 1, ..., 50$. Вычислить длину этого вектора:

$$L = \sqrt{z_1^2 + z_2^2 + ... + z_{50}^2}\,.$$

2. Вычислить полином 10-й степени по формуле Горнера:

$$a_{10}x^{10} + a_9x^9 + ... + a_1x + a_0 = ((...(a_{10}x + a_9)x + a_8)x + ... + a_1)x + a_0.$$

3. Для вектора $\{x_i\}$, $i = 1, ..., 20$, подсчитать количество компонент, значения которых лежат в интервале [0,1].

4. Даны два вектора $\{x_i\}$, $\{y_i\}$, $i = 1, ..., 10$, упорядоченные по возрастанию. Слить их в один вектор $\{z_i\}$, $i = 1, ..., 20$, так чтобы сохранилась упорядоченность.

5. Дан массив, состоящий из 100 целых чисел. Вывести все числа, которые встречаются в этом массиве:

а) несколько раз;

б) только по одному разу.

6. В целочисленной матрице размером 10×10 найти значение и индексы максимального элемента.

7. В двоичной матрице 5×10 определить номер строки с наибольшим количеством нулей.

8. Все строки вещественной матрицы 10×15 упорядочить по убыванию значений их элементов.

9. Транспонировать целочисленную матрицу 5×5, т.е. отразить относительно главной диагонали.

10. В двоичной матрице 10×10 найти совпадающие строки.

3.18. Понятие множества. Множественный тип данных

Одним из фундаментальных разделов математики является теория множеств. Некоторые моменты математического аппарата этой теории реализованы в Паскале через *множественный тип данных* (множества).

Множеством называется совокупность однотипных элементов, рассматриваемых как единое целое. В Паскале могут быть только конечные множества. В Турбо Паскале множество может содержать от 0 до 255 элементов.

В отличие от элементов массива элементы множества не пронумерованы, не упорядочены. Каждый отдельный элемент множества не идентифицируется, и с ним нельзя выполнить какие-либо действия. Действия могут выполняться только над множеством в целом.

Тип элементов множества называется базовым типом. Базовый тип может быть любым скалярным, за исключением типа `Real`.

Конструктор множества. Конкретные значения множества задаются с помощью конструктора множества, представляющего собой список элементов, заключенный в квадратные скобки. Сами элементы могут быть либо константами, либо выражениями базового типа. Вот несколько примеров задания множеств с помощью конструктора:

`[3,4,7,9,12]` — множество из пяти целых чисел;

`[1..100]` — множество целых чисел от 1 до 100;

`['a','b','c']` — множество, содержащее три литеры *a*, *b*, *c*;

`['A'..'Z','?','!']` — множество, содержащее все прописные латинские буквы, а также знаки ? и !.

Символы [] обозначают пустое множество, т.е. множество, не содержащее никаких элементов.

Не имеет значения порядок записи элементов множества внутри конструктора. Например, `[1,2,3]` и `[3,2,1]`эквивалентные множества.

Каждый элемент в множестве учитывается только один раз. Поэтому множество `[1,2,3,4,2,3,4,5]` эквивалентно `[1..5]`.

Переменные множественного типа описываются так:

```
Var <идентификатор>: Set Of <базовый тип>
```

Например:

```
Var A,D: Set Of Byte;
```

```
B: Set Of 'a'..'z';
C: Set Of Boolean;
```

Нельзя вводить значения во множественную переменную оператором ввода и выводить оператором вывода. Множественная переменная может получить конкретное значение только в результате выполнения оператора присваивания следующего формата:

```
<множественная переменная>:=<множественное выражение>
```

Например:

```
A:=[50,100,150,200];
B:=['m','n','k'];
C:=[True,False];
D:=A;
```

Кроме того, выражения могут включать в себя операции над множествами.

Операции над множествами. В Паскале реализованы основные операции теории множеств. Это *объединение*, *пересечение*, *разность* множеств. Во всех таких операциях операнды и результаты есть множественные величины одинакового базового типа.

Объединение множеств. Объединением двух множеств *A* и *B* называется множество, состоящее из всех элементов, принадлежащих хотя бы одному из множеств *A* или *B*. Знак операции объединения в Паскале +.

На рис. 34, *а* схематически показан результат объединения двух множеств.

Например:

```
[1,2,3,4]+[3,4,5,6]→[1,2,3,4,5,6]
```

Пересечение множеств. Пересечением двух множеств *A* и *B* называется множество, состоящее из всех элементов принадлежащих, одновременно множеству *A* и множеству *B* (см. рис. 34, *б*) Знак операции пересечения в Паскале *.

Например:

```
[1,2,3,4]*[3,4,5,6]→[3,4]
```

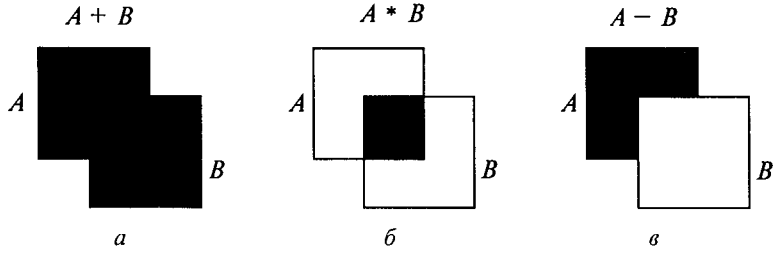
Разность множеств. Разностью двух множеств *A* и *B* называется множество, состоящее из элементов множества *A*, не принадлежащих множеству *B* (см. рис. 34, *в*).

Например:

```
[1,2,3,4]-[3,4,5,6]→[1,2]
[3,4,5,6]-[1,2,3,4]→[5,6]
```

Очевидно, что операции объединения и пересечения — перестановочны, а разность множеств — не перестановочная операция.

Операции отношения. Множества можно сравнивать между собой, т.е. для них определены операции отношения. Результатом



а *б* *в*

Рис. 34

отношения, как известно, является логическая величина *true* или *false*. Для множеств применимы все операции отношения, за исключением > и <. В таблице описаны операции отношения над множествами. Предполагается, что множества *A* и *B* содержат элементы одного типа.

Отношение	Результат	
	True	*False*
A = *B*	Множества *A* и *B* совпадают	В противном случае
A <> *B*	Множества *A* и *B* не совпадают	В противном случае
A <= *B*	Все элементы *A* принадлежат *B*	В противном случае
A >= *B*	Все элементы *B* принадлежат *A*	В противном случае

Вот несколько примеров использования операций отношения. Пусть переменная `M` описана в программе следующим образом:

```
Var M: Set Of Byte;
```

В разделе операторов ей присваивается значение:

```
M:=[3,4,7,9];
```

Тогда операции отношения дадут следующие результаты:

```
M=[4,7,3,3,9]          - true,
M<>[7,4,3,9]           - false,
[3,4]<=M               - true,
[]<=M                  - true,
M>=[1..10]             - false,
M<=[3..9]              - true.
```

Операция вхождения. Это операция, устанавливающая связь между множеством и скалярной величиной, тип которой совпадает с базовым типом множества. Если `x` — такая скалярная величина, а `M` — множество, то операция вхождения записывается так:

```
x In M
```

Результат — логическая величина true, если значение x входит в множество M, и false — в противном случае. Для описанного выше множества

```
4 In M — true,
5 In M — false.
```

Рассмотрим несколько задач, для решения которых удобно использовать множества.

Пример 1. Дана символьная строка. Подсчитать в ней количество знаков препинания (. – , ; : ! * ?).

```
Program P1;
Var S: String; I,K: Byte;
Begin
    ReadLn(S); K:=0;
    For I:=1 To Length(S) Do
     If S[I] In ['.','-',',',';',':','!','*','?']
     Then K:=K+1;
WriteLn('Число знаков препинания равно',K)
End.
```

В этом примере использована множественная константа с символьным типом элементов. Эту задачу можно решить и без множества, записав в операторе If длинное логическое выражение: (S[I]='.') Or (S[I]='–') и т.д. Использование множества сокращает запись.

Пример 2. Даны две символьные строки, содержащие только строчные латинские буквы. Построить строку S3, в которую войдут только общие символы S1 и S2 в алфавитном порядке и без повторений.

```
Program P2;
Type Mset=Set Of 'a'..'z';
Var S1,S2,S3: String;
    MS1,MS2,MS3: Mset;
    C: Char;
Procedure SM(S: String; Var MS: Mset);
{Процедура формирует множество MS, содержащее все
символы строки S}
Var I: Byte;
Begin  MS:=[];
  For I:=1 To Length(S) Do
      MS:=MS+[S[I]]
End;
Begin {Ввод исходных строк}
  ReadLn(S1);ReadLn(S2);
{Формирование множеств MS1 и MS2 из символов строк
S1 и S2}
  SM(S1,MS1);SM(S2,MS2);
```

```
{Пересечение множеств - выделение общих элементов в
множество MS3}
  MS3:=MS1*MS2;
{Формирование результирующей строки S3}
  S3:='';
  For C:='a' To 'z' Do
  If C In MS3 Then S3:=S3+C;
  WriteLn('Результат:',S3)
End.
```

Пример 3. Составить программу, по которой из последовательности натуральных чисел от 2 до N ($1 < N \le 255$) будут выбраны все простые числа.

Для решения задач такого типа существует алгоритм, известный под названием «Решето Эратосфена». Суть его в следующем:

1. Из числовой последовательности выбираем минимальное значение, это будет простое число.
2. Удаляем из последовательности все числа, кратные выбранному.
3. Если после удаления последовательность не стала пустой, то возвращаемся к выполнению пункта 1.

Вот пример работы такого алгоритма для $N = 15$ (подчеркнуты выбранные простые числа):

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15
3 5 7 9 11 13 15
5 7 11 13
7 11 13
11 13
13
```

Решение этой задачи удобно программировать, используя множественный тип.

```
Program Eratosfen;
Const N=201;
{Возможно любое значение 1<N<256}
Var A,B: Set Of 2..N;
K,P: Integer;
Begin
    {Формирование исходного множества A;
    B - искомое множество простых чисел,
    сначала - пустое}
    A:=[2..N]; B:=[]; P:=2;
    Repeat
      {Поиск минимального числа в множестве A}
      While Not(P In A) Do P:=P+1;
      {Включение найденного числа в множество B}
```

```
        B:=B+[P];
        K:=P;
        {Исключение из A чисел, кратных P}
        While K<=N Do
        Begin
          A:=A-[K];
          K:=K+P;
        End
      Until A=[];
{Вывод результата, т.е. всех чисел из множества B в
порядке возрастания}
    For P:=2 To N Do If P In B Then WriteLn(P)
End.
```

Красивая программа! К сожалению, ею нельзя воспользоваться для $N > 255$ из-за отмеченного выше ограничения на максимальный размер множества в Турбо Паскале.

Пример 4. Как уже говорилось, нельзя вводить значения непосредственно в множество. Однако такая потребность у программиста может возникнуть. В этом случае можно прибегнуть к процедуре INSET, описанной ниже. Для примера рассматривается множество с символьным базовым типом. Предполагается, что в основной программе глобально объявлен тип SetChar.

```
Type SetChar: Set Of Char;
Procedure INSET(Var M: SetChar);
Var I,N: Byte; C: Char;
Begin
   Write('Укажите размер множества:'); ReadLn(N);
   M:=[];
   For I:=1 To N Do
   Begin
      Write(I:1,'-й элемент:'); ReadLn(C);
      M:=M+[C]
   End;
   WriteLn('Ввод завершен!')
End.
```

В основной программе для ввода значений в множество, например с именем SIM, достаточно записать оператор: INSET(SIM); Произойдет диалоговый ввод значений.

Упражнения

1. В программе присутствуют описания:

```
Var P: Set Of 0..9; I,J: Integer;
```

Если $i=3$ и $j=5$, то какое значение получит переменная P при выполнении следующих операторов присваивания:

а) `P:=[I+3,J Div 2,J..Sqr(I)-3];`
б) `P:=[2*I..J];`
в) `P:=[I,J,2*I,2*J]`?

2. Вычислить значения отношений:

а) `[2]<>[2,2,2];`
б) `['a','b']=['b','a'];`
в) `[4,5,6]=[4,5,6];`
г) `['c','b']=['c'..'b'];`
д) `[2,3,5,7]<=[1..9];`
е) `[3,6..8]<=[2..7,9];`
ж) `Trunc(3.9)In[1,3,5];`
з) `Odd(4) In[].`

3. Вычислить значения выражений:

а) `[1,3,5]+[2,4];`
б) `[1,3,5]*[2,4];`
в) `[1,3,5]-[2,4];`
г) `[1..6]+[3..8];`
д) `[1..6]*[3..8];`
е) `[1..6]-[3..8];`
ж) `[]+[4];`
з) `[]*[4];`
и) `[]-[4].`

4. Составить программу подсчета количества различных значащих цифр в десятичной записи натурального числа.

5. Составить программу, печатающую в возрастающем порядке все целые числа из диапазона 1...255, представимые в виде n^2+m^2, где n, $m>0$.

6. Дана строка из строчных русских букв. Между соседними словами — запятая, за последним словом — точка. Напечатать в алфавитном порядке:

а) все гласные буквы, которые входят в каждое слово;
б) все согласные буквы, которые не входят ни в одно слово;
в) все согласные буквы, которые входят только в одно слово;
г) все гласные буквы, которые входят более чем в одно слово.

3.19. Файлы. Файловые переменные

С термином «файл» вам уже приходилось встречаться. Прежде всего это понятие обычно связывают с информацией на устройствах внешней памяти. В Паскале понятие файла употребляется в двух смыслах:

- как поименованная информация на внешнем устройстве (внешний файл);
- как переменная файлового типа в Паскаль-программе (внутренний файл).

В программе между этими объектами устанавливается связь. Вследствие этого все, что происходит в процессе выполнения программы с внутренним файлом, дублируется во внешнем фай-

ле. С элементами файла можно выполнять только две операции: читать из файла и записывать в файл.

Файловый тип переменной — это структурированный тип, представляющий собой совокупность однотипных элементов, количество которых заранее (до исполнения программы) не определено.

Структура описания файловой переменной:

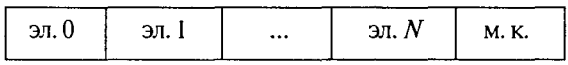
```
Var <имя переменной>: File Of <тип элемента>;
```

где `<тип элемента>` может быть любым, кроме файлового.

Например:

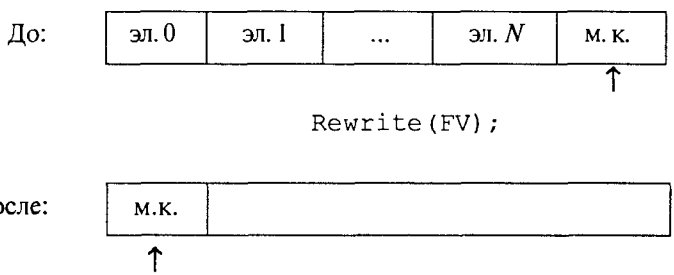
```
Var Fi: File Of Integer;
    Fr: File Of Real;
    Fc: File Of Char;
```

Файл можно представить как последовательную цепочку элементов (эл.), пронумерованных от 0, заканчивающуюся специальным кодом, называемым *маркером конца* (<м. к.>):



Количество элементов, хранящихся в данный момент в файле, называется его *текущей длиной*. Существует специальная ячейка памяти, которая хранит адрес элемента файла, предназначенного для текущей обработки (записи или чтения). Этот адрес называется *указателем* или *окном файла*.

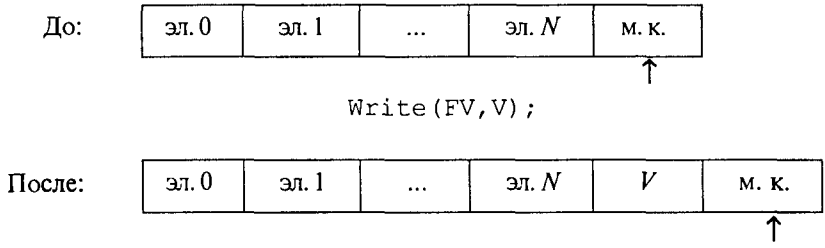
Для того чтобы начать запись в файл, его следует *открыть для записи*. Это обеспечивает процедура `Rewrite(FV);` где `FV` — имя файловой переменной. При этом указатель устанавливается на начало файла. Если в файле есть информация, то она исчезает. Схематически выполнение процедуры `Rewrite` можно представить так:



Стрелка внизу отмечает позицию указателя.

Запись в файл осуществляется процедурой `Write(FV,V);` где `V` — переменная того же типа, что и файл `FV`. Запись происходит туда, где установлено окно (указатель). Сначала записывается значение,

затем указатель смещается в следующую позицию. Если новый элемент вносится в конец файла, то сдвигается маркер конца. Схема выполнения оператора:

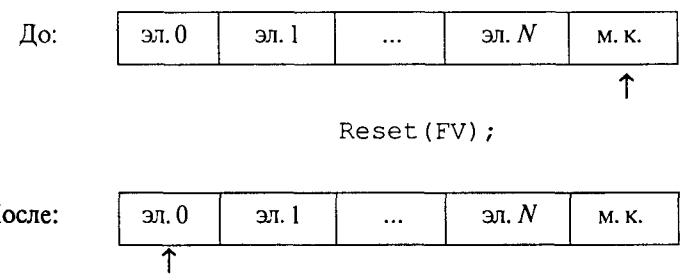


Пример 1. В файловую переменную `Fx` занести 20 вещественных чисел, последовательно вводимых с клавиатуры.

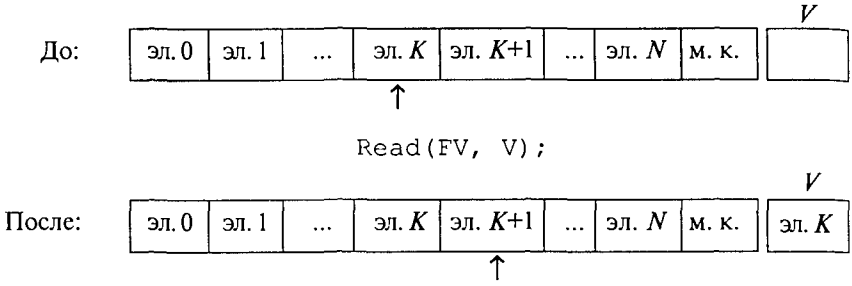
```
Var Fx: File Of Real;
    X: Real; I: Byte;
Begin
  Rewrite(Fx);
  For I:=1 To 20 Do
  Begin
    Write('?'); ReadLn(X);
    Write(Fx,X)
  End
End.
```

Для чтения элементов файла с его начала следует *открыть файл для чтения*. Это делает процедура `Reset(FV)`.

В результате указатель устанавливается на начало файла. При этом вся информация в файле сохраняется. Схема выполнения процедуры:



Чтение из файла осуществляется процедурой `Read(FV,V);` где `V` — переменная того же типа, что и файл `FV`. Значение текущего элемента файла записывается в переменную `V`; указатель смещается к следующему элементу.



Доступ к элементам файла может быть *последовательным* или *прямым*. В стандартном Паскале реализован только последовательный доступ.

Принцип последовательного доступа: для того чтобы прочитать n-ю запись файла, сначала нужно прочитать все предыдущие записи с 1-й по (n–1)-ю.

Пример 2. В переменной `x` получить 10-й элемент вещественного файла `Fx`.

```
Program A;
Var Fx: File Of Real;
    X: Real;
Begin
  Reset(Fx);
  For I:=1 To 10 Do Read(Fx,X)
End.
```

Функция **`Eof(FV)`** проверяет маркер конца файла (*end of file*). Это логическая функция, которая получает значение `true`, если указатель установлен на маркер конца, в противном случае — `false`.

Пример 3. Просуммировать все числа из файла `Fx`, описанного в предыдущем примере.

```
Reset(Fx);
Sx:=0;
While Not Eof(Fx) Do
Begin
    Read(Fx,X);
    Sx:=Sx+X
End;
```

То же самое с помощью цикла `Repeat` можно делать следующим образом:

```
Repeat
    Read(Fx,X);
    Sx:=Sx+X
Until Eof(Fx);
```

Во втором варианте возможна ошибка чтения, если файл Fx пустой. Первый вариант от такой ошибки застрахован, поэтому он более предпочтителен.

Внешние файлы. В Турбо Паскале все внешние устройства (дисплей, клавиатура, принтер, диски и т.д.) трактуются как логические устройства с файловой структурой организации данных. Все немагнитные внешние устройства однофайловые. Иначе говоря, с каждым из них связан один файл со стандартным именем, предназначенный для обмена с внутренней памятью ЭВМ текстовой (символьной) информацией.

Стандартные имена логических устройств определяются операционной системой, в среде которой работает Паскаль. В системе MS DOS определены следующие имена:

CON (консоль) — логическое устройство, связанное при вводе с клавиатурой, при выводе — с экраном;

PRN (принтер) — логическое имя файла, связанного с устройством печати;

AUX — логическое имя коммуникационного канала, который используется для связи ПК с другими машинами;

INPUT — логическое имя стандартного устройства ввода, связанного с клавиатурой; при этом вводимые с клавиатуры символы отражаются на экране дисплея;

OUTPUT — логическое имя стандартного устройства вывода на экран.

Магнитный диск (МД) — многофайловое устройство. На нем хранятся как стандартные (системные) файлы, так и файлы, создаваемые пользователем. На магнитном диске могут создаваться файлы любых типов. Файлы на МД используются как в режиме чтения, так и в режиме записи.

Список файлов на диске хранится в директории (каталоге) диска. Каталог вызывается на экран системной командой DIR. В полной форме каталог содержит идентификаторы файлов, объем занимаемой памяти, дату и время создания файла. Идентификатор файла состоит из имени и типа файла:

```
<имя файла>.<тип файла>
```

Имя содержит от 1 до 8 латинских букв и (или) цифр; тип — необязательный элемент (от 0 до 3 символов), указывающий на характер информации, хранимой в файле.

Например:

```
PROGRAM.PAS — в файле текст программы на Паскале;
NUMBER.DAT — файл числовых данных;
NAMES.TXT — текстовый файл.
```

Для организации связи между файловой переменной и внешним файлом в Турбо Паскале используется *процедура назначения*:

```
Assign(<имя файловой переменной>,
       <идентификатор внешнего файла>);
```

Здесь <идентификатор внешнего файла> — строковая величина (константа или переменная). Например:

```
Assign(Fi,'Number.dat');
```

После выполнения процедур Assign и Rewrite создается новый внешний файл, имя которого заносится в директорию.

Если файл открывается для чтения (Assign и Reset), то в указанном каталоге уже должен содержаться указанный внешний файл. В противном случае будет обнаружена ошибка.

Работа с файлом в программе завершается его *закрытием* с помощью процедуры

```
Close(<имя файловой переменной>)
```

Например:

```
Close(Fi)
```

Подведем итог сказанному. Для создания и заполнения файла требуется следующая последовательность действий:

1. Описать файловую переменную.
2. Описать переменную того же типа, что и файл.
3. Произвести назначение (Assign).
4. Открыть файл для записи (Rewrite).
5. Записать в файл данные (Write).
6. Закрыть файл (Close).

Пример 4. Создать файл, содержащий среднесуточные температуры за некоторое количество дней. При этом необязательно предварительно указывать количество чисел во вводимой информации. Можно договориться о каком-то условном значении, которое будет признаком конца ввода. Пусть, например, признаком конца ввода будет число 9999.

```
Program Task1;
Var Ft: File Of Real; T: Real;
    Begin
      Assign(Ft,'Temp.dat'); Rewrite(Ft);
      WriteLn('Вводите данные. Признак конца -
               9999');
      ReadLn(T);
      While T<>9999 Do
        Begin
            Write(Ft,T); Write('?'); ReadLn(T)
        End;
      WriteLn('Ввод данных закончен!');
      Close(Ft)
    End.
```

В результате работы этой программы на диске будет создан файл с именем Temp.dat, в котором сохранится введенная информация.

Для последовательного чтения данных из файла требуется выполнить следующие действия:

1. Описать файловую переменную.
2. Описать переменную того же типа.
3. Выполнить назначение (Assign).
4. Открыть файл для чтения (Reset).
5. В цикле читать из файла (Read).
6. Закрыть файл (Close).

Пример 5. Определить среднюю температуру для значений, хранящихся в файле Temp.dat.

```
Program Task2;
Var Ft: File Of Real;
    T,St: Real; N: Integer;
Begin Assign(Ft,'Temp.dat');
   Reset(Ft);
   St:=0;
   While Not Eof(Ft) Do
   Begin
     Read(Ft,T);
     St:=St+T
   End;
   N:=FileSize(Ft);
   St:=St/N;
   WriteLn('Средняя температура за',N:3,
           'суток равна',St:7:2,'гр-в');
   Close(Ft)
End.
```

В этой программе использована функция определения размера файла:

```
FileSize(<имя файловой переменной>);
```

Ее результат — целое число, равное текущей длине файла.

Замечание: согласно стандарту Паскаля в файл, открытый оператором Rewrite, можно только записывать информацию, а файл, открытый оператором Reset, можно использовать только для чтения. В Турбо Паскале допускается запись (Write) в файл, открытый для чтения (Reset). Это создает определенные удобства для модификации файлов.

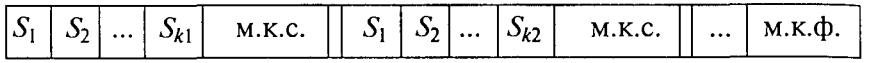
Текстовые файлы. Текстовый файл — наиболее часто употребляемая разновидность файлов. Как уже отмечалось раньше, немагнитные внешние устройства (логические) работают только с текстовыми файлами. Файлы, содержащие тексты программ на Пас-

кале и других языках программирования, являются текстовыми. Различная документация, информация, передаваемая по каналам электронной связи, — все это текстовые файлы.

В программе файловая переменная текстового типа описывается следующим образом:

```
Var <идентификатор>:text;
```

Текстовый файл представляет собой символьную последовательность, разделенную на строки. Каждая строка заканчивается специальным кодом — маркером конца строки (м.к.с.). Весь файл заканчивается маркером конца файла (м.к.ф.). Схематически это выглядит так:



Каждый символ представлен во внутреннем коде (ASCII) и занимает 1 байт. Текстовый файл отличается от символьного не только делением на строки. *В текстовой файл можно записать и из него прочитать информацию любого типа.* Если эта информация несимвольная, то в процессе чтения или записи происходит ее преобразование из символьной формы во внутреннюю и обратно.

Текстовый файл можно создать или преобразовать с помощью текстового редактора. Его можно просмотреть на экране дисплея или распечатать на принтере.

В программах на Паскале для работы с текстовыми файлами наряду с процедурами `Read` и `Write` употребляются процедуры `ReadLn` и `WriteLn`.

```
ReadLn(FV,<список ввода>)
```

Эта процедура читает строку из файла с именем `FV`, помещая прочитанное в переменные из списка ввода.

```
WriteLn(FV,<список вывода>)
```

Процедура записывает в файл `FV` значения из списка вывода, после чего выставляет маркер конца строки.

Для обнаружения конца строки в текстовом файле используется функция

```
Eoln(FV)
```

(*End of line — конец строки*). Это логическая функция, которая принимает значение `true`, если указатель файла достиг маркера конца строки и `false` — в противном случае.

Употребление операторов `Read` и `ReadLn` без указания имени файловой переменной обозначает чтение из стандартного файла

Input (ввод с клавиатуры). Употребление операторов Write и WriteLn без имени файловой переменной обозначает запись в стандартный файл Output (вывод на экран). В таком варианте этими операторами мы уже многократно пользовались. Считается, что файлы Input и Output всегда открываются соответственно для чтения и записи при работе любой программы.

При вводе с клавиатуры маркер конца строки обнаруживается при нажатии на клавишу **Enter**.

Процедура ReadLn может использоваться без списка ввода. В этом случае происходит пропуск текущей строки в читаемом файле.

Употребление процедуры WriteLn без списка вывода обозначает вывод пустой строки (в файле выставляется маркер конца строки).

При записи в текстовый файл в списке вывода могут присутствовать форматы. Действия форматов мы уже рассматривали при обсуждении вывода данных на экран. Точно так же форматы работают и при выводе в текстовые файлы, связанные с любыми другими устройствами.

Пример 6. Пусть файл с именем Note.txt содержит некоторый текст. Требуется подсчитать количество строк в этом тексте.

```
Var Note: Text; K: Integer;
Begin Assign(Note,'Note.txt');
      Reset(Note);
      K:=0;
      While Not Eof(Note) Do
      Begin
        ReadLn(Note); K:=K+1
      End;
      WriteLn('Количество строк равно',K);
      Close(Note)
End.
```

Используемый здесь оператор ReadLn(Note) «пролистывает» строки из текстового файла Note, не занося их в какую-либо переменную.

Пример 7. В текстовом файле Note.txt определить длину самой большой строки.

```
Var Note: Text;
    Max,K: Integer; C: Char;
Begin
   Assign(Note,'Note.txt');
   Reset(Note);
   Max:=0;
   While Not Eof(Note) Do
   Begin
     K:=0;
     While Not Eoln(Note) Do
     Begin
```

```
            Read(Note,C); K:=K+1
          End;
          If K>Max Then Max:=K;
          ReadLn(Note)
        End;
          WriteLn('Наибольшая строка имеет',
                  Max,'знаков');
          Close(Note)
    End.
```

Здесь каждая строчка прочитывается посимвольно, при этом в переменной K работает счетчик числа символов в строке. В переменной Max отбирается наибольшее значение счетчика.

Пример 8. Требуется решить следующую задачу. Под действием силы F с начальной скоростью V в вязкой среде движется тело массой M. Сопротивление пропорционально квадрату скорости с коэффициентом K. Определить время прохождения пяти контрольных точек траектории, расстояние до которых от точки старта заданы.

Пусть с помощью текстового редактора в файле Date.txt сформированы исходные данные в форме следующей таблицы:

ИСХОДНЫЕ ДАННЫЕ

M(кг)	F(Н)	V(м/с)	K(кг/м)	
36,3	2000	50,5	0,5	
	Координаты контрольных точек (м)			
$x(1)$	$x(2)$	$x(3)$	$x(4)$	$x(5)$
10	100	150	1000	3000

Требуется ввести числовые данные в вещественные переменные M, F, V, K и массив X[1..5], произвести расчеты и получить результаты в массиве T[1..5]. Результаты следует вывести на экран, а также в текстовый файл на диске с именем Result.txt.

Ниже приводится программа без расчетной части. Показаны только ввод исходных данных и вывод результатов.

```
Var M,F,V,K: Real; I: Integer;
    T,X: Array[1..5] Of Real;
    FR,FD: Text;
    Begin
       Assign(FD,'DATE.TXT'); Reset(FD);
       Assign(FR,'Result.txt'); Rewrite(FR);
       {Пропуск первых трех строк}
       ReadLn(FD); ReadLn(FD); ReadLn(FD);
       {Ввод данных}
       ReadLn(FD,M,F,V,K);
       {Пропуск трех строк}
       ReadLn(FD); ReadLn(FD); ReadLn(FD);
       {Ввод данных}
```

```
    For I:=1 To 5 Do Read(FD,X[I]);
            ...................
        {РАСЧЕТНАЯ ЧАСТЬ ПРОГРАММЫ}
            ...................
    {Вывод результатов на экран и в файл FR}
    WriteLn('Результаты'); WriteLn;
    WriteLn(FR,'Результаты'); WriteLn(FR);
    WriteLn('T(1) T(2) T(3) T(4) T(5)');
    WriteLn(FR,'T(1) T(2) T(3) T(4) T(5)');
    For I:=1 To 5 Do
    Begin
        Write(T[I]:8:2); Write(FR,T[I]:8:2)
    End;
    Close(FD); Close(FR)
End.
```

Результаты будут сохранены в файле `Result.txt`. Их можно посмотреть на экране, распечатать на бумаге. При необходимости этот файл может стать входным для другой программы, если содержащаяся в нем информация будет исходной для решения другой задачи.

Упражнения

1. Дан файл вещественных чисел. Определить количество нулевых значений в этом файле.
2. Даны два файла целых чисел. Определить, являются ли они тождественными.
3. Даны два символьных файла одинакового размера. Произвести обмен информацией между ними.
4. Имеется внешний текстовый файл. Напечатать первую из самых коротких его строк.
5. Описать процедуру `Lines(T)`, которая построчно печатает содержимое непустого текстового файла `T`, вставляя в начало каждой печатаемой строки ее порядковый номер (он должен занимать четыре позиции) и пробел.
6. В текстовом файле `T` записана непустая последовательность вещественных чисел, разделенных пробелами. Описать функцию `Max(T)` для нахождения наибольшего из этих чисел.

3.20. Комбинированный тип данных

Все структурированные типы данных, с которыми мы уже познакомились, представляют собой совокупности однотипных величин. *Комбинированный* тип данных — это структурированный тип, состоящий из фиксированного числа компонент (по-

лей) разного типа. Комбинированный тип имеет еще и другое название — *запись*.

Обычно запись содержит совокупность разнотипных атрибутов, относящихся к одному объекту. Например, анкетные сведения о студенте вуза могут быть представлены в виде информационной структуры (рис. 35).

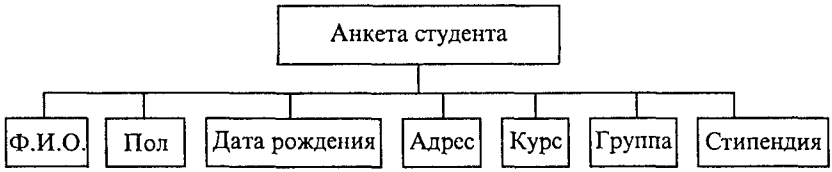


Рис. 35

Такая структура называется *двухуровневым деревом*. В Паскале эта информация может храниться в одной переменной типа `Record` (запись). Задать тип и описать соответствующую переменную можно следующим образом:

```
Type Anketa1=Record
       FIO: String[50];      {поля}
       Pol: Char;
       Dat: String[16];      {записи}
       Adres: String[50];
       Curs: 1..5;           {или элементы}
       Grup: 1..10;
         Stip: Real          {записи}
     End;
Var Student: Anketa1;
```

Такая запись, так же как и соответствующее ей дерево, называется двухуровневой.

К каждому элементу записи можно обратиться, используя *составное имя*, которое имеет следующую структуру:

```
<имя переменной>.<имя поля>
```

Например, `student.fio`; `student.dat` и т.п. Если, например, требуется полю *курс* присвоить значение 3, то это делается так:

```
Student.Curs:=3;
```

Поля записи могут иметь любой тип, в частности сами могут быть записями. Такая возможность используется в том случае, когда требуется представить многоуровневое дерево (более 2 уровней). Например, те же сведения о студентах можно отобразить трехуровневым деревом (рис. 36).

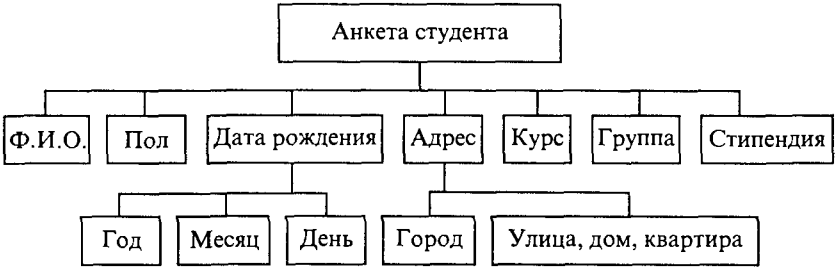


Рис. 36

Такая организация данных позволит, например, делать выборки информации по году рождения или по городу, где живут студенты. В этом случае описание соответствующей записи будет выглядеть так:

```
Type Anketa2=Record
        FIO: String[50];
        Pol: Char;
        Dat: Record
            God: Integer;
            Mes: String[10];
            Den: 1..31
               End;
        Adres: Record
            Gorod: String[20];
          UlDomKv: String[30];
               End;
        Curs: 1..5;
        Grup: 1..10;
        Stip: Real
        End;
Var Student: Anketa2;
```

Поля такой записи, находящиеся на третьем уровне, идентифицируются тройным составным именем.

Например, `student.Dat.God; student.Adres.Gorod.`

Приведем структурограмму задания комбинированного типа (рис. 37).

В программе могут использоваться массивы записей. Если на факультете 500 студентов, то все анкетные данные о них можно представить в массиве:

```
Var Student: Array[1..500] Of Anketa1;
```

В таком случае, например, год рождения пятого в списке студента хранится в переменной `student[5].Dat.God`.

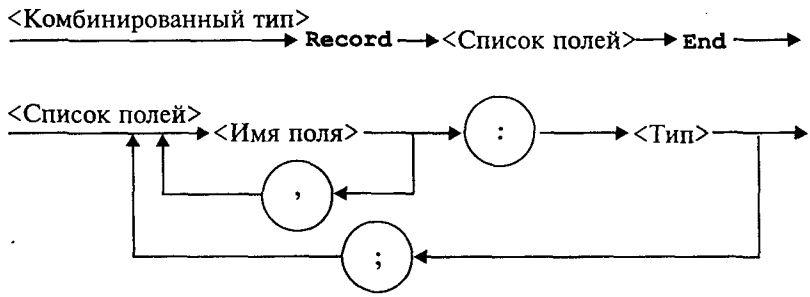


Рис. 37

Любая обработка записей, в том числе ввод и вывод, производится поэлементно. Например, ввод сведений о 500 студентах можно организовать следующим образом:

```
For I:=1 To 500 Do
With Student[I] Do
Begin
   Write('Ф.И.О.:'); ReadLn(FIO);
   Write('Пол (м/ж):'); ReadLn(Pol);
   Write('Дата рождения:'); ReadLn(Dat);
   Write('Адрес:'); ReadLn(Adres);
   Write('Курс:');  ReadLn(Curs);
   Write('Группа:'); ReadLn(Grup);
   Write('Стипендия (руб.):'); ReadLn(Stip)
End;
```

В этом примере использован *оператор присоединения*, который имеет следующий вид:

```
With <переменная типа запись> Do <оператор>;
```

Он позволяет, один раз указав имя переменной типа *запись* после слова `With`, работать в пределах оператора с именами полей как с обычными переменными, т. е. не писать громоздких составных имен.

Тип *запись* в Паскале может иметь переменный состав полей, который меняется в ходе выполнения программы. Такая возможность реализуется с использованием так называемой вариантной части записи. Подробнее об этом можно прочитать в книгах по Паскалю.

Работа с файлами записей. Чаще всего записи используются как элементы файлов, составляющих компьютерные информационные системы. Рассмотрим примеры программ, работающих с файлами записей.

Пример 1. Сформировать файл `FM.DAT`, содержащий экзаменационную ведомость одной студенческой группы. Записи файла состоят из следующих элементов: фамилия, имя, отчество; номер зачетной книжки; оценка.

```
Program Examen;
Type Stud=Record
              FIO: String[30];
              Nz: String[6];
              Mark: 2..5
            End;
Var Fstud: File Of Stud;
    S: Stud;
    N,I: Byte;
Begin
    Assign(Fstud,'FM.DAT'); Rewrite(Fstud);
    Write('Количество студентов в группе?');
    ReadLn(N);
    For I:=1 To N Do
    Begin
       Write(I:1,'-й,Фамилия И.О.'); ReadLn(S.FIO);
       Write('Номер зачетки:'); ReadLn(S.Nz);
       Write('Оценка:'); ReadLn(S.Mark);
       Write(Fstud,S)
    End;
    WriteLn('Формирование файла закончено!');
    Close(Fstud)
End.
```

Прежде чем перейти к следующему примеру, связанному с обработкой сформированного файла, рассмотрим еще одно средство работы с файлами, которое мы пока не обсуждали.

Прямой доступ к записям файла. В стандарте языка Паскаль допустим только последовательный доступ к элементам файла. Одной из дополнительных возможностей, реализованных в Турбо Паскале, является прямой доступ к записям файла.

Как уже отмечалось, элементы файла пронумерованы в порядке их занесения в файл, начиная с нуля. Задав номер элемента файла, можно непосредственно установить на него указатель. После этого можно читать или перезаписывать данный элемент. Установка указателя на нужный элемент файла производится процедурой

```
Seek(FV,n)
```

Здесь `FV` — имя файловой переменной, `n` — порядковый номер элемента. В следующем примере эта процедура будет использована.

Пример 2. Имеется файл, сформированный программой из предыдущего примера. Пусть некоторые студенты пересдали экзамен и получили новые оценки. Составить программу внесения результатов переэкзаменовки в файл. Программа будет запрашивать номер студента в ведомости и его новую оценку. Работа заканчивается, если вводится несуществующий номер (9999).

```
Program New_Marks;
Type Stud=Record
       FIO: String[30];
       Nz: String[6];
       Mark: 2..5
       End;
Var Fstud: File Of Stud;
    S: Stud;
    N: Integer;
Begin
       Assign(Fstud,'FM.DAT');
       Reset(Fstud);
       Write('Номер в ведомости?');
       ReadLn(N);
       While N<>9999 Do
       Begin
          Seek(Fstud,N-1);
          Read(Fstud,S);
          Write(S.FIO,'оценка?');
          ReadLn(S.Mark);
          Seek(Fstud,N-1);
          Write(Fstud,S);
          Write('Номер в ведомости?');
          ReadLn(N);
       End;
       WriteLn('Работа закончена!');
       Close(Fstud)
End.
```

Пример требует некоторых пояснений. Список студентов в ведомости пронумерован, начиная от 1, а записи в файле нумеруются от 0. Поэтому, если n — это номер в ведомости, то номер соответствующей записи в файле равен n – 1. После прочтения записи «номер n – 1» указатель смещается к следующей n-й записи. Для повторного занесения на то же место исправленной записи повторяется установка указателя.

Упражнения

1. Описать запись, содержащую сведения о рейсе самолета.
2. Описать массив записей, содержащий таблицу Д. И. Менделеева. Составить программу заполнения массива.
3. Рассматривая комплексное число как двухэлементную запись, составить процедуры выполнения арифметических операций с комплексными числами.
4. Сведения о деталях, хранящихся на складе, содержат следующие атрибуты: название, количество, стоимость одной детали.

Составить программы, решающие следующие задачи:

а) заполнить файл с информацией о деталях на складе;

б) вычислить общую стоимость деталей;

в) выяснить, какие детали имеются в наибольшем количестве, какие — в наименьшем;

г) вывести информацию о наличии на складе деталей данного типа и их количестве;

д) внести изменения в файл после выдачи со склада определенного количества данного вида деталей. Если какой-то тип деталей полностью выбран со склада, то уничтожить запись о нем в файле.

3.21. Указатели и динамические структуры

До сих пор мы рассматривали программирование, связанное с обработкой только статических данных. *Статическими называются такие величины, память под которые выделяется во время компиляции и сохраняется в течение всей работы программы.*

В Паскале существует и другой способ выделения памяти под данные, который называется динамическим. В этом случае *память под величины отводится во время выполнения программы. Такие величины будем называть динамическими.* Раздел оперативной памяти, распределяемый статически, называется *статической* памятью; динамически распределяемый раздел памяти называется *динамической* памятью.

Использование динамических величин предоставляет программисту ряд дополнительных возможностей. Во-первых, подключение динамической памяти позволяет увеличить объем обрабатываемых данных. Во-вторых, если потребность в каких-то данных отпала до окончания программы, то занятую ими память можно освободить для другой информации. В-третьих, использование динамической памяти позволяет создавать структуры данных переменного размера.

Работа с динамическими величинами связана с использованием еще одного типа данных — *ссылочного.* Величины, имеющие ссылочный тип, называют *указателями.*

Указатель содержит адрес поля в динамической памяти, хранящего величину определенного типа. Сам указатель располагается в статической памяти (рис. 38).

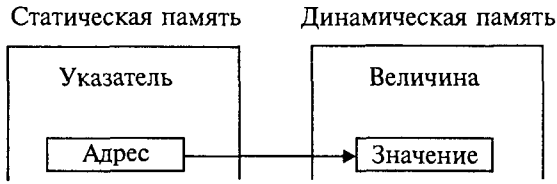


Рис. 38

Адрес величины — это номер первого байта поля памяти, в котором располагается величина. Размер поля однозначно определяется типом.

Величина ссылочного типа (указатель) описывается в разделе описания переменных следующим образом:

```
Var <идентификатор>:<ссылочный тип>
```

В стандарте Паскаля каждый указатель может ссылаться на величину только одного определенного типа, который называется *базовым* для указателя. Имя базового типа и указывается в описании в следующей форме:

```
<ссылочный тип>:=^<имя типа>
```

Вот примеры описания указателей:

```
Type Massiv=Array[1..100] Of Integer;
Var P1: ^Integer;
    P2: ^Char;
    PM: ^Massiv;
```

Здесь `P1` — указатель на динамическую величину целого типа; `P2` — указатель на динамическую величину символьного типа; `PM` — указатель на динамический массив, тип которого задан в разделе `Type`.

Сами динамические величины не требуют описания в программе, поскольку во время компиляции память под них не выделяется. Во время компиляции память выделяется только под статические величины. Указатели — это статические величины, поэтому они требуют описания.

Каким же образом происходит выделение памяти под динамическую величину? Память под динамическую величину, связанную с указателем, выделяется в результате выполнения стандартной процедуры `NEW`. Формат обращения к этой процедуре выглядит так:

```
NEW(<указатель>);
```

Считается, что после выполнения этого оператора создана динамическая величина, имя которой имеет следующий вид:

```
<имя динамической величины>::=<указатель>^.
```

Пусть в программе, в которой имеется приведенное выше описание, присутствуют операторы

```
NEW(P1); NEW(P2); NEW(PM);
```

После их выполнения в динамической памяти оказывается выделенным место под три величины (две скалярные и один массив), которые имеют идентификаторы

```
P1^, P2^, PM^
```

Например, обозначение P1^ можно расшифровать так: динамическая переменная, на которую ссылается указатель P1.

На схеме, представленной на рис. 39, показана связь между динамическими величинами и их указателями.

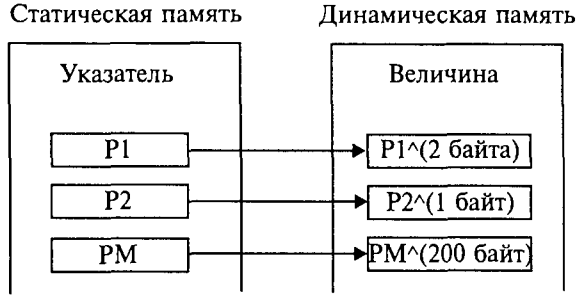


Рис. 39

Дальнейшая работа с динамическими переменными происходит точно так же, как со статическими переменными соответствующих типов. Им можно присваивать значения, их можно использовать в качестве операндов в выражениях, параметров подпрограмм и т.п. Например, если переменной P1^ нужно присвоить значение 25, переменной P2^ присвоить значение символа 'W', а массив PM^ заполнить по порядку целыми числами от 1 до 100, то это делается так:

```
P1^:=25;
P2^:='W';
For I:=1 To 100 Do PM^[I]:=I;
```

Кроме процедуры NEW значение указателя может определяться оператором присваивания:

```
<указатель>:=<ссылочное выражение>;
```

В качестве ссылочного выражения можно использовать:

- указатель;
- ссылочную функцию (т.е. функцию, значением которой является указатель);
- константу Nil.

Nil — это зарезервированная константа, обозначающая пустую ссылку, т.е. ссылку, которая ни на что не указывает. При присваивании базовые типы указателя и ссылочного выражения должны быть одинаковыми. Константу Nil можно присваивать указателю с любым базовым типом.

До присваивания значения ссылочной переменной (с помощью оператора присваивания или процедуры NEW) она является неопределенной.

Ввод и вывод указателей не допускается.

Рассмотрим пример. Пусть в программе описаны следующие указатели:

```
Var D,P: ^Integer;
    K: ^Boolean;
```

Тогда допустимыми являются операторы присваивания

```
D:=P; K:=Nil;
```

поскольку соблюдается принцип соответствия типов. Оператор `K:=D` ошибочен, так как базовые типы у правой и левой части разные.

Если динамическая величина теряет свой указатель, то она становится «мусором». В программировании под этим словом понимают информацию, которая занимает память, но уже не нужна.

Представьте себе, что в программе, в которой присутствуют описанные выше указатели, в разделе операторов записано следующее:

```
NEW(D); NEW(P);
   {Выделено место в динамической памяти под
   две целые переменные. Указатели получили
   соответствующие значения}
D^:=3; P^:=5;
   {Динамическим переменным присвоены
   значения}
P:=D;
   {Указатели P и D стали ссылаться на одну и
   ту же величину, равную 3}
WriteLn(P^,D^); {Дважды напечатается число 3}
```

Таким образом, динамическая величина, равная 5, потеряла свой указатель и стала недоступной. Однако место в памяти она занимает. Это и есть пример возникновения «мусора». На схеме, представленной на рис. 40, показано, что произошло в результате выполнения оператора `P:=D`.

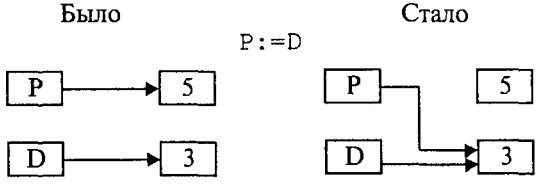


Рис. 40

В Паскале имеется стандартная процедура, позволяющая освобождать память от данных, потребность в которых отпала. Ее формат:

```
DISPOSE(<указатель>);
```

Например, если динамическая переменная P^ больше не нужна, то оператор

```
DISPOSE(P)
```

удалит ее из памяти. После этого значение указателя P становится неопределенным. Особенно существенным становится эффект экономии памяти при удалении больших массивов.

В версиях Турбо Паскаля, работающих под управлением операционной системы MS DOS, под данные одной программы выделяется 64 килобайта памяти. Это и есть статическая область памяти. При необходимости работать с большими массивами информации этого может оказаться мало. Размер динамической памяти намного больше (сотни килобайт). Поэтому использование динамической памяти позволяет существенно увеличить объем обрабатываемой информации.

Следует отчетливо понимать, что работа с динамическими данными замедляет выполнение программы, поскольку доступ к величине происходит в два шага: сначала ищется указатель, затем по нему — величина. Как это часто бывает, действует «закон сохранения неприятностей»: выигрыш в памяти компенсируется проигрышем во времени.

Пример 1. Создать вещественный массив из 10 000 чисел, заполнить его случайными числами в диапазоне от 0 до 1. Вычислить среднее значение массива. Очистить динамическую память. Создать целый массив размером 10 000, заполнить его случайными целыми числами в диапазоне от –100 до 100 и вычислить его среднее значение.

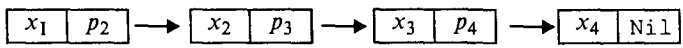
```
Program Sr;
Const NMax=10000;
Type Diapazon=1..NMax;
     MasInt=Array[Diapazon] Of Integer;
     MasReal=Array[Diapazon] Of Real;
Var PIint: ^MasInt;
    PReal: ^MasReal;
    I,Midint: LongInt;
    MidReal: Real;
Begin
    MidReal:=0; MidInt:=0;
    Randomize;
    NEW(PReal);
    {Выделение памяти под вещественный массив}
    {Вычисление и суммирование массива}
    For I:=1 To NMax Do
    Begin PReal^[I]:=Random;
        MidReal:=MidReal+PReal^[I]
    End;
```

```
    DISPOSE(PReal);{Удаление вещественного
                    массива}
    NEW(PInt); {Выделение памяти под целый
                массив}
    {Вычисление и суммирование целого массива}
     For I:=1 To NMax Do
     Begin
         PInt^[I]:=Random(200)-100;
         MidInt:=MidInt+PInt^[I]
     End;
     {Вывод средних значений}
     WriteLn('среднее целое равно:',MidInt Div
             Max);
     WriteLn('среднее вещественное равно:',
     (MidReal/NMax):10:6)
End.
```

Связанные списки. Обсудим вопрос о том, как в динамической памяти можно создать структуру данных переменного размера.

Разберем следующий пример. В процессе физического эксперимента многократно снимаются показания прибора (допустим, термометра) и записываются в компьютерную память для дальнейшей обработки. Заранее неизвестно, сколько измерений будет произведено.

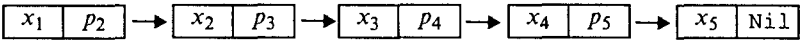
Если для обработки таких данных не использовать внешнюю память (файлы), то разумно расположить их в динамической памяти. Во-первых, динамическая память позволяет хранить больший объем информации, чем статическая. А во-вторых, в динамической памяти эти числа можно организовать в *связанный список*, который не требует предварительного указания количества чисел, подобно массиву. Что же такое связанный список? Схематически он выглядит так:



Каждый элемент списка состоит из двух частей: информационной части (x_1, x_2 и т.д.) и указателя на следующий элемент списка (p_2, p_3 и т.д.). Последний элемент имеет пустой указатель `Nil`. Связанный список такого типа называется однонаправленной цепочкой.

Для сформулированной выше задачи информационная часть представляет набор вещественных чисел: x_1 — результат первого измерения, x_2 — результат второго измерения и т.д., x_4 — результат последнего измерения. Связанный список обладает тем замечательным свойством, что его можно дополнять по мере поступления новой информации. Добавление происходит путем присое-

динения нового элемента к концу списка. Значение Nil в последнем элементе заменяется ссылкой на новый элемент цепочки:



Связанный список не занимает лишней памяти. Память расходуется в том объеме, который требуется для поступившей информации.

В программе для представления элементов цепочки используется комбинированный тип (запись). Для нашего примера тип такого элемента может быть следующим:

```
Type Pe=^Elem;
     Elem=Record
            T: Real;
            P: Pe
          End;
```

Здесь Pe — ссылочный тип на переменную типа Elem. Этим именем обозначен комбинированный тип, состоящий из двух полей: T — вещественная величина, хранящая температуру, P — указатель на динамическую величину типа Elem.

В таком описании нарушен один из основных принципов Паскаля, согласно которому на любой программный объект можно ссылаться только после его описания. В самом деле, тип Pe определяется через тип Elem, а тот, в свою очередь, определяется через тип Pe. Однако в Паскале допускается единственное исключение из этого правила, связанное со ссылочным типом. Приведенный фрагмент программы является правильным.

Пример 2. Рассмотрим программу формирования связанного списка в ходе ввода данных.

```
Type Pe=^TypElem;
     TypElem=Record
               T: Real; P: Pe
               End;
Var Elem,Beg: Pe;
X: Real; Ch: Char;
Begin {Определение адреса начала списка и его
       сохранение}
NEW(Elem); Beg:=Elem;
Elem^.P:=Elem;
{Диалоговый ввод значений с занесением их в
список и организацией связи между элементами}
While Elem^.P<>Nil Do
Begin
     Write('Вводите число:');
     ReadLn(Elem^.T);
```

```
        Write('Повторить ввод? (Y/N)');
        ReadLn(Ch);
        If (Ch='n') Or (Ch='N')
        Then Elem^.P:=Nil
        Else Begin
                  NEW(Elem^.P);
                  Elem:=Elem^.P
             End
End;
WriteLn(«Ввод данных закончен»);
{Вывод полученной числовой последовательности}
WriteLn(«Контрольная распечатка»);
Elem:=Beg;
Repeat
    WriteLn(N,':',Elem^.T:8:3);
    Elem:=Elem^.P
Until Elem=Nil
End.
```

Здесь ссылочная переменная `Beg` используется для сохранения адреса начала цепочки. Всякая обработка цепочки начинается с ее первого элемента. В программе показано, как происходит продвижение по цепочке при ее обработке (в данном примере — распечатке информационной части списка по порядку).

Однонаправленная цепочка — простейший вариант связанного списка. В практике программирования используются двунаправленные цепочки (когда каждый элемент хранит указатель на следующий и на предыдущий элементы списка), а также двоичные деревья. Подобные структуры данных называются динамическими структурами.

Пример 3. Задача о стеке. Одной из часто употребляемых в программировании динамических структур данных является стек. Стек — это связанная цепочка, начало которой называется вершиной. Состав элементов постоянно меняется. Каждый вновь поступающий элемент размещается в вершине стека. Удаление элементов из стека также производится с вершины.

Стек подобен детской пирамидке, в которой на стержень надеваются кольца. Всякое новое кольцо оказывается на вершине пирамиды. Снимать кольца можно только сверху. Принцип изменения содержания стека часто формулируют так: «Последним пришел — первым вышел».

Составим процедуры добавления элемента в стек (`INSTEK`) и исключения элемента из стека (`OUTSTEK`). При этом будем считать, что элементы стека — символьные величины.

В процедурах используется тип `Pe`, который должен быть глобально объявлен в основной программе.

```
Type Pe=^TypElem;
     TypElem=Record
                C: Char; P: Pe
                End;
```

В процедуре INSTEK аргументом является параметр Sim, содержащий включаемый в стек символ. Ссылочная переменная Beg содержит указатель на вершину стека при входе в процедуру и при выходе из нее. Следовательно, этот параметр является и аргументом, и результатом процедуры. В случае, когда стек пустой, указатель Beg равен Nil.

```
Procedure INSTEK(Var Beg: Pe; Sim: Char);
Var X: Pe;
Begin New(X);
      X^.C:=Sim;
      X^.P:=Beg;
      Beg:=X
End;
```

В процедуре OUTSTEK параметр Beg играет ту же роль, что и в предыдущей процедуре. После удаления последнего символа его значение станет равным Nil. Ясно, что если стек пустой, то удалять из него нечего. Логический параметр Flag позволяет распознать этот случай. Если на выходе из процедуры Flag=True, то, значит, удаление произошло; если Flag=False, значит, стек был пуст и ничего не изменилось.

Процедура не оставляет после себя «мусора», освобождая память из-под удаленных элементов.

```
Procedure OUTSTEK(Var Beg: Pe; Var Flag: Boolean);
Var X: Pe;
Begin
     If Beg=Nil
     Then Flag:=False
     Else Begin
           Flag:=True;
           X:=Beg;
           Beg:=Beg^.P;
           DISPOSE(X)
          End
End;
```

Упражнения

1. В программе имеются описания

```
Var P,Q: ^Integer; R: ^Char;
```

Какие из следующих операторов неправильные и почему?

а) P:=Q; б) Q:=R; в) P:=Nil;
г) R:=Nil; д) Q:=P^; е) P^:=Nil;
ж) R^:=P^; з) Q^:=Ord(R^);
и) If R<>Nil Then R^:=Nil^;
к) If Q>Nil Then Q^:=P^;
л) If P=Q Then Write(Q);
м) If Q<>R Then Read(R^).

2. В программе имеются описания

```
Type A=^Char;
B=Record F1: Char; F2: A End;
Var P: ^B; Q: A;
```

Определить значения всех указателей и динамических переменных после выполнения следующих операторов:

```
NEW(Q); Q^:='7';
NEW(P); P^.F1:=Succ(Q^); P^.F2:=Q;
```

3. Найти ошибки в следующей программе:

```
Program Example;
Var A,B: ^Integer;
Begin If A=Nil Then Read(A);
     A^:=5; B:=Nil; B^:=2;
     NEW(B); Read(B^); WriteLn(B,B^);
     NEW(A); B:=A; DISPOSE(A); B^:=4
End.
```

4. Составить программу занесения в динамическую память вещественного массива из 10 000 чисел, хранящегося в файле на магнитном диске, а также поиска в нем значения и номера первого максимального элемента.

5. Решить предыдущую задачу при условии, что размер числового массива заранее не определен (т.е. определяется размером файла: предполагается, что размер файла не превышает размера динамической памяти). Данные в динамической памяти организовать в виде связанного списка.

6. Связанный список представляет собой символьную цепочку из строчных латинских букв. Описать процедуру или функцию, которая:

а) определяет, является ли список пустым;

б) заменяет в списке все вхождения одного символа на вхождения другого;

в) меняет местами первый и последний элементы непустого списка;

г) проверяет, упорядочены ли элементы списка по алфавиту;

д) определяет, какая буква входит в список наибольшее количество раз.

7. Составить тестовую программу, проверяющую правильность работы процедур INSTEK и OUTSTEK из задачи о стеке.

8. Очередью называется динамическая структура (связанная цепочка), в которой действует принцип: «Первым пришел — первым вышел». Напишите процедуры включения элемента в очередь и исключения элемента из очереди. Составьте тестовую программу, проверяющую работу процедур.

3.22. Внешние подпрограммы и модули

Стандартный Паскаль не располагает средствами разработки и поддержки библиотек программиста (в отличие, скажем, от Фортрана и других языков программирования высокого уровня), которые компилируются отдельно и в дальнейшем могут быть использованы не только самим разработчиком. Если программист имеет достаточно большие наработки и те или иные подпрограммы могут быть использованы при написании новых приложений, то приходится эти подпрограммы целиком включать в новый текст.

В Турбо Паскале это ограничение преодолевается за счет, во-первых, введения внешних подпрограмм, во-вторых, разработки и использования модулей. В данном разделе мы рассмотрим оба способа.

Организация внешних подпрограмм. Начнем с внешних подпрограмм. В этом случае исходный текст каждой процедуры или функции хранится в отдельном файле и при необходимости с помощью специальной директивы компилятора включается в текст создаваемой программы.

Проиллюстрируем этот прием на примере задачи целочисленной арифметики. Условие задачи: дано натуральное число *n*. Найти сумму первой и последней цифр этого числа.

Для решения будет использована функция, вычисляющая количество цифр в записи натурального числа. Вот ее возможный вариант:

```
Function Digits(N: LongInt): Byte;
Var Kol: Byte;
Begin
     Kol:=0;
     While N<>0 Do
     Begin
          Kol:=Kol+1;
          N:=N Div 10
     End;
     Digits:=Kol
End;
```

Сохраним этот текст в файле с расширением inc (это расширение внешних подпрограмм в Турбо Паскале), например digits.inc.

Опишем еще одну функцию: возведение натурального числа в натуральную степень (a^n).

```
Function Power(A,N:LongInt): LongInt;
Var I,St: LongInt;
Begin
  St:=1;
  For I:=1 To N Do
    St:=St*A;
  Power:=St
End;
```

А теперь составим основную программу, решающую поставленную задачу. В ней будут использованы описанные выше функции.

```
Program Example1;
Var N,S: Integer;
{$I digits.inc} {подключение внешней функции из
файла  digits.inc, вычисляющей количество цифр в
записи числа}
{$I power.inc} {подключение внешней функции из
файла power.inc, вычисляющей результат возведения
числа A в степень N}
Begin
    Write('Введите натуральное число:'); ReadLn(N);
    {для определения последней цифры числа N
    берется остаток от деления этого числа на 10, а
    для определения первой цифры N делится на 10,
    возведенное в степень на единицу меньшую, чем
    количество цифр в записи числа (т.к. нумерация
    разрядов начинается с 0)}
    S:=N Mod 10+N Div Power(10,Digits(N)-1);
    WriteLn('Искомая сумма:',S)
End.
```

{$I <имя файла>} — это директива компилятора (псевдокомментарий), позволяющая в данное место текста программы вставить содержимое файла с указанным именем.

Файлы с расширением inc можно накапливать на магнитном диске, формируя таким образом личную библиотеку подпрограмм.

Внешние процедуры создаются и внедряются в использующие их программы точно так же, как и функции в рассмотренном примере.

Создание и использование модулей. Далее речь пойдет о модулях: их структуре, разработке, компиляции и использовании.

Модуль — это набор ресурсов (функций, процедур, констант, переменных, типов и т.д.), разрабатываемых и хранимых независимо от использующих их программ. В отличие от внешних подпрограмм модуль может содержать достаточно большой набор процедур и функций, а также других ресурсов для разработки программ. В основе идеи модульности лежат принципы структурного программирования. Существуют стандартные модули Турбо Паскаля (SYSTEM, CRT, GRAPH и т.д.), справочная информация по которым дана в приложении.

Модуль имеет следующую структуру:

```
Unit <имя модуля>; {заголовок модуля}
Interface
        {интерфейсная часть}
Implementation
        {раздел реализации}
Begin
        {раздел инициализации модуля}
End.
```

После служебного слова Unit записывается имя модуля, которое (для удобства дальнейших действий) должно совпадать с именем файла, содержащего данный модуль. Поэтому (как принято в MS DOS) имя не должно содержать более 8 символов.

В разделе Interface объявляются все ресурсы, которые будут в дальнейшем доступны программисту при подключении модуля. Для подпрограмм здесь лишь указывается полный заголовок.

В разделе Implementation описываются все подпрограммы, которые были ранее объявлены. Кроме того, в нем могут содержаться свои константы, переменные, типы, подпрограммы и т.д., которые носят вспомогательный характер и используются для написания основных подпрограмм. В отличие от ресурсов, объявленных в разделе Interface, все, что дополнительно объявляется в Implementation, уже не будет доступно при подключении модуля. При описании основной подпрограммы достаточно указать ее имя (т.е. не требуется полностью переписывать весь заголовок), а затем записать тело подпрограммы.

Наконец, раздел инициализации (часто отсутствующий) содержит операторы, которые должны быть выполнены сразу же после запуска программы, использующей модуль.

Приведем пример разработки и использования модуля. Поскольку рассмотренная ниже задача достаточно элементарна, ограничимся распечаткой текста программы с подробными комментариями.

Рассмотрим следующую задачу. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над обыкновенными дробями вида P/Q (P — целое, Q — натуральное):

1) сложение;
2) вычитание;
3) умножение;
4) деление;
5) сокращение дроби;
6) возведение дроби в степень N (N — натуральное);
7) функции, реализующие операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Дробь представить следующим типом:

```
Type Frac=Record
       P: Integer;
       Q: 1..32767
       End;
```

Используя этот модуль, решить задачи:

1. Дан массив A, элементы которого — обыкновенные дроби. Найти сумму всех элементов и их среднее арифметическое; результаты представить в виде несократимых дробей.

2. Дан массив A, элементы которого — обыкновенные дроби. Отсортировать его в порядке возрастания.

```
Unit Droby;
Interface
      Type
       Natur=1..High(LongInt);
       Frac=Record
              P: LongInt;
              Q: Natur
            End;
Procedure Sokr(Var A: Frac);
Procedure Summa(A,B: Frac; Var C: Frac);
Procedure Raznost(A,B: Frac; Var C: Frac);
Procedure Proizvedenie(A,B: Frac; Var C: Frac);
Procedure Chastnoe(A,B: Frac; Var C: Frac);
Procedure Stepen(A: Frac; N: Natur; Var C: Frac);
Function Menshe(A,B: Frac): Boolean;
Function Bolshe(A,B: Frac): Boolean;
Function Ravno(A,B: Frac): Boolean;
Function MensheRavno(A,B: Frac): Boolean;
Function BolsheRavno(A,B: Frac): Boolean;
Function NeRavno(A,B: Frac): Boolean;
{Раздел реализации модуля}
Implementation
{Наибольший общий делитель двух чисел -
вспомогательная функция, ранее не объявленная}
Function NodEvklid(A,B: Natur): Natur;
Begin
```

```
  While A<>B Do
    If A>B Then
         If A Mod B<>0 Then A:=A Mod B
                       Else A:=B
         Else
         If B Mod A<>0 Then B:=B Mod A
                       Else B:=A;
    NodEvklid:=A
End;
{Сокращение дроби}
Procedure Sokr;
Var M,N: Natur;
Begin
   If A.P<>0
   Then
   Begin
      If A.P<0
      Then M:=Abs(A.P)
      Else M:=A.P; {Совмещение типов, т.к. A.P -
      LongInt}
      N:=NodEvklid(M,A.Q);
      A.P:=A.P Div N;
      A.Q:=A.Q Div N
   End
End;
Procedure Summa; {Сумма дробей}
Begin
      {Знаменатель дроби}
      C.Q:=(A.Q*B.Q) Div NodEvklid(A.Q,B.Q);
      {Числитель дроби}
      C.P:=A.P*C.Q Div A.Q+B.P*C.Q Div B.Q;
      Sokr(C)
End;
Procedure Raznost; {Разность дробей}
Begin
      {Знаменатель дроби}
      C.Q:=(A.Q*B.Q) Div NodEvklid(A.Q,B.Q);
      {Числитель дроби}
      C.P:=A.P*C.Q Div A.Q-B.P*C.Q Div B.Q;
      Sokr(C)
End;
Procedure Proizvedenie; {Умножение дробей}
Begin
      {Знаменатель дроби}
      C.Q:=A.Q*B.Q;
      {Числитель дроби}
```

```
      C.P:=A.P*B.P;
      Sokr(C)
End;
Procedure Chastnoe; {Деление дробей}
Begin
      {Знаменатель дроби}
      C.Q:=A.Q*B.P;
      {Числитель дроби}
      C.P:=A.P*B.Q;
      Sokr(C)
End;
Procedure Stepen; {Возведение дроби в степень}
Var I: Natur;
Begin
      C.Q:=1;
      C.P:=1;
      Sokr(A);
      For I:=1 To N Do
            Proizvedenie(A,C,C)
End;
Function Menshe; {отношение '<' между дробями}
Begin
      Menshe:=A.P*B.Q<A.Q*B.P
End;
Function Bolshe; {отношение '>' между дробями}
Begin
      Bolshe:=A.P*B.Q>A.Q*B.P
End;
Function Ravno; {отношение '=' между дробями}
Begin
      Ravno:=A.P*B.Q=A.Q*B.P
End;
Function BolsheRavno; {отношение '>=' между
дробями}
Begin
      BolsheRavno:=Bolshe(A,B) Or Ravno(A,B)
End;
Function MensheRavno; {отношение '<=' между
дробями}
Begin
      MensheRavno:=Menshe(A,B) Or Ravno(A,B)
End;
Function NeRavno; {отношение '<>' между дробями}
Begin
      NeRavno:=Not Ravno(A,B)
End;
```

```
{Раздел инициализации модуля}
Begin
End.
```

При разработке модуля рекомендуется такая последовательность действий:

1. Спроектировать модуль, т.е. определить основные и вспомогательные подпрограммы и другие ресурсы.
2. Описать компоненты модуля.
3. Каждую подпрограмму целесообразно отладить отдельно, после чего «вклеить» в текст модуля.

Сохраним текст разработанной программы в файле DROBY.PAS и откомпилируем наш модуль. Для этого можно воспользоваться внешним компилятором, поставляемым вместе с Турбо Паскалем. Команда будет выглядеть так: TPC DROBY.PAS. Если в тексте нет синтаксических ошибок, получим файл DROBY.TPU, иначе будет выведено соответствующее сообщение с указанием строки, содержащей ошибку. Другой вариант компиляции: в меню системы программирования Турбо Паскаль выбрать **Compile/Destination Disk**, затем — **Compile/Build**.

Теперь можно подключить модуль к программе, где планируется его использование. Решим первую задачу — выполним суммирование массива дробей.

```
Program Sum;
Uses Droby;
Var A: Array[1..100] Of Frac;
    I,N: Integer;
    S: Frac;
Begin
  Write('Введите количество элементов массива:');
  ReadLn(N);
  S.P:=0; S.Q:=1; {Первоначально сумма равна нулю}
  For I:=1 To N Do {Вводим и суммируем дроби}
  Begin
  Write('Введите числитель',I,'-й дроби:');
  ReadLn(A[I].P);
  Write('Введите знаменатель ',I,'-й дроби:');
  ReadLn(A[I].Q);
  Summa(A[I],S,S);
  End;
  WriteLn('Ответ:',S.P,'/',S.Q)
End.
```

Вторую задачу читателю предлагается решить самостоятельно.

Как видно из примера, для подключения модуля используется служебное слово Uses, после которого указывается имя модуля. Данная строка записывается сразу же после заголовка программы.

Если необходимо подключить несколько модулей, они перечисляются через запятую.

При использовании ресурсов модуля программисту совсем не обязательно иметь представление о том, как работают вызываемые подпрограммы. Достаточно знать назначение подпрограмм и их спецификации, т.е. имена и параметры. По такому принципу осуществляется работа со всеми стандартными модулями. Поэтому если программист разрабатывает модули не только для личного пользования, ему необходимо выполнить полное описание всех доступных при подключении ресурсов.

3.23. Объектно-ориентированное программирование

Основные понятия объектно-ориентированного программирования (ООП). Основополагающей идеей одного из современных подходов к программированию — объектно-ориентированному — является объединение данных и обрабатывающих их процедур в единое целое — объекты.

Объектно-ориентированное программирование — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию, основанную на принципах наследуемости. При этом объект характеризуется как совокупностью всех своих свойств и их текущих значений, так и совокупностью допустимых для данного объекта действий.

Несмотря на то что в различных источниках делается акцент на те или иные особенности внедрения и применения ООП, три основных (базовых) понятия ООП остаются неизменными. К ним относятся:

- наследование (Inheritance);
- инкапсуляция (Encapsulation);
- полиморфизм (Polymorphism).

Эти понятия как три кита лежат в основе ООП.

При процедурном подходе требуется описать каждый шаг, каждое действие алгоритма для достижения конечного результата. В отличие от него объектно-ориентированный подход оставляет за объектом право решать, как отреагировать и что сделать в ответ на поступивший вызов. Достаточно в стандартной форме поставить перед ним задачу и получить ответ.

Объект состоит из следующих трех частей:

- имени объекта;
- состояния (переменных состояния);
- методов (операций).

Можно дать обобщающее определение: ***объект ООП*** — это совокупность переменных состояния и связанных с ними методов

(операций). Упомянутые методы определяют, как объект взаимодействует с окружающим миром.

Под *методами объекта* понимают процедуры и функции, объявление которых включено в описание объекта и которые выполняют действия. Возможность управлять состояниями объекта посредством вызова методов в итоге и определяет поведение объекта. Совокупность методов часто называют интерфейсом объекта.

Инкапсуляция — это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект.

Применяя инкапсуляцию, мы защищаем данные, принадлежащие объекту, от возможных ошибок, которые могут возникнуть при прямом доступе к этим данным. Кроме того, применение указанного принципа очень часто помогает локализовать возможные ошибки в коде программы. А это намного упрощает процесс поиска и исправления этих ошибок. Можно сказать, что инкапсуляция обеспечивает сокрытие данных, что позволяет защитить эти данные. Однако применение инкапсуляции ведет к снижению эффективности доступа к элементам объекта. Это обусловлено необходимостью вызова методов для изменения внутренних элементов (переменных) объекта. Но при современном уровне развития вычислительной техники подобные потери в эффективности не играют существенной роли.

Наследование — это процесс, посредством которого один объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него. В итоге создается иерархия объектных типов, где поля данных и методов «предков» автоматически являются и полями данных и методов «потомков».

Смысл и универсальность наследования заключаются в том, что не надо каждый раз заново («с нуля») описывать новый объект, а можно указать «родителя» (базовый класс) и описать отличительные особенности нового класса. В результате новый объект будет обладать всеми свойствами родительского класса плюс своими собственными отличительными особенностями.

Пример 1. Можно создать базовый класс «транспортное средство», который универсален для всех средств передвижения, к примеру, на четырех колесах. Этот класс «знает», как двигаются колеса, как они поворачиваются, тормозят и т.д. Затем на основе этого класса создадим класс «легковой автомобиль». Поскольку новый класс унаследован из класса «транспортное средство», унаследованы все особенности этого класса, и нам не надо в очередной раз описывать, как двигаются колеса и т.д. Мы просто

добавим те черты, которые характерны для легковых автомобилей. В то же время мы можем взять за основу этот же класс «транспортное средство» и построить класс «грузовые автомобили». Описав отличительные особенности грузовых автомобилей, получим новый класс «грузовые автомобили». А, к примеру, на основании класса «грузовой автомобиль» уже можно описать определенный подкласс грузовиков и т.д. Таким образом, сначала формируем простой шаблон, а затем, усложняя и конкретизируя, поэтапно создаем все более сложные шаблоны.

Полиморфизм — это свойство, которое позволяет одно и то же имя использовать для решения нескольких технически разных задач. Полиморфизм подразумевает такое определение методов в иерархии типов, при котором метод с одним именем может применяться к различным родственным объектам. В общем смысле концепцией полиморфизма является идея «один интерфейс — множество методов». Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного интерфейса для единого класса действий. Выбор конкретного действия в зависимости от ситуации возлагается на компилятор.

Пример 2. Пусть есть класс «автомобиль», в котором описано, как должен передвигаться автомобиль, как он поворачивает, как подает сигнал и т.д. Там же описан метод «переключение передачи». Допустим, что в этом методе класса «автомобиль» описана автоматическая коробка передач. А теперь необходимо описать класс «спортивный автомобиль», у которого механическое (ручное) переключение скоростей. Конечно, можно было бы описать заново все методы для класса «спортивный автомобиль». Вместо этого указываем, что класс «спортивный автомобиль» унаследован из класса «автомобиль», а следовательно, он обладает всеми свойствами и методами, описанными для класса-родителя. Единственное, что надо сделать — это переписать метод «переключение передач» для механической коробки передач. В результате при вызове метода «переключение передач» будет выполняться метод не родительского класса, а самого класса «спортивный автомобиль».

Механизм работы ООП в таких случаях можно описать примерно так: при вызове того или иного метода класса сначала ищется метод в самом классе. Если метод найден, то он выполняется, и поиск этого метода завершается. Если же метод не найден, то обращаемся к родительскому классу и ищем вызванный метод в нем. Если он найден, то поступаем, как при нахождении метода в самом классе. А если нет, то продолжаем дальнейший поиск вверх по иерархическому дереву, вплоть до корня (верхнего класса) иерархии. Этот пример отражает так называемый механизм раннего связывания.

Объекты в Турбо Паскале. ***Инкапсуляция.*** Для описания объектов зарезервировано слово `object`. Тип `object` — это структура данных, которая содержит поля и методы. Описание объектного типа выглядит следующим образом:

```
Type <идентификатор типа объекта>=Object
<поле>;
...
<поле>;
<метод>;
...
<метод>;
End;
```

Поле содержит имя и тип данных. Методы — это процедуры или функции, объявленные внутри декларации объектного типа, в том числе и особые процедуры, создающие и уничтожающие объекты (конструкторы и деструкторы). Объявление метода внутри описания объектного типа состоит только из заголовка (как в разделе `Interface` в модуле).

Пример 3. Опишем объект «обыкновенная дробь» с методами «НОД числителя и знаменателя», «сокращение», «натуральная степень».

```
Type Natur=1..32767;
Frac=Record P: Integer; Q: Natur End;
Drob=Object A: Frac;
     Procedure NOD(Var C: Natur);
     Procedure Sokr;
     Procedure Stepen(N: Natur; Var C: Frac);
        End;
```

Описание объектного типа, собственно, и выражает такое свойство, как *инкапсуляция.*

Проиллюстрируем далее работу с описанным объектом, реализацию его методов и обращение к указанным методам. При этом понадобятся некоторые вспомогательные методы.

```
Type Natur=1..MaxInt;
     Frac=Record P: Integer; Q: Natur End;
{Описание объектного типа}
Drob=Object
     A: Frac;
      Procedure Vvod; {ввод дроби}
      Procedure NOD(Var C: Natur); {НОД}
      Procedure Sokr;
      Procedure Stepen(N: Natur; Var C: Frac);
      Procedure Print; {вывод дроби}
```

```
    End;
{Описания методов объекта}
Procedure Drob.NOD;
Var M,N: Natur;
Begin M:=Abs(A.P); N:=A.Q;
   While M<>N Do
   If M>N
   Then If M Mod N<>0 Then M:=M Mod N Else M:=N
   Else If N Mod M<>0 Then N:=N Mod M Else N:=M;
   C:=M
End;
Procedure Drob.Sokr;
Var N: Natur;
Begin If A.P<>0
      Then Begin
           Drob.NOD(N);
           A.P:=A.P Div N; A.Q:=A.Q Div N
           End
      Else A.Q:=1
End;
Procedure Drob.Stepen;
Var I: Natur;
Begin
    C.P:=1; C.Q:=1;
    For I:=1 To N Do Begin C.P:=C.P*A.P;
                           C.Q:=C.Q*A.Q
                           End;
End;
Procedure Drob.Vvod;
Begin
   Write('Введите числитель дроби:'); ReadLn(A.P);
   Write('Введите знаменатель дроби:');ReadLn(A.Q);
End;
Procedure Drob.Print;
Begin WriteLn(A.P,'/',A.Q) End;
{Основная программа}
Var Z: Drob; F: Frac;
Begin
  Z.Vvod; {ввод дроби}
  Z.Print; {печать введенной дроби}
  Z.Sokr; {сокращение введенной дроби}
  Z.Print; {печать дроби после сокращения}
  Z.Stepen(4,F); {возведение введенной дроби в 4-ю
                  степень}
  WriteLn(F.P,'/',F.Q)
End.
```

Прокомментируем отдельные моменты в рассмотренном примере. Во-первых, реализация методов осуществляется в разделе описаний, после объявления объекта, причем при реализации метода достаточно указать его заголовок без списка параметров, но с указанием объектного типа, методом которого он является. Еще раз отметим, что все это напоминает создание модуля, где те ресурсы, которые доступны при его подключении, прежде всего объявляются в разделе `Interface`, а затем реализуются в разделе `Implementation`. В действительности объекты и их методы реализуют чаще всего именно в виде модулей.

Во-вторых, все действия над объектом выполняются только с помощью его методов.

В-третьих, для работы с отдельным экземпляром объектного типа в разделе описания переменных должна быть объявлена переменная (или переменные) соответствующего типа. Легко видеть, что объявление статических объектов не отличается от объявления других переменных, а их использование в программе напоминает использование записей.

Наследование. Объектные типы можно выстроить в иерархию. Один объектный тип может наследовать компоненты из другого объектного типа. Наследующий объект называется потомком. Объект, которому наследуют, — предком. Если предок сам является чьим-либо наследником, то потомок наследует и эти поля и методы. Следует подчеркнуть, что наследование относится только к типам, но не экземплярам объекта.

Описание типа-потомка имеет отличительную особенность:

```
<имя типа-потомка>=Object(<имя типа-предка>),
```

дальнейшая запись описания обычная.

Следует помнить, что поля наследуются без какого-либо исключения. Поэтому, объявляя новые поля, необходимо следить за уникальностью их имен, иначе совпадение имени нового поля с именем наследуемого поля вызовет ошибку. На методы это правило не распространяется, но об этом ниже.

Пример 4. Опишем объектный тип «Вычислитель» с методами «сложение», «вычитание», «умножение», «деление» (некоторый исполнитель) и производный от него тип «Продвинутый вычислитель» с новыми методами «степень», «корень *n*-й степени».

```
Type BaseType=Double;
Vichislitel=Object
        A,B,C: BaseType;
        Procedure Init; {ввод или инициализация
                         полей}
        Procedure Slozh;
        Procedure Vich;
        Procedure Umn;
```

```
        Procedure Delen
        End;
    NovijVichislitel=Object(Vichislitel)
            N: Integer;
            Procedure Stepen;
            Procedure Koren
        End;
```

Обобщая вышесказанное, перечислим правила наследования:

• информационные поля и методы родительского типа наследуются всеми его типами-потомками независимо от числа промежуточных уровней иерархии;

• доступ к полям и методам родительских типов в рамках описания любых типов-потомков выполняется так, как будто бы они описаны в самом типе-потомке;

• ни в одном из типов-потомков не могут использоваться идентификаторы полей, совпадающие с идентификаторами полей какого-либо из родительских типов. Это правило относится и к идентификаторам формальных параметров, указанных в заголовках методов;

• тип-потомок может доопределить произвольное число собственных методов и информационных полей;

• любое изменение текста в родительском методе автоматически оказывает влияние на все методы порожденных типов-потомков, которые его вызывают;

• в противоположность информационным полям идентификаторы методов в типах-потомках могут совпадать с именами методов в родительских типах. При этом одноименный метод в типе-потомке подавляет одноименный ему родительский, и в рамках типа-потомка при указании имени такого метода будет вызываться именно метод типа-потомка, а не родительский.

Вызов наследуемых методов осуществляется согласно следующим принципам:

• при вызове метода компилятор сначала ищет метод, имя которого определено внутри типа объекта;

• если в типе объекта не определен метод с указанным в операторе вызова именем, то компилятор в поисках метода с таким именем поднимается выше к непосредственному родительскому типу;

• если наследуемый метод найден и его адрес подставлен, то следует помнить, что вызываемый метод будет работать так, как он определен и компилирован для родительского типа, а не для типа-потомка. Если этот наследуемый родительский тип вызывает еще и другие методы, то вызываться будут только родительские или вышележащие методы, так как вызовы методов из нижележащих по иерархии типов не допускаются.

Полиморфизм. Как отмечалось выше, полиморфизм (многообразие) предполагает определение класса или нескольких классов

методов для родственных объектных типов так, что каждому классу отводится своя функциональная роль. Методы одного класса обычно наделяются общим именем.

Пример 5. Пусть имеется родительский объектный тип «выпуклый четырехугольник» (поля типа «координаты вершин, заданные в порядке их обхода») и типы, им порожденные: параллелограмм, ромб, квадрат. Описать для указанных фигур методы «вычисление углов» (в градусах), «вычисление диагоналей», «вычисление длин сторон», «вычисление периметра», «вычисление площади».

```
Type BaseType=Double;
     FourAngle=Object
          x1,y1,x2,y2,x3,y3,x4,y4,
          A,B,C,D,D1,D2,
          Alpha,Beta,Gamma,Delta,
          P,S: BaseType;
          Procedure Init;
          Procedure Storony;
          Procedure Diagonali;
          Procedure Angles;
          Procedure Perimetr;
          Procedure Ploshad;
          Procedure PrintElements;
        End;
      Parall=Object(FourAngle)
          Procedure Storony;
          Procedure Perimetr;
          Procedure Ploshad;
        End;
      Romb=Object(Parall)
          Procedure Storony;
          Procedure Perimetr;
        End;
      Kvadrat=Object(Romb)
          Procedure Angles;
          Procedure Ploshad;
        End;
      Procedure FourAngle.Init;
      Begin
        Write('Введите координаты вершин заданного
             четырехугольника:');
        ReadLn(x1,y1,x2,y2,x3,y3,x4,y4);
     End;
      Procedure FourAngle.Storony;
      Begin A:=Sqrt(Sqr(x2-x1)+Sqr(y2-y1));
            B:=Sqrt(Sqr(x3-x2)+Sqr(y3-y2));
```

```
            C:=Sqrt(Sqr(x4-x3)+Sqr(y4-y3));
            D:=Sqrt(Sqr(x4-x1)+Sqr(y4-y1));
      End;
      Procedure FourAngle.Diagonali;
      Begin
        D1:=Sqrt(Sqr(x1-x3)+Sqr(y1-y3));
        D2:=Sqrt(Sqr(x2-x4)+Sqr(y2-y4));
      End;
      Procedure FourAngle.Angles;
        Function Ugol(Aa,Bb,Cc: BaseType):
                    BaseType;
        Var VspCos, VspSin: BaseType;
     Begin
        VspCos:=(Sqr(Aa)+Sqr(Bb)-Sqr(Cc))/(2*Aa*Bb);
        VspSin:=Sqrt(1-Sqr(VspCos));
        If Abs(VspCos)>1e-7
        Then Ugol:=(ArcTan(VspSin/VspCos)
                    +Pi*Ord(VspCos<0))/Pi*180
        Else Ugol:=90
    End;
  Begin Alpha:=Ugol(D,A,D2); Beta:=Ugol(A,B,D1);
     Gamma:=Ugol(B,C,D2); Delta:=Ugol(C,D,D1);
  End;
  Procedure FourAngle.Perimetr;
  Begin P:=A+B+C+D End;
  Procedure FourAngle.Ploshad;
  Var Per1, Per2: BaseType;
  Begin Per1:=(A+D+D2)/2; Per2:=(B+C+D1)/2;
    S:=Sqrt(Per1*(Per1-A)*(Per1-D)*(Per1-D2)) +
       Sqrt(Per2*(Per2-B)*(Per2-C)*(Per2-D1))
  End;
  Procedure FourAngle.PrintElements;
  Begin
 WriteLn('Стороны:',A:10:6,B:10:6,C:10:6,
         D:10:6,'Углы:',Alpha:10:4,Beta:10:4,
         Gamma:10:4,Delta:10:4,'Периметр:',
         P:10:6,'Площадь:',S:10:6,'Диагонали:',
         D1:10:6,D2:10:6)
 End;
 Procedure Parall.Storony;
 Begin A:=Sqrt(Sqr(x2-x1)+Sqr(y2-y1));
       B:=Sqrt(Sqr(x3-x2)+Sqr(y3-y2));
       C:=A; D:=B
 End;
 Procedure Parall.Perimetr;
 Begin P:=2*(A+B) End;
```

```
Procedure Parall.Ploshad;
Var Per: BaseType;
Begin Per:=(A+D+D2)/2;
        S:=2*Sqrt(Per*(Per-A)*(Per-D)*(Per-D2))
End;
Procedure Romb.Storony;
Begin
    A:=Sqrt(Sqr(x2-xl)+Sqr(y2-yl));
    B:=A; C:=A; D:=A
End;
Procedure Romb.Perimetr;
Begin P:=2*A End;
Procedure Kvadrat.Angles;
Begin Alpha:=90; Beta:=90; Gamma:=90;
      Delta:=90;
End;
Procedure Kvadrat.Ploshad;
Begin  S:=Sqr(A)
End;
 {Основная программа}
Var obj: Kvadrat;
Begin
          obj.Init;
          obj.Storony;
          obj.Diagonali;
          obj.Angles;
          obj.Perimetr;
          obj.Ploshad;
          obj.PrintElements
    End.
```

Таким образом, вычисление соответствующего элемента в фигуре, если это действие по сравнению с другими является уникальным, производится обращением к своему методу.

3.24. Виртуальные методы. Конструкторы и деструкторы

Объекты в динамической памяти. При работе с объектами довольно типичной является ситуация, когда сложный метод приходится создавать заново для каждого типа объекта, хотя различия в поведении объектов могут быть небольшими. В этом случае обычно создается общий сложный метод, а различия вносятся в сменные подчиненные методы. Реализация такого подхода осуществляется с помощью *виртуальных* подчиненных методов. С этой целью после заголовка *каждого* сменного метода требуется напи-

сать `Virtual`. Заголовки виртуальных методов предка и потомка должны в точности совпадать, причем оба метода должны быть виртуальными. Отсюда следует, что при проектировании методов следует учесть, что некоторые из них потребуют дальнейшего развития, и объявить их виртуальными.

И наконец, инициализация экземпляра объекта должна выполняться методом особого вида, который называется *конструктор.* Обычно на конструктор возлагается присвоение полям исходных значений, открытие файлов, первоначальный вывод на экран и т.д. Помимо действий, заложенных в него программистом, конструктор выполняет подготовку так называемого механизма позднего связывания виртуальных методов. Отсюда следует, что до вызова любого виртуального метода должен быть выполнен какой-либо конструктор. Структура конструктора такая же, как и у любой процедуры, только вместо слова `Procedure` в заголовке метода пишется слово `Constructor`.

В паре с конструктором всегда существует и *деструктор,* роль которого противоположна роли конструктора, — он выполняет действия, завершающие работу с объектом: закрывает файлы, очищает динамическую память, осуществляет восстановление некоторых состояний, предшествующих работе с объектом и т.д. Вообще деструктор может не выполнять никаких действий, но обязательно должен присутствовать в списке методов объекта. Заголовок метода-деструктора начинается со слова `Destructor`, в остальном же его структура такая же, как и у любого другого метода.

Скажем теперь несколько слов о механизме раннего и позднего связывания. Раннее связывание предполагает, что связь между методами устанавливается во время трансляции программы, в то время как позднее связывание предусматривает динамическую связь, т.е. реализуемую по мере необходимости в процессе выполнения программы. За счет такого механизма и удается правильно установить все нужные связи для виртуальных методов. Результат позднего связывания в этом случае зависит от типа того объекта, чей метод обратился к виртуальному методу. Конструктор для подготовки позднего связывания устанавливает связь между экземпляром объекта и таблицей виртуальных методов (VMT) объекта. Для каждого виртуального метода VMT содержит его адрес. Вызов виртуального метода делается не прямо, а через VMT: сначала по имени метода определяется его адрес, а затем по этому адресу передается управление. Именно по этим причинам использованию виртуальных методов должен предшествовать вызов конструктора.

Чтобы разместить объект в динамической памяти, надо описать указатель на него. Выделение памяти для динамического объекта выполняется процедурой `NEW`. Поскольку сразу после этого производится инициализация объекта, то для объектов процедура `NEW` выглядит так:

```
NEW(<указатель на объект>, <конструктор>)
```

Высвобождение динамической памяти, занятой объектом, выполняется процедурой DISPOSE. Перед этим выполняются действия, завершающие работу с объектом, поэтому для объектов процедура DISPOSE выглядит так:

```
DISPOSE(<указатель на объект>, <деструктор>)
```

Нельзя освободить память, занятую динамическим объектом, если у него нет деструктора, хотя бы и пустого.

Пример 6. Сложное меню в динамической памяти. Построим сложное иерархическое меню: пробел будет открывать главное меню, нажатие на клавишу **Enter** будет разворачивать подсвеченный пункт в меню или, если пункт находится в подменю, клавиша **Enter** будет сворачивать подменю. Работа будет завершаться по нажатию на клавишу **Esc**. Нижний уровень меню располагаем вертикально, главное меню — горизонтально. (В роли пунктов главного меню будут выступать некоторые классы языков программирования, а в качестве подпунктов — примеры таких языков.)

```
Program Menu;
Uses crt;
Const Stroka='Операциональное(Фортран(b) Бейсик(c)
      Си(d)) Структурное(Паскаль(e) Модула(f))
      Объектное(СИ++(g) Делфи(h))';
Type PMenu=^TMenu;
TMenu=Object
     x,y:Integer;{координаты меню}
     Items:String;{строка названий и возвращаемых
символов}
     MaxItem:Integer;{общее количество пунктов}
     sel:Integer;{номер отмеченного пункта}
Constructor
Init(ax,ay:Integer;TItems:String);{заполняет поле
Items, подсчитывает количество пунктов,делает
выбранным первый пункт}
Procedure Select(Var w:Char);Virtual;{позволяет
выбрать пункт меню и возвращает символ выбранного
пункта. При отказе от выбора возвращает #27}
Procedure Draw;Virtual;{рисует меню, выделяя
выбранный пункт цветом}
Function
LeftBoard(Item:Integer):Integer;{возвращает начало
названия пункта Item в строке Items}
Function Len(Item:Integer):Integer;{возвращает
длину названия пункта Item в строке Items}
Function WhatSel:Char;{возвращает символ выбранного
```

```
пункта}
End;{Конец описания TMenu}
Type PNeatMenu=^TNeatMenu;
  TNeatMenu=Object(TMenu)
  Store:Array[1..4000] Of Byte;
  Constructor
  Init(aX,aY:Integer;TItems:String);{сохраняет
  экран и вызывает Init предка}
  Destructor Done;{восстанавливает состояние
  экрана}
  End;
Type
   PVertMenu=^TVertMenu;
   TVertMenu=Object(TNeatMenu)
   Procedure Draw;Virtual;
   End;
Type PCompMenu=^TCompMenu;
   TCompMenu=Object(TNeatMenu)
   SubMenus:Array[1..10] Of String;
   Constructor Init(aX,aY:Integer;TItems:String);
   Procedure Select(var w:Char);Virtual;
   End;
{Описания методов}
Constructor
TMenu.Init(ax,ay:Integer;TItems:String);
Var i:Integer;
Begin
  x:=ax; y:=ay;
  Items:=TItems;
  Sel:=1;
  maxItem:=0;
  For i:=1 To length(Items) Do
  If Items[i]='('then inc(maxItem);
  Draw;
End;
Constructor TNeatMenu.Init(aX,aY:Integer;
TItems:String);
Begin
  move(mem[$b800:0],Store,4000);
  TMenu.Init(aX,aY,TItems);
End;
Constructor TCompMenu.Init(aX,aY:Integer;
TItems:String);
Var i,br:Integer;
Begin
  x:=aX;
```

```
  y:=aY;
  maxItem:=0;
  br:=0;
  Items:='';
  For i:=1 To length(TItems) Do
  Begin
   Case TItems[i] Of
  '(':Begin
      inc(br);
      If br=1 Then
      Begin
         inc(maxItem);
         SubMenus[maxItem]:='';
         End
         Else
         SubMenus[maxItem]:=SubMenus[maxItem]+'(';
      End;
  ')':Begin
         dec(br);
         If br=1 Then
         SubMenus[maxItem]:=SubMenus[maxItem]+')'
      Else
         If TItems[i-1]=')' Then Items:=Items+'()'
         Else Items:=Items+'('+TItems[i-1]+')';
      End;
      Else
      If br=0 Then Items:=Items+TItems[i]
      Else
      SubMenus[maxItem]:=SubMenus[maxItem]+TItems[i];
  End;{case}
  End;{for}
  Sel:=1;
  move(mem[$b800:0],Store,4000);
End;
Procedure TCompMenu.Select(Var w:Char);
Var wSub,c:Char;
    SubMenu:PNeatMenu;
    x1:Integer;
Begin
  Sel:=0;
  w:=#0;
  Repeat
  c:=readkey;
  Case c Of
  '':Begin
       inc(Sel);
```

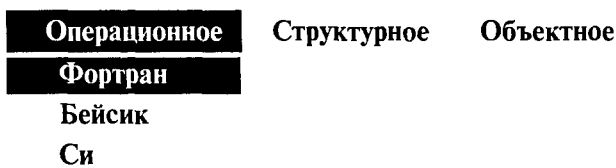
```
        If Sel>maxItem Then Sel:=1;
        Draw;
      End;
   #13:If pos('(',SubMenus[Sel])=0 Then w:=WhatSel
      Else
       Begin
            x1:=x+LeftBoard(Sel);
            If pos(')',SubMenus[Sel])=0
            Then SubMenu:=new(PVertMenu,
                        Init(x1,y+1,SubMenus[Sel]))
        Else SubMenu:=new(PCompMenu,Init(x1,y+1,
                              SubMenus[Sel]));
            SubMenu^.Select(wSub);
            dispose(SubMenu,Done);
            If wSub=#27 Then w:=wSub;
       End;
    #27: w:=#27;
    #0:  c:=readkey;
   End;{case}
   Until w<>#0;
End;
Destructor TNeatMenu.Done;
Begin
  move(Store,mem[$b800:0],160*25);
End;
Procedure TMenu.Select(Var w:Char);
Var c:Char;
Begin
 w:=#0;
 Repeat
   c:=readkey;
   Case c Of
   ' ':Begin
            inc(Sel);
            If Sel>maxItem Then Sel:=1;
            draw;
       End;
    #13:w:=whatSel;{enter}
    #27:w:=#27;{esc}
    #0:c:=readkey
  End;{case}
  Until w<>#0;
End;
Procedure TMenu.Draw;
Var a,b,Xnext,Item:Integer;
Begin
```

```
  Xnext:=X;
  For Item:=1 To maxItem Do
   Begin
      gotoXY(Xnext,Y);{перемещает курсор в данные
      координаты}
      If Item=Sel Then TextColor(Yellow)
        {выбор цвета символов}
      Else TextColor(White);
      a:=Leftboard(item);
      b:=len(item);
      Write(copy(Items,a,b));{копирует из строки
      items b символов, начиная с символа с номером a}
      inc(Xnext,Len(Item)+2)
   End;{for}
 gotoXY(80,25);
End;
Procedure TVertMenu.Draw;
Var Item:Integer;
Begin
   For Item:=1 To maxItem Do
   Begin
   gotoXY(x,y+Item-1);
   If Item=Sel Then TextColor(Yellow)
        Else TextColor(White);
   Write(copy(Items,LeftBoard(Item),Len(Item)));
   End;{for}
   gotoXY(80,25);
End;
Function TMenu.LeftBoard(Item:Integer):Integer;
Var  a,i:Integer;
Begin
  i:=1;
  a:=1;
  While i<length(Items) Do
  Begin
   If Item=a Then
    Begin
       leftBoard:=i;
       i:=length(Items);
    End
   Else
    Begin
       i:=i+len(a)+3;
       inc(a);
    End;
  End;{while}
```

```
End;
Function TMenu.Len(Item:Integer):Integer;
Var i:Integer;
Begin
    i:=LeftBoard(Item);
    While Items[i]<>'(' Do inc(i);
    Len:=i-leftboard(item);
End;
Function TMenu.WhatSel:Char;
Begin
    WhatSel:=Items[LeftBoard(Sel)+len(sel)+1];
End;
{Основная программа}
Var M:PCompMenu;
    n:Char;
Begin clrScr;new(M,Init(10,3,Stroka));
    M^.Select(n);dispose(M,Done)
End.
```

При исполнении этой программы появляется пустой экран. После нажатия на клавишу **пробел** на экране появится строка из трех пунктов главного меню:

Желтым цветом выделен первый пункт меню. Если нажать на клавишу **Enter**, то появится вертикальное подменю:



с выделенным цветом первым пунктом. Смена выбранного пункта (продвижение по пунктам) производится при помощи клавиши **пробел**. Нажатием на клавишу **Enter** закрывается подменю. Выход из программы выполняется при помощи клавиши **Esc**.

Упражнения

1. Описать объект с указанием его методов:

а) телефонный звонок (номер телефона, дата разговора, продолжительность, код города и т.д.);

б) поездка (номер поезда, пункт назначения, дни следования, время прибытия, время стоянки и т.д.);

в) кинотеатр (название кинофильма, сеанс, стоимость билета, количество зрителей и т.д.).

Реализовать объявленные в объектном типе методы.

2. Дополнить объект Drob в примере 3 методами «положительная дробь» (функция логического типа), «правильная дробь» (функция логического типа), «конечная десятичная дробь» (функция логического типа), «период дроби» (существует, если соответствующая десятичная дробь не является конечной, в общем случае результат — «длинное» число); реализовать и протестировать эти методы.

3. Описать объекты-потомки для объектов из задания 1. Реализовать объявленные в объектном типе методы.

4. Реализовать методы «вычислителей» из примера 4.

5. Дополнить набор объектов из примера 5 объектом «прямоугольник», а также другими выпуклыми четырехугольниками (трапеция и т.д.), не являющимися параллелограммами, и реализовать соответствующие методы.

6. Выполнить другой вариант проектирования и реализации объектов и их методов, используя для одинаковых действий виртуальные методы.

7. Реализовать примеры 3—5, создавая объекты в динамической памяти.

8. Построить в каждом случае иерархию объектов. Последовательно применить для объектов методы отобразить, сдвинуть, изменить размеры, спрятать (задача взята на http://petrsu.karelia.ru/Structure/Deps/IMO/pascal/theory/1.htm):

а) координаты — точка — горизонтальная линия — горизонтально-вертикальное перекрестье;

б) координаты — точка — наклонная под углом 45° линия — наклонное перекрестье;

в) координаты — точка — окружность — дуга (процедура Arc);

г) координаты — точка — эллипс (процедура FillEllipse) — эллиптическая дуга (процедура Ellipse);

д) координаты — точка — окружность — сектор (процедура PieSlice);

е) координаты — точка — сектор (процедура PieSlice) — эллиптическая дуга (процедура Ellipse);

ж) координаты — точка — прямоугольник (процедура Rectangle) — трехмерная полоса (процедура Bar3D);

з) координаты — точка — заштрихованный эллипс (процедура FillEllipse) — заштрихованный сектор (процедура Sector);

и) координаты — точка — окружность — заштрихованный сектор (процедура Sector);

к) координаты — точка — окружность — эллиптическая дуга (процедура Ellipse).

9. Реализовать объекты и их методы из примера 1 в графическом режиме.

ГЛАВА 4. ЯЗЫК ПРОГРАММИРОВАНИЯ СИ++

4.1. Введение в Си и Си++

История языка программирования С++. Вторым языком программирования, который предлагается для изучения в данном пособии, является язык Си (в английском варианте его название обозначается одной заглавной буквой латинского алфавита — *C*). Язык Си был создан в 1972 г. сотрудником фирмы Bell Laboratories в США Денисом Ритчи.

По замыслу автора, язык Си должен был обладать противоречивыми свойствами. С одной стороны, это язык программирования высокого уровня, поддерживающий методику структурного программирования (подобно Паскалю). С другой стороны, этот язык должен обеспечивать возможность создавать такие системные программы, как компиляторы и операционные системы. До появления Си подобные программы писались исключительно на языках низкого уровня — Ассемблерах, Автокодах. Первым системным программным продуктом, разработанным с помощью Си, стала операционная система UNIX. Из-за упомянутой выше двойственности свойств нередко в литературе язык Си называют языком среднего уровня. Стандарт Си был утвержден в 1983 г. Американским национальным институтом стандартов (ANSI) и получил название ANSI C.

В начале 1980-х гг. в той же фирме Bell Laboratories ее сотрудником Бьерном Строуструпом было разработано расширение языка Си, предназначенное для объектно-ориентированного программирования. По сути дела, был создан новый язык, первоначально названный «Си с классами», а позднее (в 1983 г.) получивший название Си++ (Си-плюс-плюс). Язык Си++ принято считать языком объектно-ориентированного программирования. Однако этот язык как подмножество включает в себя Си и по-прежнему сохраняет свойства языка для системного программирования. Все существующие версии трансляторов для Си++ поддерживают стандарт ANSI C.

Из сказанного выше следует, что язык Си++ поддерживает как процедурную, так и объектно-ориентированную парадигмы программирования. Последующий материал пособия в большей степени посвящен процедурному программированию на Си++ и лишь в разд. 4.10 приводится краткое введение в ООП на Си++, подобно тому как это сделано в гл. 3, посвященной Паскалю.

Примеры программ. Для того чтобы получить первоначальное представление о программировании на Си/Си++, рассмотрим несколько примеров.

В литературе по программированию стало традицией приводить в качестве примера первой программы на Си следующий текст.

Пример 1.

```
/* Ваша первая программа на Си */
#include <stdio.h>
void main()
{
  printf("\n Здравствуй, Мир!\n");
}
```

Здесь первая строка представляет собой комментарий. Начало и конец комментария ограничиваются парами символов `/*` и `*/` Все, что расположено между этими символами, компилятор игнорирует.

Вторая строка программы содержит *директиву препроцессора:*

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

Она сообщает компилятору информацию о необходимости подключить к тексту программы содержимое файла `stdio.h`, в котором находится описание (прототип) библиотечной функции `printf()` — функции вывода на экран.

Вся последующая часть программы называется *блоком описания главной функции.* Она начинается с заголовка главной функции:

```
void main()
```

Любая программа на Си обязательно содержит главную функцию, имя которой — `main`. Слово `void` обозначает то, что главная функция не возвращает никаких значений в результате своего выполнения, а пустые скобки обозначают отсутствие аргументов. Тело главной функции заключается между парой фигурных скобок, следующих после заголовка.

Текст программы содержит всего лишь один исполняемый оператор — это оператор вывода на экран. Вывод осуществляется путем обращения к стандартной библиотечной функции `printf()`. В результате его выполнения на экран выведется текст:

```
Здравствуй, Мир!
```

Впереди данной строки и после нее будет пропущено по одной пустой строке, что обеспечивается наличием управляющих символов `\n`.

Следующий пример содержит программу, выполняющую те же самые действия, но написанную на Си++.

Пример 2.

```
// Ваша первая программа на Си++
#include <iostream.h>
void main()
{
   cout"\nЗдравствуй Мир!\n";
}
```

Первое отличие от программы из примера 1 состоит в форме комментария. В Си++ можно использовать *строчный комментарий*, который начинается с символов // и заканчивается концом строки. Для организации вывода на экран к программе подключается специальная *библиотека объектов*, заголовочный файл которой имеет имя `iostream.h`. Вывод осуществляется посредством объекта `cout` из этой библиотеки.

В примере 1 используется механизм *форматного ввода/вывода*, характерный для Си. В примере 2 работает механизм *потокового ввода/вывода*, реализованный в Си++. Преемственность Си++ по отношению к Си выражается в том, что программа из примера 1 будет восприниматься компилятором Си++, т.е. эта программа исполнима в любой системе программирования, ориентированной на Си++.

Рассмотрим еще один пример программы на Си/Си++. Сопоставим ее с аналогичной программой на Паскале.

Пример 3. Деление простых дробей (см. разд. 3.1):

$$a/b : c/d = m/n$$

Паскаль

```
Program Example_3;
Var a,b,c,d,
m,n: Integer;
Begin
Write("a="); ReadLn(a);
Write("b="); ReadLn(b);
Write("c="); ReadLn(c);
Write("d="); ReadLn(d);
m:=a*d;
n:=b*c;
WriteLn("m=",m);
WriteLn("n=",n)
End.
```

Си

```
#include <stdio.h>
void main()
{
 int a,b,c,m,n;
 printf("\na=");
 scanf("%d",&a);
 printf("\nb=");
 scanf("%d",&b);
 printf("\nc=");
 scanf("%d",&c);
 printf("\nd=");
 scanf("%d",&d);
 m=a*d;
 n=b*c;
 printf("\nm=%d",m);
 printf("\nn=%d",n);
}
```

В этом примере появился целый ряд новых элементов по сравнению с предыдущим. Первая строка в теле главной функции является

объявлением пяти переменных целого типа — `int`. Далее наряду с уже знакомым оператором форматного вывода на экран используется оператор форматного ввода с клавиатуры — `scanf()`. Это также стандартная функция из библиотеки ввода/вывода, подключаемая к программе с помощью файла `stdio.h`. Первый аргумент этой функции `%d` является спецификацией формата вводимых значений. В данном случае он указывает на то, что с клавиатуры будет вводиться целое число. Перед именем вводимой переменной принято писать символ `&`. Это необходимо делать для правильной работы функции `scanf()`. Смысл данного символа будет пояснен позже. В отличие от Паскаля в качестве знака присваивания используется символ =. Однако читать его надо как «присвоить». Спецификации формата `%d` используются и при организации вывода на экран целых чисел с помощью функции `printf()`.

Этапы работы с программой на Си++ в системе программирования (рис. 41 — прямоугольниками отображены системные программы, а блоки с овальной формой обозначают файлы на входе и на выходе этих программ).

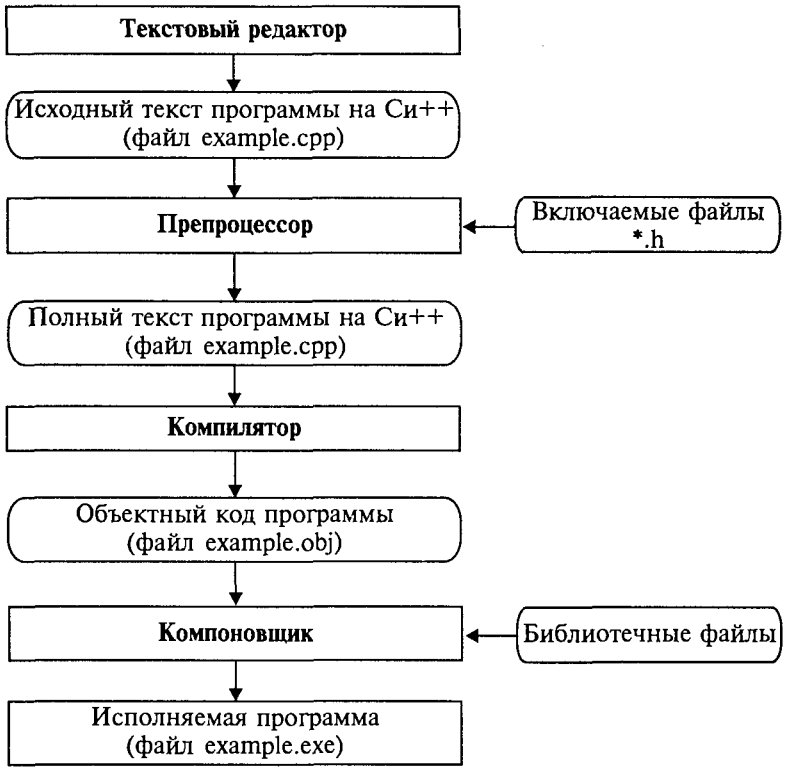


Рис. 41

1. С помощью *текстового редактора* формируется текст программы и сохраняется в файле с расширением `cpp`. Пусть, например, это будет файл с именем `example.cpp`.

2. Осуществляется этап *препроцессорной обработки*, содержание которого определяется директивами препроцессора, расположенными перед заголовком программы (функции). В частности, по директиве `#include` препроцессор подключает к тексту программы заголовочные файлы (*.h) стандартных библиотек.

3. Происходит *компиляция текста* программы на Си++. В ходе компиляции могут быть обнаружены синтаксические ошибки, которые должен исправить программист. В результате успешной компиляции получается *объектный код программы* в файле с расширением `obj`. Например, `example.obj`.

4. Выполняется этап *компоновки* с помощью системной программы Компоновщик (`Linker`). Этот этап еще называют *редактированием связей*. На данном этапе к программе подключаются библиотечные функции. В результате компоновки создается исполняемая программа в файле с расширением `exe`. Например, `example.exe`.

4.2. Элементы языка Си++

Алфавит. В алфавит языка Си++ входят:

- латинские буквы: от *a* до *z* (строчные) и от *A* до *Z* (прописные);
- десятичные цифры: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`;
- специальные символы: `" { } , | [ ] ( ) + – / % \ ; ': ? < = > _ ! & # ~ ^ . *`

К специальным символам относится также пробел.

В комментариях, строках и символьных константах могут использоваться и другие знаки (например, русские буквы).

Комбинации некоторых символов, не разделенных пробелами, интерпретируются как один значимый символ. К ним относятся:

```
++ —— == && || << >> >= <= += –= *= /= ?: /* */ //
```

В Си++ в качестве ограничителей комментариев могут использоваться как пары символов /* и */, принятые в языке Си, так и символы //, используемые только в Си++. Признаком конца такого комментария является невидимый символ перехода на новую строку. Примеры:

```
/* Это комментарий, допустимый в Си и Си++ */
//Это строчный комментарий, используемый только в
Си++
```

Из символов алфавита формируются лексемы — единицы текста программы, которые при компиляции воспринимаются как единое целое и не могут быть разделены на более мелкие элементы. К лексемам относятся идентификаторы, служебные слова, константы, знаки операций, разделители.

Идентификаторы. Последовательность латинских букв, цифр, символов подчеркивания (_), начинающаяся с буквы или символа подчеркивания, является идентификатором. Например:

```
B12     rus    hard_RAM_disk     MAX    ris_32
```

В отличие от Паскаля в Си/Си++ различаются прописные и строчные буквы. Это значит, что, например, `flag`, `FLAG`, `Flag`, `FlAg` — разные идентификаторы.

Ограничения на длину идентификатора могут различаться в разных реализациях языка. Компиляторы фирмы Borland позволяют использовать до 32 первых символов имени. В некоторых других реализациях допускаются идентификаторы длиной не более 8 символов.

Служебные (ключевые) слова. Как и в Паскале, служебные слова в Си — это идентификаторы, назначение которых однозначно определено в языке. Они не могут быть использованы как свободно выбираемые имена. Полный список служебных слов зависит от реализации языка, т. е. различается для разных компиляторов. Однако существует неизменное ядро, которое определено стандартом Си++. Вот его список:

```
asm         else        operator    template
auto        enum        private     this
break       extern      protected   throw
case        float       public      try
catch       for         register    typedef
char        friend      return      typeid
class       goto        short       union
const       if          signed      unsigned
continue    inline      sizeof      virtual
default     int         static      void
delete      long        struct      volatile
do          new         switch      while
double
```

Дополнительные к этому списку служебные слова приведены в описаниях конкретных реализаций Си++. Некоторые из них начинаются с символа подчеркивания, например: _export, _ds, _AH и др. Существуют служебные слова, начинающиеся с двойного подчеркивания. В связи с этим программисту не рекомендуется использовать в своей программе идентификаторы, начинающиеся с одного или двух подчеркиваний, во избежание совпадения со служебными словами.

4.3. Типы данных

Концепция типов данных является важнейшей стороной любого языка программирования. Особенность Паскаля состоит в большом разнообразии типов, схематически представленном в разд. 3.4 на рис. 9. Аналогичная схема для языка Си++ представлена на рис. 42.

Сравнение схем приводит к выводу о том, что разнообразие типов данных в Си++ меньше, чем в Турбо Паскале.

В Си/Си++ имеется четыре базовых арифметических (числовых) типа данных. Из них два целочисленных — `char`, `int` — и два плавающих (вещественных) — `float` и `double`. Кроме того, в программах можно использовать некоторые модификации этих типов, описываемых с помощью служебных слов — модификаторов. Существуют два модификатора размера — `short` (короткий) и `long` (длинный) — и два модификатора знаков — `signed` (знаковый) и `unsigned` (беззнаковый). Знаковые модификаторы применяются только к целым типам.

Как известно, тип величины связан с ее формой внутреннего представления, множеством принимаемых значений и множеством операций, применимых к этой величине. В табл. 4.1 перечислены арифметические типы данных Си++, указан объем занимаемой памяти и диапазон допустимых значений.

Размер типа `int` и `unsigned int` зависит от размера слова операционной системы, в которой работает компилятор Си++. В 16-разрядных ОС (MS DOS) этим типам соответствуют 2 байта, в 32-разрядных (Windows) — 4 байта.

Рис. 42

Таблица 4.1

Тип данных	Размер (байт)	Диапазон значений	Эквивалентные названия типа
`char`	1	−128...+127	`signed char`
`int`	2/4	зависит от системы	`signed, signed int`
`unsigned char`	1	0...255	нет
`unsigned int`	2/4	зависит от системы	`unsigned`
`short int`	2	−32768...32767	`short, signed short int`
`unsigned short`	2	0...65535	`unsigned short int`
`long int`	4	−2147483648...2147483647	`long, signed long int`
`unsigned long int`	4	0...4294967295	`unsigned long`
`float`	4	±(3.4E−38...3.4E+38)	нет
`double`	8	±(1.7E−308...1.7E+308)	нет
`long double`	10	±(3.4E−4932...1.1E+4932)	нет

Анализируя данные табл. 4.1, можно сделать следующие выводы:

- если не указан базовый тип, то по умолчанию подразумевается `int`;
- если не указан модификатор знаков, то по умолчанию подразумевается `signed`;
- с базовым типом `float` модификаторы не употребляются;
- модификатор `short` применим только к базовому типу `int`.

Программисту, работавшему на Паскале, покажется странным, что тип `char` причислен к арифметическим типам. Ведь даже его имя указывает на то, что это символьный тип! В Си/Си++ величины типа `char` могут рассматриваться в программе и как символы, и как целые числа. Все зависит от контекста, т.е. от способа использования этой величины. В случае интерпретации величины типа `char` как символа ее числовое значение является ASCII-кодом. Следующий пример иллюстрирует сказанное.

```
char a=65;
printf("%c",a);/*На экране появится символ A*/
printf("%d",a);/*На экране появится число 65*/
```

Символы "`%c`" являются спецификацией формата ввода/вывода символьных данных, а "`%d`" — спецификацией для целых чисел.

Еще одной особенностью Си, которая может удивить знатоков Паскаля, является отсутствие среди базовых типов логического типа данных. Между тем, как мы дальше увидим, в Си используются логические операции и логические выражения. В качестве логических величин в Си/Си++ выступают целые числа. Интерпретация их значений в логические величины происходит по правилу: *равно нулю — ложь* (в Паскале — `false`), *не равно нулю — истина* (в Паскале — `true`).

В последние версии Си++ добавлен отдельный логический тип с именем `bool`. Его относят к разновидности целых типов данных.

Описание переменных в программах на Си/Си++ имеет вид:

```
имя_типа список_переменных;
```

Примеры описаний:

```
char symbol,cc;
unsigned char code;
int number,row;
unsigned long long_number;
float x,X,cc3;
double e,b4;
long double max_num;
```

Одновременно с описанием можно задать начальные значения переменных. Такое действие называется *инициализацией переменных*. Описание с инициализацией производится по следующей схеме:

```
тип имя_переменной = начальное_значение
```

Например:

```
float pi=3.14159,c=1.23;
unsigned int year=2000;
```

Константы. ***Запись целых констант.*** *Целые десятичные числа,* начинающиеся не с нуля, например: 4, 356, —128.

Целые восьмеричные числа, запись которых начинается с нуля, например: 016, 077.

Целые шестнадцатеричные числа, запись которых начинается с символов 0x, например: 0x1A, 0x253, 0xFFFF.

Тип константы компилятор определяет по следующим правилам: если значение константы лежит в диапазоне типа `int`, то она получает тип `int`; в противном случае проверяется, лежит ли константа в диапазоне типа `unsigned int`, в случае положительного ответа она получает этот тип; если не подходит и он, то пробуется тип `long` и, наконец, `unsigned long`. Если значение числа не укладывается в диапазон типа `unsigned long`, то возникает ошибка компиляции.

Запись вещественных констант. Если в записи числовой константы присутствует десятичная точка (`2.5`) или экспоненциальное расширение (`1E-8`), то компилятор рассматривает ее как вещественное число и ставит ей в соответствие тип `double`. Примеры вещественных констант: `44. 3.14159 44E0 1.5E-4`.

Использование суффиксов. Программист может явно задать тип константы, используя для этого суффиксы. Существуют три вида суффиксов: `F(f)` – `float`; `U(u)` – `unsigned`; `L(l)` – `long` (для целых и вещественных констант). Кроме того, допускается совместное использование суффиксов `U` и `L` в вариантах `UL` или `LU`.

Примеры:

`3.14159F` — константа типа `float`, под которую выделяется 4 байта памяти;

`3.14L` — константа типа `long double`, занимает 10 байт;

`50000U` — константа типа `unsigned int`, занимает 2 байта памяти (вместо четырех без суффикса);

`0LU` — константа типа `unsigned long`, занимает 4 байта;

`24242424UL` — константа типа `unsigned long`, занимает 4 байта.

Запись символьных и строковых констант. Символьные константы заключаются в апострофы. Например: `'A','a','5','+'`. Строковые константы, представляющие собой символьные последовательности, заключаются в двойные кавычки. Например: `"rezult"`, `"введите исходные данные"`.

Особую разновидность символьных констант представляют так называемые *управляющие символы*. Их назначение — управление выводом на экран. Как известно, такие символы расположены в начальной части кодовой таблицы ASCII (коды от 0 до 31) и не имеют графического представления. В программе на Си они изображаются парой символов, первый из которых `'\'`. Вот некоторые из управляющих символов:

`'\n'` — переход на новую строку;

`'\t'` — горизонтальная табуляция;

`'\a'` — подача звукового сигнала. Полный список управляющих символьных последовательностей будет дан позднее.

Управляющие символьные последовательности являются частным случаем *эскейп-последовательностей* (`ESC-sequence`), с помощью которых можно задать символьную константу указанием ее кода. Код символа можно указать в восьмеричном или в шестнадцатеричном представлении. Формат восьмеричного представления: `'\ddd'`. Здесь `d` — восьмеричная цифра (от 0 до 7). Формат шестнадцатеричного представления: `'\xhh'` (или `'\Xhh'`), где `h` — шестнадцатеричная цифра (от 0 до F). Например, константа, соответствующая заглавной латинской букве *A*, может быть представлена тремя способами: `'A'`, `'\101'`, `'\x41'`.

Именованные константы (константные переменные). Как и в Паскале, в программе на Си/Си++ могут использоваться именованные константы. Употребляемое для их определения служебное слово **const** принято называть *квалификатором доступа.* Квалификатор `const` указывает на то, что данная величина не может изменяться в течение всего времени работы программы. В частности, она не может располагаться в левой части оператора присваивания. Примеры описания константных переменных:

```
const float pi=3.14159;
const int iMIN=1, iMAX=1000;
```

Определение констант на стадии препроцессорной обработки программы. Еще одной возможностью ввести именованную константу является использование препроцессорной директивы **#define** в следующем формате:

```
#define <имя константы> <значение константы>
```

Например:

```
#define iMIN 1
#define iMAX 1000
```

Тип констант явно не указывается и определяется по форме записи. В конце директивы не ставится точка с запятой.

На стадии препроцессорной обработки указанные имена заменяются на соответствующие значения. Например, если в программе присутствует оператор

```
X=iMAX-iMIN;
```

то в результате препроцессорной обработки он примет вид:

```
X=1000-1;
```

При этом идентификаторы `iMAX` и `iMIN` не требуют описания внутри программы.

Константы перечисляемого типа. Данное средство языка позволяет определять последовательность целочисленных именованных констант. Описание перечисляемого типа начинается со служебного слова **enum**, а последующий список констант заключается в фигурные скобки. Например:

```
enum {A,B,C,D};
```

В результате имени `A` будет сопоставлена константа `0`, имени `B` — константа `1`, `C` — `2`, `D` — `3`. По умолчанию значение первой константы равно нулю.

Для любой константы можно явно указать значение. Например:

```
enum {A=10,B,C,D};
```

В результате будут установлены следующие соответствия: `A=10, B=11, C=12, D=13.`

Возможен и такой вариант определения перечисления:

```
enum {A=10,B=20,C=35,D=100};
```

Если перечисляемому типу дать имя, то его можно использовать в описании переменных. Этот вариант аналогичен использованию перечисляемого типа данных в Паскале (см. разд. 3.4). Например:

```
enum metal {Fe,Co,Na,Cu,Zn};
metal Met1,Met2;
```

Здесь идентификатор `metal` становится именем типа. После такого описания в программе возможны следующие операторы:

```
Met1=Na; Met2=Zn;
```

4.4. Операции и выражения

Во всех языках программирования под выражением подразумевается конструкция, составленная из констант, переменных, знаков операций, функций, скобок. Выражение определяет порядок вычисления некоторого значения. Если это числовое значение, то такое выражение называют арифметическим. Вот несколько примеров арифметических выражений, записанных по правилам языка Си:

```
a+b          12.5-z          2*(X+Y)
x++          x+++b           --n*2          n*=1
```

Три первых выражения имеют традиционную форму для языков программирования высокого уровня, поэтому их смысл очевиден. Следующие четыре выражения специфичны для языка Си.

Опишем набор операций, используемых в Си, а также правила записи и вычисления выражений. Напомним, что операция, применяемая к одному операнду, называется *унарной*, а операция с двумя операндами — *бинарной*.

Арифметические операции. К арифметическим операциям относятся:

- `-` вычитание или унарный минус;
- `+` сложение или унарный плюс;
- `*` умножение;
- `/` деление;
- `%` деление по модулю (аналог Mod в Паскале);
- `++` унарная операция увеличения на единицу (инкремент);
- `--` унарная операция уменьшения на единицу (декремент).

Все операции, кроме деления по модулю, применимы к любым числовым типам данных. Операция `%` применима только к целым числам.

Рассмотрим особенности выполнения операции деления. *Если делимое и делитель — целые числа, то и результат — целое число.* В этом случае операция / действует аналогично `Div` в Паскале. Например, значение выражения `5/3` будет равно `2`, а при вычислении `1/5` получится `0`.

Если хотя бы один из операндов имеет вещественный тип, то и результат будет вещественным. Например, операции `5./3`, `5./3.`, `5/3.` дадут вещественный результат `1.6666`.

Операции инкремента и декремента могут применяться только к переменным и не могут — к константам и выражениям. Операция ++ увеличивает значение переменной на единицу, операция -- уменьшает значение переменной на единицу. Оба знака операции могут записываться как перед операндом (префиксная форма), так и после операнда (постфиксная форма), например: `++X` или `X++`, `--a` или `a--`. Три следующих оператора дают один и тот же результат:

```
x=x+1;    ++x;    x++
```

Различие проявляется при использовании префиксной и постфиксной форм в выражениях. Проиллюстрируем это на примерах. Первый пример:

```
a=3; b=2;
c=a++*b++;
```

В результате выполнения переменные получат следующие значения: `a = 4, b = 3, c = 6`.

Второй пример:

```
a=3; b=2;
c=++a*++b;
```

Результаты будут такими: `a = 4, b = 3, c = 12`.

Объяснение следующее: при использовании постфиксной формы операции ++ и -- выполняются *после* того, как значение переменной было использовано в выражении, а префиксные операции — *до* использования. Поэтому в первом примере значение переменной *c* вычислялось как произведение 3 на 2, а во втором — как произведение 4 на 3.

По убыванию старшинства арифметические операции расположены в следующем порядке:

```
++, --
– (унарный минус)
*, /, %
+, –
```

Одинаковые по старшинству операции выполняются в порядке слева направо. Как и в Паскале, для изменения порядка выполнения операций в выражениях могут применяться круглые скобки.

Операции отношения. В Си используется тот же набор операций отношения, что и в Паскале. Следует лишь обратить внимание на различие в записи операций «равно» и «не равно».

`<`	меньше,
`<=`	меньше или равно,
`>`	больше,
`>=`	больше или равно,
`==`	равно,
`!=`	не равно.

Как уже говорилось раньше, в стандарте Си нет логического типа данных. Поэтому результатом операции отношения является целое число: если отношение истинно — то 1, если ложно — то 0.

Примеры отношений:

```
a<0,   101>=105,  'a'=='A',  'a'!='A'
```

Результатом второго и третьего отношений будет 0 — ложь; результат четвертого отношения равен 1 — истина; результат первого отношения зависит от значения переменной `a`.

Логические операции. Три основные логические операции в языке Си записываются иначе, чем в Паскале.

`!`	операция отрицания (НЕ),
`&&`	конъюнкция, логическое умножение (И),
`||`	дизъюнкция, логическое сложение (ИЛИ).

Правила их выполнения определяются таблицей истинности (см. табл. 3.5).

Например, логическое выражение, соответствующее системе неравенств $0 < x < 1$ в программе на Си запишется в виде следующего логического выражения:

```
x>0 && x<1
```

Обратите внимание на то обстоятельство, что здесь не понадобились круглые скобки для выделения операций отношения. В отличие от Паскаля, где операции отношения имеют самый низкий приоритет, в Си операции отношения старше конъюнкции и дизъюнкции. По убыванию приоритета логические операции и операции отношения расположены в следующем порядке:

```
!
>  <   >=   <=
==   !=
&&
||
```

Помимо рассмотренных в Си имеются *поразрядные* логические операции. Эти операции выполняются над каждой парой соответствующих двоичных разрядов внутреннего представления операндов. Их еще называют *битовыми* логическими операциями. Знаки битовых логических операций:

`&` поразрядная конъюнкция (И),
`|` поразрядная дизъюнкция (ИЛИ),
`^` поразрядное исключающее ИЛИ,
`~` поразрядное отрицание (НЕ).

Битовые логические операции вместе с операциями поразрядного сдвига влево (<<) и вправо (>>) позволяют добраться до каждого бита внутреннего кода. Чаще всего такие действия приходится выполнять в системных программах. В данном пособии мы их рассматривать не будем.

Операция присваивания. То, что присваивание в Си является операцией, а не оператором, оказывается, наверное, самой большой неожиданностью для знатоков Паскаля. А между тем это действительно так! Знак операции присваивания =. Следствием отмеченного факта является то, что присваивание, как любой другой знак операции, может несколько раз входить в выражение. Например:

```
a=b=c=x+y;
```

Присваивание имеет самый низкий приоритет (ниже только у операции «запятая»). Кроме того, операция присваивания — правоассоциативная. Это значит, что несколько подряд расположенных присваиваний выполняются справа налево. Поэтому в приведенном выше выражении первой выполнится операция сложения, затем переменной *c* присвоится значение суммы, затем это значение присвоится переменной *b* и в конце — переменной *a*.

В языке Си имеются дополнительные операции присваивания, совмещающие присваивание с выполнением других операций. Среди них: `+=`, `-=`, `/=`, `*=`, `%=`. Приоритет у них такой же, как и у простого присваивания. Примеры использования этих операций:

```
a+=2    эквивалентно a=a+2,
x-=a+b  эквивалентно x=x-(a+b),
p/=10   эквивалентно p=p/10,
m*=n    эквивалентно m=m*n,
r%=5    эквивалентно r=r%5.
```

Заметим, что вместо выражения `a=a+2` предпочтительнее писать в программе `a+=2`, поскольку второе выражение будет вычисляться быстрее.

Операция явного преобразования типа (операция «тип»). Применение этой операции имеет следующий формат:

```
(имя_типа) операнд
```

Операндом могут быть константа, переменная, выражение. В результате значение операнда преобразуется к указанному типу. Примеры использования преобразования типа:

```
(long)8, (float)1, (int)x%2
```

По поводу последнего выражения заметим, что приоритет операции «тип» выше деления (и других бинарных арифметических операций), поэтому сначала значение переменной *x* приведется к целому типу (отбросится дробная часть), а затем выполнится деление по модулю.

Следующий фрагмент программы иллюстрирует одну из практических ситуаций, в которой потребовалось использовать преобразование типа:

```
float c;
int a=1, b=2;
c=(float)a/b;
```

В результате переменная *c* получит значение 0,5. Без преобразования типа ее значение стало бы равно 0.

Операция `sizeof`. Эта операция имеет две формы записи:

`sizeof(тип)` и `sizeof(выражение)`

Результатом операции является целое число, равное количеству байтов, которое занимает в памяти величина явно указанного типа или величина, полученная в результате вычисления выражения. Последняя определяется также по типу результата выражения. Хотя по форме записи это похоже на функцию, однако `sizeof` является именно операцией. Ее приоритет выше, чем у бинарных арифметических операций, логических операций и отношений. Примеры использования операции:

`sizeof(int)`	результат — 2
`sizeof(1)`	результат — 2
`sizeof(0.1)`	результат — 8
`sizeof(1L)`	результат — 4
`sizeof(char)`	результат — 1
`sizeof('a')`	результат — 2

Операция «запятая». Эта необычная операция используется для связывания нескольких выражений в одно. Несколько выражений, разделенных запятыми, вычисляются последовательно слева направо. В качестве результата такого совмещенного выражения принимается значение самого правого выражения. Например, если переменная `X` имеет тип `int`, то значение выражения `(X=3, 5*X)` будет равно 15, а переменная `X` примет значение 3.

Операция «условие ?:». Это единственная операция, которая имеет три операнда. Формат операции:

`выражение1 ? выражение2 : выражение3`

Данная операция реализует алгоритмическую структуру ветвления. Алгоритм ее выполнения следующий: первым вычисляется значение выражения 1, которое обычно представляет собой неко-

торое условие. Если оно истинно, т.е. не равно 0, то вычисляется выражение 2 и полученный результат становится результатом операции. В противном случае в качестве результата берется значение выражения 3.

Пример 1. Вычисление абсолютной величины переменной *X* можно организовать с помощью одной операции:

```
X<0 ? -X : X;
```

Пример 2. Выбор большего значения из двух переменных *a* и *b*:

```
max=(a<=b)?b:a;
```

Пример 3. Заменить большее значение из двух переменных *a* и *b* на единицу:

```
(a>b)?a:b=1;
```

Правила языка в данном случае позволяют ставить условную операцию слева от знака присваивания.

Операции ( ) и [ ]. В языке Си круглые и квадратные скобки рассматриваются как операции, причем эти операции имеют наивысший приоритет. Их смысл будет раскрыт позже.

Подведем итог всему разговору об операциях Си/Си++, сведя их в общую табл. 4.2 и расположив по рангам. Ранг операции — это порядковый номер в ряду приоритетов. Чем больше ранг, тем ниже приоритет. В таблице отражено еще одно свойство операций — ассоциативность. Если одна и та же операция, повторяющаяся в выражении несколько раз, выполняется в порядке расположения слева направо, то она называется левоассоциативной; если выполняется справа налево, то операция правоассоциативная. В таблице эти свойства отображены стрелками влево и вправо. Некоторые операции, присутствующие в таблице, пока не обсуждались.

Таблица 4.2

Приоритеты (ранги) операций

Ранг	Операции	Ассоциативность
1	`( ) [ ] -> .`	→
2	`! ~ + - ++ -- & * (тип) sizeof` (унарные)	←
3	`* / %` (мультипликативные бинарные)	→
4	`+ -` (аддитивные бинарные)	→
5	`<< >>>` (поразрядного сдвига)	→
6	`< <= >= >` (отношения)	→

Ранг	Операции	Ассоциативность
7	`==` `!=` (отношения)	→
8	`&` (поразрядная конъюнкция «И»)	→
9	`^` (поразрядное исключающее «ИЛИ»)	→
10	`|` (поразрядная дизъюнкция «ИЛИ»)	→
11	`&&` (конъюнкция «И»)	→
12	`||` (дизъюнкция «ИЛИ»)	→
13	`?:` (условная)	←
14	`=` `*=` `/=` `%=` `+=` `-=` `&=` `^=` `|=` `<<=` `>>=`	←
15	`,` («запятая»)	→

Приведение типов при вычислении выражений. Практически во всех языках программирования высокого уровня работает ряд общих правил записи выражений:

- все символы, составляющие выражение, записываются в строку (нет надстрочных и подстрочных символов);
- в выражении проставляются все знаки операций;
- при записи выражения учитываются приоритеты операций;
- для влияния на последовательность операций используются круглые скобки.

Некоторые специфические особенности записи выражений на Си были описаны выше при рассмотрении операций языка.

В процессе вычисления выражений с разнотипными операндами производится автоматическое преобразование типов величин. Знание программистом правил, по которым происходят эти преобразования, предупреждает некоторые ошибки в записи выражений. Суть правил преобразования при выполнении бинарных операций сводится к следующему:

- преобразование не выполняется, если оба операнда имеют одинаковый тип;
- при разных типах операндов происходит приведение величины с младшим типом к старшему типу (кроме операции присваивания);
- при выполнении операции присваивания величина, полученная в правой части, преобразуется к типу переменной, стоящей слева от знака =.

Старшинство типов друг по отношению к другу определяется по следующему принципу: *старший тип включает в себя все значения младшего типа как подмножество.* Вещественные (плавающие) типы являются старшими по отношению к целым. В свою

очередь, внутри каждой из этих групп иерархия типов устанавливается в соответствии с указанным принципом. Целые типы по возрастанию старшинства расположены в таком порядке:

```
char→shot→int→long
```

Порядок старшинства вещественных типов следующий:

```
float→double→long double
```

Следует иметь в виду, что при преобразовании целой величины к плавающему типу может произойти потеря точности (вместо 1 получится 0,999).

Следующий пример иллюстрирует порядок выполнения операций и происходящие преобразования типов при вычислении выражения (рис. 43, цифры вверху — порядок операций).

```
char ch; int i; float f;
double d; long double r;
```

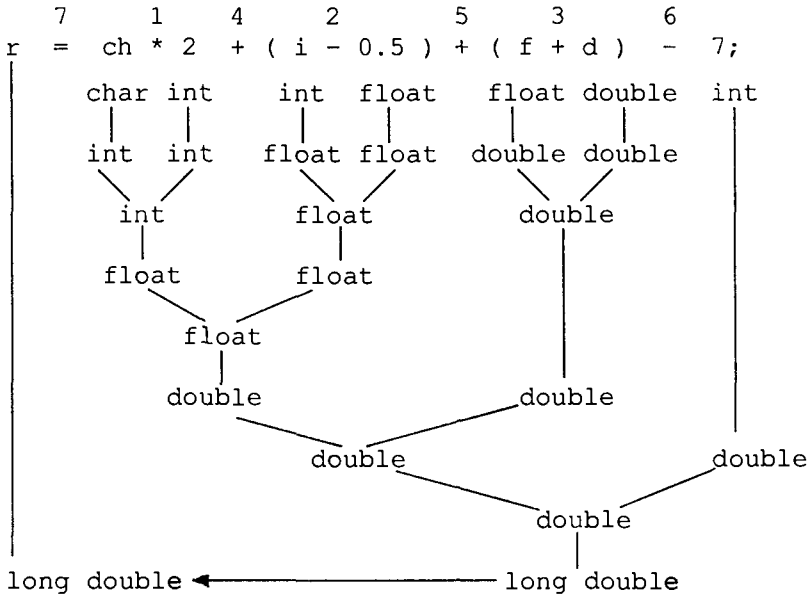


Рис. 43

Упражнения

1. Определить тип константы:

а) `315`; б) `-32.4`; в) `102408`; г) `3.7E57`; д) `0315`;
е) `0x24`; ж) `2.6L`; з) `70700U`; и) `'5'`; к) `'\121'`.

2. В программе объявлена переменная: `int n=10`. Определить результаты вычислений следующих выражений:

а) `n++;` б) `++n;` в) `n%2;` г) `n/3;` д) `n/3.;`
е) `++n+5;` ж) `5+n++;` з) `(float)n/4;`
и) `sizeof(n);` к) `sizeof(1.*n).`

3. Координаты точки на плоскости заданы переменными *X* и *Y*. Записать следующие условия в форме логических выражений:

а) точка лежит в первой четверти координатной плоскости;

б) точка лежит на оси *X*;

в) точка лежит на одной из осей;

г) точка лежит в первой или второй четверти внутри единичной окружности;

д) точка лежит на единичной окружности в третьей или четвертой четверти;

е) точка лежит внутри кольца с внутренним радиусом 1 и внешним радиусом 2 во второй или четвертой четверти.

4. В программе объявлена переменная: `float x=2`. Какое значение получит переменная *x* в результате вычисления следующих выражений?

а) `x+=2;` б) `x/=10;` в) `x*=(x+1);` г) `x+=x+=x+=1.`

5. Определить значения выражений для трех вариантов объявления переменной x: 1) `float x=1.`; 2) `float x=10.`; 3) `int x=1.`

а) `x>1?2*x:x;` б) `x/5==2?5:x/10;` в) `x>0&&x<=1?1:0.`

4.5. Линейные программы на Си/Си++

Структура программы. Общая структура программы на Си/Си++ следующая:

```
директивы_препроцессора
определение_функции_1
определение_функции_2
.....
определение_функции_N
```

Среди функций обязательно присутствует главная функция с именем **main**. Простейшая программа содержит только главную функцию и имеет следующую структуру:

```
директивы_препроцессора
void main()
{   определения_объектов;
    исполняемые_операторы;
}
```

Пока мы будем составлять только простейшие программы такого вида. Рассмотрим все необходимые средства языка для составления линейных вычислительных программ. В качестве опор-

ного примера рассмотрим программу для вычисления площади треугольника по формуле Герона.

Пример 1. Дано: a, b, c — стороны треугольника. Вычислить S — площадь треугольника. По формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p — полупериметр треугольника.

```
#include <stdio.h>
#include <math.h>
void main()
{float a,b,c,p,s;
printf("\na="); scanf("%f",&a);
printf("\nb="); scanf("%f",&b);
printf("\nc="); scanf("%f",&c);
p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
printf("\nПлощадь треугольника=%f",s);
    }
```

Разберемся более подробно, чем это делалось раньше, во всех элементах данной программы. Как уже было сказано, программа состоит из одной главной функции со стандартным именем `main`. Слово `void` обозначает отсутствие какого-либо возвращаемого этой функцией результата, а пустые скобки — отсутствие у нее аргументов. Все, что следует после заголовка функции и заключено в фигурные скобки, можно назвать телом функции. Первая строка — объявление используемых переменных. Все они плавающего типа `double`. Обратите внимание на то, что объявление переменных заканчивается точкой с запятой.

Дальнейшая часть программы — исполняемые операторы. Среди них операторы вывода на экран, ввода данных с клавиатуры, операторы присваивания.

Понятие «оператор» в Си трактуется следующим образом: *любое выражение, после которого стоит точка с запятой, воспринимается компилятором как отдельный оператор.* Оператор определяет законченное действие на очередном шаге выполнения программы.

С точки зрения данного выше определения следующая конструкция в программе является оператором:

```
i++;
```

Его называют *оператором-выражением.* Если вычисление выражения заканчивается присваиванием, то его можно назвать *оператором присваивания.* В рассматриваемой программе присутствуют два оператора присваивания: вычисления полупериметра (p) и вычисления площади треугольника (S).

В выражении для вычисления площади используется библиотечная функция `sqrt()` — квадратный корень (как и в Паскале). Данная функция относится к *библиотеке математических функций*. Для подключения этой библиотеки к нашей программе используется директива препроцессора `#include <math.h>`. Здесь `math.h` — имя заголовочного файла этой библиотеки. В табл. 4.3 даны описания некоторых наиболее часто используемых функций математической библиотеки Си.

Таблица 4.3

Математические функции (заголовочный файл `math.h`)

Обращение	Тип аргумента	Тип результата	Функция
`abs(x)`	`int`	`int`	абсолютное значение целого числа
`acos(x)`	`double`	`double`	арккосинус (радианы)
`asin(x)`	`double`	`double`	арксинус (радианы)
`atan(x)`	`double`	`double`	арктангенс (радианы)
`ceil(x)`	`double`	`double`	ближайшее целое, не меньшее x
`cos(x)`	`double`	`double`	косинус (x в радианах)
`exp(x)`	`double`	`double`	e^x — экспонента от x
`fabs(x)`	`double`	`double`	абсолютное значение вещественного x
`floor(x)`	`double`	`double`	наибольшее целое, не превышающее x
`fmod(x,y)`	`double` `double`	`double`	остаток от деления нацело x на y
`log(x)`	`double`	`double`	логарифм натуральный — $\ln x$
`log10(x)`	`double`	`double`	логарифм десятичный — $\lg x$
`pow(x,y)`	`double` `double`	`double`	x в степени y — x^y
`sin(x)`	`double`	`double`	синус (x в радианах)
`sinh(x)`	`double`	`double`	гиперболический синус
`sqrt(x)`	`double`	`double`	корень квадратный (положительное значение)
`tan(x)`	`double`	`double`	тангенс (x в радианах)
`tanh(x)`	`double`	`double`	гиперболический тангенс

В рассматриваемой программе операторы `printf(...);` и `scanf(...);` реализуют соответственно вывод на экран и ввод исходных данных с клавиатуры. Они осуществляют обращение к соответствующим функциям стандартной библиотеки ввода-вывода Си, заголовочный файл которой имеет имя `stdio.h`.

Форматированный вывод на экран. Оператор вызова функции `printf()` имеет следующую структуру:

```
printf(форматная_строка, список_аргументов);
```

Форматная строка ограничена двойными кавычками (т.е. является текстовой константой) и может включать в себя *произвольный текст, управляющие символы* и *спецификаторы формата.* Список аргументов может отсутствовать или же состоять из выражений, значения которых выводятся на экран (в частном случае из констант и переменных).

В примере 1 оператор `printf("\na=");` содержит текст (`"a="`) и управляющие символы (`"\n"`). Текст выводится на экран в том виде, в котором он записан. Управляющие символы влияют на расположение на экране выводимых знаков. В результате выполнения этого оператора на экран с новой строки выведутся символы `a=`.

Признаком управляющего символа является значок `\`. Ниже приводится их список:

`\n` — перевод строки;
`\t` — горизонтальная табуляция;
`\r` — возврат курсора к началу новой строки;
`\a` — сигнал-звонок;
`\b` — возврат на один символ (одну позицию);
`\f` — перевод (прогон) страницы;
`\v` — вертикальная табуляция.

Оператор `printf("\nПлощадь треугольника=%f",s);` содержит все виды параметров функции `printf`. Список аргументов состоит из одной переменной `s`. Ее значение выводится на экран. Пара символов `%f` является спецификацией формата выводимого значения переменной `s`. Значок `%` — признак формата, а буква `f` указывает на то, что выводимое число имеет вещественный (плавающий) тип и выводится на экран в форме с фиксированной точкой. Например, если в результате вычислений переменная `s` получит значение 32,435621, то на экран выведется:

```
Площадь треугольника=32.435621
```

Спецификатор формата определяет форму внешнего представления выводимой величины. Вот некоторые спецификаторы формата:

`%c` — символ;
`%s` — строка;
`%d` — целое десятичное число (тип `int`);

`%u` — целое десятичное число без знака (тип `unsigned`);
`%f` — вещественные числа в форме с фиксированной точкой;
`%e` — вещественные числа в форме с плавающей точкой (с мантиссой и порядком).

Например, после выполнения следующих операторов

```
float m,p;
int k;
m=84.3; k=-12; p=32.15;
printf("\nm=%f\tk=%d\tp=%e",m,k,p);
```

на экран выведется строка:

```
m=84.299999      k=-12     p=3.21500e+01
```

Здесь дважды используемый управляющий символ табуляции `\t` отделил друг от друга выводимые значения. Из этого примера видно, что соответствие между спецификаторами формата и элементами списка аргументов устанавливается в порядке их записи слева направо.

К спецификатору формата могут быть добавлены числовые параметры: *ширина поля* и *точность*. Ширина — это число позиций, отводимых на экране под величину, а точность — число позиций под дробную часть (после точки). Параметры записываются между значком `%` и символом формата и отделяются друг от друга точкой. Внесем изменения в оператор вывода для рассмотренного выше примера.

```
printf("\nm=%5.2f\tk=%5d\tp=%8.2e\tp=%11.4e",m,k,p,p);
```

В результате на экране получим:

```
m=84.30   k= -12    p=    32.15          p= 3.2150e+01
```

Если в пределы указанной ширины поля выводимое значение не помещается, то этот параметр игнорируется и величина будет выводиться полностью.

К спецификаторам формата могут быть добавлены *модификаторы* в следующих вариантах:
`%ld` — вывод `long int`;
`%hu` — вывод `short unsigned`;
`%Lf` — вывод `long double`.

Форматированный ввод с клавиатуры. Оператор вызова функции `scanf()` имеет следующую структуру:

```
scanf(форматная_строка, список_аргументов);
```

Данная функция осуществляет чтение символов, вводимых с клавиатуры, и преобразование их во внутреннее представление в соответствии с типом величин. В функции `scanf()` форматная строка и список аргументов присутствуют обязательно. В программе из примера 1 имеется оператор:

```
scanf("%f",&a);
```

Здесь "%f" - форматная строка; &a — список аргументов, состоящий из одного элемента. Этот оператор производит ввод числового значения в переменную a.

Символьную последовательность, вводимую с клавиатуры и воспринимаемую функцией scanf(), принято называть *входным потоком*. Функция scanf() разделяет этот поток на отдельные вводимые величины, интерпретирует их в соответствии с указанным типом и форматом и присваивает переменным, содержащимся в списке аргументов.

Список аргументов — это перечень вводимых переменных, причем перед именем каждой переменной ставится значок &. Это знак операции «взятие адреса переменной». Подробнее смысл этого действия будет объяснен позже, а пока примем это правило формально.

Форматная строка заключается в кавычки (как и для printf) и состоит из списка спецификаций. Каждая спецификация начинается со знака %, после которого могут следовать

```
*ширина_поля модификатор спецификатор
```

Из них обязательным элементом является лишь спецификатор. Для ввода числовых данных используются следующие спецификаторы:

d — для целых десятичных чисел (тип int);

u — для целых десятичных чисел без знака (тип unsigned int);

f — для вещественных чисел (тип float) в форме с фиксированной точкой;

e — для вещественных чисел (тип float) в форме с плавающей точкой.

Звездочка в спецификации позволяет пропустить во входном потоке определенное количество символов. *Ширина поля* — целое положительное число, позволяющее определить число символов из входного потока, принадлежащих значению соответствующей вводимой переменной. Как и в спецификациях вывода для функции printf(), в спецификациях ввода функции scanf() допустимо использование *модификаторов* h, l, L. Они применяются при вводе значений модифицированных типов:

hd — для ввода значений типа short int;

ld — для ввода значений типа long int;

lf, le — для ввода значений типа double в форме с фиксированной и плавающей точкой;

Lf, Le — для ввода значений типа long double в форме с фиксированной и плавающей точкой.

В программе из примера 1 все три величины *a*, *b*, *c* можно ввести одним оператором:

```
scanf("%f%f%f",&a,&b,&c);
```

Если последовательность ввода будет такой:

```
5  3.2  2.4 <Enter>
```

то переменные получат следующие значения: $a = 5{,}0$, $b = 3{,}2$, $c = 2{,}4$. Разделитслем в потоке ввода между различными значениями может быть любое количество пробелов, а также другие пробельные символы: знак табуляции, конец строки. Только после нажатия на клавишу **Enter** вводимые значения присвоятся соответствующим переменным. До этого входной поток помещается в буфер клавиатуры и может редактироваться.

Потоковый ввод-вывод в Си++. Программируя на языке Си++, можно пользоваться средствами ввода-вывода стандартной библиотеки Си, подключаемой с помощью заголовочного файла `stdio.h`, как это делалось выше. Однако в Си++ имеются свои специфические средства ввода-вывода. Это *библиотека классов*, подключаемая к программе с помощью файла `iostream.h`. В этой библиотеке определены в качестве *объектов* стандартные символьные потоки со следующими именами:

`cin` — стандартный поток ввода с клавиатуры;

`cout` — стандартный поток вывода на экран.

Ввод данных интерпретируется как *извлечение из потока* `cin` и присваивание значений соответствующим переменным. В Си++ определена операция извлечения из стандартного потока, знак которой >>. Например, ввод значений в переменную `x` реализуется оператором

```
cin>>x;
```

Вывод данных интерпретируется как *помещение в стандартный поток cout* выводимых значений. Выводиться могут тексты, заключенные в двойные кавычки, и значения выражений. Знак операции помещения в поток <<. Примеры использования потокового вывода:

```
cout<<a+b;
cout<<"\nРезультат="<<Y;
cout<<"x="<<x<<" y="<<y<<" z="<<z<<endl;
```

Из приведенных примеров видно, что в выходном потоке можно использовать управляющие символы, как и при использовании функции `printf()`; перед каждым элементом вывода нужно ставить знак операции <<. Элемент вывода `endl` является так называемым манипулятором, определяющим перевод курсора на новую строку (действует аналогично управляющему символу `\n`).

В процессе потокового ввода-вывода происходит преобразование из формы внешнего символьного представления во внутренний формат и обратно. Тип данных и необходимый формат

определяются автоматически. Стандартные форматы задаются специальными *флагами форматирования*, которые устанавливаются с помощью функции `setf()`. Кроме того, на формат отдельных выводимых данных можно влиять путем применения специальных манипуляторов. Здесь мы не будем подробно разбирать эти вопросы.

Перепишем программу из примера 1 в варианте с использованием потокового ввода-вывода Си++.

```
#include <iostream.h>
#include <math.h>
void main()
{float a,b,c,p,s;
cout<<"\na="; cin>>a;
cout<<"\nb="; cin>>b;
cout<<"\nc="; cin>>c;
p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
cout<<"\nПлощадь треугольника="<<s;
    }
```

Упражнения

1. При выполнении программы

```
#include <stdio.h>
void main()
{ float a,b,c;
  int m,n;
  skanf("%f%d%f%d%f",&a,&m,&b,&n,&c);
  printf("\na=%8.3fd=%7.2fc=%12.3e",a,b,c);
  printf("\nm=%10dn=%5d",m,n);
}
```

с клавиатуры была введена следующая символьная последовательность:

```
32.4 87 0.05 4567 2314.45 <Enter>
```

Как будет выглядеть на экране результат работы программы?

2. Приведенная ниже программа решает следующую задачу: идет k-я секунда суток. Определить, сколько целых часов (H) и целых минут (M) прошло с начала суток. Например, если $k = 13257 = 3 \times 36000 + 40 \times 60 + 57$, то $H = 3$, $M = 40$. Вывести на экран фразу: «Это...часов...минут». Вместо многоточий поставить вычисленные значения H и M.

```
#include <stdio.h>
void main()
{long k;
 int h,m;
```

```
    printf("Введите текущее время в секундах:");
    scanf("%ld",&k);
    h=k/3600;
    m=(k%3600)/60;
    printf("Это %d часов %d минут.\n",h,m);
}
```

Разобрать, как работает данная программа. Переписать ее с использованием потокового ввода-вывода Си++.

3. Составить программу решения обратной задачи по отношению к предыдущей: дано количество часов и минут, прошедших от начала суток. Определить количество секунд.

4. Составить программу вычисления объема и площади поверхности куба по данной длине ребра.

5. Составить программу для вычисления корней квадратного уравнения.

4.6. Программирование ветвлений

Для программирования ветвящихся алгоритмов в языке Си имеется несколько различных средств. К ним относятся рассмотренная выше операция условия `?:`, условный оператор **`if`** и оператор выбора **`switch`**.

Условный оператор. Формат условного оператора следующий:

```
if (выражение) оператор1; else оператор2;
```

Это полная форма оператора, программирующая структуру полного ветвления. Обычно *выражение* — это некоторое условие, содержащее операции отношения и логические операции. Значение выражения приводится к целому и интерпретируется в соответствии с правилом: равно нулю — ложь, не равно нулю — истина. Если выражение истинно, выполняется `оператор1`, если ложно — `оператор2`.

Необходимо обратить внимание на следующие особенности синтаксиса условного оператора:

- выражение записывается в круглых скобках;
- точка с запятой после оператора 1 ставится обязательно.

Последнее обозначает, что правило Паскаля — не ставить точку с запятой перед `else` — здесь не работает.

Возможно использование неполной формы условного оператора

```
if (выражение) оператор;
```

Вот пример использования полной формы условного оператора для нахождения большего значения из двух переменных *a* и *b*:

```
if(a>b) max=a; else max=b;
```

Та же самая задача может быть решена с использованием неполного ветвления следующим образом:

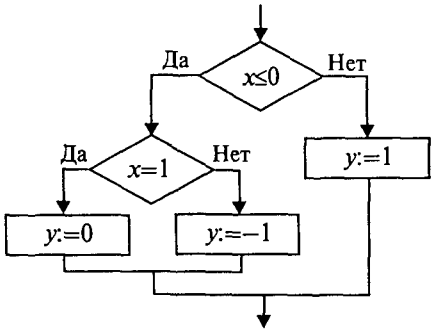
```
max=a; if(b>a) max=b;
```

Напомним, что эту же задачу можно решить с помощью операции «условие» (см. разд. 4.4).

Теперь рассмотрим примеры программирования вложенных ветвящихся структур. Требуется вычислить функцию sign(x) — знак x, которая определена следующим образом:

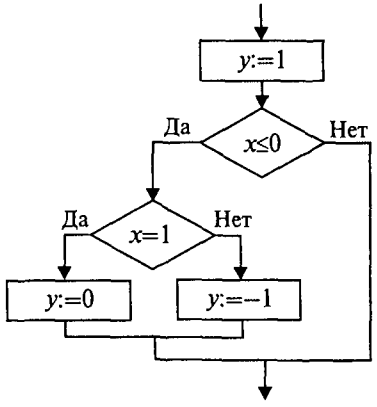
$$\text{sign}(x) = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

Пример 1. Алгоритм с полными вложенными ветвлениями:



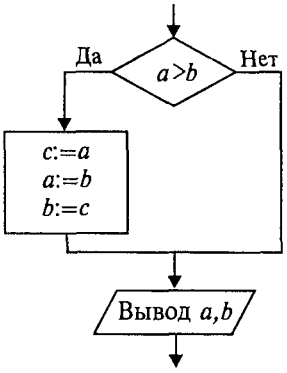

```
if(x<=0)
  if(x==0) y=0;
  else y=-1;
else y=1;
```

Пример 2. Алгоритм с неполным ветвлением:




```
y=1;
if(x<=0)
  if(x==0) y=0;
  else y=-1;
```

Пример 3. Упорядочить по возрастанию значения в двух переменных *a*, *b*:




```
if(a>b)
{c=a; a=b; b=c;}
cout<<"a="<<a<<"b="<<b;
```

В данном примере использован *составной оператор* — последовательность операторов, заключенная в фигурные скобки. В Си фигурные скобки выполняют роль операторных скобок по аналогии с **Begin**, **End** в Паскале.

Обратите внимание на то, что перед закрывающей фигурной скобкой точку с запятой надо ставить обязательно, а после скобки точка с запятой не ставится.

В следующем примере вернемся к задаче вычисления площади треугольника по длинам трех сторон. Добавим в программу проверку условия правильности исходных данных: *a*, *b*, *c* должны быть положительными, а сумма длин каждой пары сторон треугольника должна быть больше длины третьей стороны (аналогичную программу на Паскале см. в разд. 3.10, пример 1).

Пример 4.

```
// Площадь треугольника
#include <iostream.h>
#include <math.h>
void main()
{float a,b,c,p,s;
cout<<"\na="; cin>>a;
cout<<"\nb="; cin>>b;
cout<<"\nc="; cin>>c;
if(a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a)
{ p=(a+b+c)/2;
  s=sqrt(p*(p-a)*(p-b)*(p-c));
  cout<<"\nПлощадь треугольника="<<s;
}
    else cout("\n Неверные исходные данные.");
}
```

Оператор выбора (переключатель). Формат оператора выбора:

```
switch (целочисленное_выражение)
{ case константа1: список_операторов;
  case константа2: список_операторов;
  . . . . . .
  default: список_операторов;
}
```

Последняя строка (`default`) может отсутствовать.

Выполнение оператора происходит в следующем порядке:

1. Вычисляется выражение.

2. Полученное значение последовательно сравнивается с константами, помещенными после служебного слова `case`; при первом совпадении значений выполняются операторы, стоящие после двоеточия.

3. Если ни с одной из констант совпадения не произошло, то выполнятся операторы после слова `default`.

Для того чтобы «обойти» выполнение операторов на последующих ветвях, нужно принять специальные меры, используя операторы выхода или перехода.

Рассмотрим фрагмент программы, который переводит числовую оценку знаний ученика в ее словесный эквивалент. Согласно вузовской системе: 5 — «отлично», 4 — «хорошо», 3 — «удовлетворительно», 2 — «неудовлетворительно».

Пример 5.

```
#include <iostream.h>
void main()
{int ball;
cout<<"\nВведите оценку: "; cin>>ball;
switch (ball)
{ case 2: cout<<"\tЭто неудовлетворительно!\n";
          break;
   case 3: cout<<"\tЭто удовлетворительно!\n";
          break;
   case 4: cout<<"\t Это хорошо!\n"; break;
   case 5: cout<<"\tЭто отлично!\n"; break;
   default: cout<<"\tНет такой оценки!\n";
   }
}
```

Здесь используется еще один новый для нас оператор **`break`** — оператор выхода. Его исполнение завершает работу оператора выбора, т.е. происходит «обход» других ветвей. Вот два варианта результатов выполнения этой программы:

```
Введите оценку: 3     Это удовлетворительно!
Введите оценку: 7     Нет такой оценки!
```

Если на всех ветвях убрать оператор `break`, то результат может выглядеть следующим образом:

```
Введите оценку: 3     Это удовлетворительно!
        Это хорошо!
        Это отлично!
        Нет такой оценки!
```

В этом случае выполнились операторы на всех ветвях, начиная с той, которая помечена константой 3.

Возможны задачи, в которых такой порядок выполнения ветвей оператора выбора может оказаться полезным. В следующем фрагменте программы происходит возведение вещественного числа x в целую степень n, где n изменяется в диапазоне от 1 до 5.

```
y=1.0;
switch(n)
{ case 5: y=y*x;
  case 4: y=y*x;
  case 3: y=y*x;
  case 2: y=y*x;
  case 1: y=y*x; cout<<"y="<<y; break;
  default: cout<<"Степень больше 5";
}
```

Упражнения

1. Составить программу упорядочения по возрастанию значений в трех переменных.
2. Составить программу, которая выводит на экран меню:
   1. Первое
   2. Второе
   3. Третье

и в зависимости от выбранного пункта выдает одну из надписей: «Получите суп», «Получите картошку», «Получите компот», «Оставайтесь голодным». Написать два варианта программы: с использованием условного оператора `if` и с использованием переключателя.

3. Составить программу решения квадратного уравнения $ax^2 + bx + c = 0$, учитывающую все возможные варианты исходных данных:
   1) $a = 0,\ b = 0,\ c = 0$;
   2) $a = 0,\ b = 0,\ c \neq 0$;
   3) $a = 0,\ b \neq 0$;
   4) $a \neq 0,\ D \geq 0$ (D — дискриминант);
   5) $a \neq 0,\ D < 0$.

В каждом случае должно выводиться соответствующее решение или сообщение.

4.7. Программирование циклов

В Си, как и в Паскале, существуют все три типа операторов цикла: цикл с предусловием, цикл с постусловием и цикл с параметром.

Цикл с предусловием. Формат оператора цикла с предусловием:

```
while (выражение) оператор;
```

Цикл повторяет свое выполнение, пока значение выражения отлично от нуля, т.е. заключенное в нем условие цикла истинно.

В качестве примера использования оператора цикла рассмотрим программу вычисления факториала целого положительного числа $N!$. Сопоставим программу решения этой задачи, написанную на Паскале, с программой на Си.

Пример 1.

Программа на Паскале

```
Program Faktorial
Var F:Longint; i,N:
Integer;
Begin write('N=');
ReadLn(N);
      F:=1; i:=1;
      While i≤N Do
      Begin
          F:=F*i;
          i:=i+1
      End;
      WriteLn(N,'!=',F)
End.
```

Программа на Си++

```
  // Программа вычисле-
ния факториала
  #include <iostream.h>
  void main()
  { long int F;
    int i,N;
    cout<<"N="; cin>>N;
    F=i=1;
    while(i<=N) F=F*i++;
   cout<<"\n"<<N<<"!="<<F;
  }
```

Обратите внимание на операторы в теле цикла. Конечно, и в Си-программе можно было написать два оператора присваивания, объединив их фигурными скобками. Однако использованный способ записи более лаконичен и более характерен для Си. Этот же самый оператор можно было записать еще короче: `F*=i++`

При практическом использовании этой программы не следует забывать, что факториал — очень быстро растущая функция, и поэтому при определенных значениях N выйдет из диапазона, соответствующего типу `long int`. Задав для переменной F тип `unsigned long`, можно сдвинуть эту границу, но этого может оказаться недостаточно. Предлагаем в качестве самостоятельного задания исследовать предельные значения N для двух указанных типов переменной F.

Интересно свойство следующего оператора:

```
while(1);
```

Это бесконечный пустой цикл. Использование в качестве выражения константы 1 приводит к тому, что условие повторения цикла все время остается истинным и работа цикла никогда не заканчивается. Тело в этом цикле представляет собой *пустой оператор*. При исполнении такого оператора программа будет «топтаться на месте».

Рассмотрим еще один пример использования оператора цикла **while**. Вернемся к задаче итерационного вычисления суммы гармонического ряда: $1+1/2+1/3+\ldots$ с заданной точностью ε (см. разд. 3.10, пример 2). Снова сопоставим программу на Си с программой на Паскале.

Пример 2.

Программа на Паскале

```
Var n: Integer; S,eps:
Real;
Begin
 Write('Точность:');
 ReadLn(eps);
 n:=1; S:=0;
 While(1/n>eps)
    And(n<MAXINT) Do
    Begin S:=S+1/n;
       n:=n+1
    End;
    WriteLn('Сумма=',S)
End.
```

Программа на Си++

```
// Сумма
//гармонического ряда
#include <iostream.h>
#include <limits.h>
void main()
{int n=1;
 double S=0, eps;
 cout<<"Точность:";
 cin>>eps;
 while(1.0/n>eps &&
 n<INT_MAX)
   S+=1./n++;
 cout<<"\nСумма="<<S;
}
```

Файл `limits.h`, подключаемый препроцессором, содержит определения предельных констант для целых типов данных. В частности, константа с именем `INT_MAX` равна максимальному значению типа `int` в данной реализации компилятора. Если для типа `int` используется двухбайтовое представление, то `INT_MAX=32767`. В этом же заголовочном файле определены и другие константы: `INT_MIN=-32768; LONG_MAX=2147483647` и т.д.

Цикл с постусловием. Формат оператора цикла с постусловием:

do оператор **while** (выражение);

Цикл выполняется до тех пор, пока выражение отлично от нуля, т.е. заключенное в нем условие цикла истинно. Выход из цикла происходит после того, как значение выражения станет ложным, иными словами равным нулю. Таким образом, в отличие от оператора **repeat** ... **until**, используемого в Паскале, где в конце пишется условие выхода из цикла, в операторе **do** ... **while** в Си в конце пишется условие повторения цикла. В качестве примера рассмотрим программу вычисления $N!$, в которой использу-

ется цикл с постусловием, и сопоставим ее с аналогичной программой на Паскале.

Пример 3.

Программа на Паскале

```
Program Faktorial
Var F:Longint; i,N:
Integer;
Begin Write('N=');
   ReadLn(N);
   F:=1; i:=1;
   Repeat
     F:=F*i;
     i:=i+1
   Until i>N;
   WriteLn(N,'!=',F)
End.
```

Программа на Си++

```
// Программа
вычисления факториала
#include <iostream.h>
void main()
{ long int F;
  int i,N;
  cout<<"N="; cin>>N;
  F=i=1;
  do F*=i++;
while(i<=N);
 cout<<"\n"<<N<<"!="<<F;
}
```

Цикл с параметром. Формат оператора цикла с параметром:

```
for (выражение_1; выражение_2; выражение_3)
      оператор;
```

Выражение 1 выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла). Выражение 2 — это условие выполнения цикла. Выражение 3 обычно определяет изменение параметра цикла, оператор — тело цикла, которое может быть простым или составным. В последнем случае используются фигурные скобки.

Алгоритм выполнения цикла **for** представлен на блок-схеме на рис. 44.

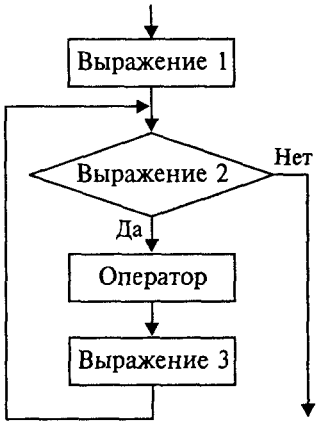


Рис. 44

Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

С помощью цикла **for** нахождение $N!$ можно организовать следующим образом:

```
F=1;
for(i=1;i<=N;i++) F=F*i;
```

Используя операцию «запятая», можно в выражение 1 внести инициализацию значений сразу нескольких переменных:

```
for(F=1,i=1;i<=N;i++) F=F*i;
```

Некоторых элементов в операторе **for** может не быть, однако разделяю-

щие их точки с запятой обязательно должны присутствовать. В следующем примере инициализирующая часть вынесена из оператора **for**:

```
F=1;
i=1;
for(;i<=N;i++) F=F*i;
```

Ниже показан еще один вариант вычисления *N*!. В нем на месте тела цикла находится пустой оператор, а вычислительная часть внесена в выражение 3.

```
for(F=1,i=1;i<=N;F=F*i,i++);
```

Этот же оператор можно записать в следующей форме:

```
for(F=1,i=1;i<=N;F*=i++);
```

В языке Си оператор **for** является достаточно универсальным средством для организации циклов. С его помощью можно программировать даже итерационные циклы, что невозможно в Паскале. Вот пример вычисления суммы элементов гармонического ряда, превышающих заданную величину ε:

```
for(n=1,S=0;1.0/n>eps && n<INT_MAX;n++) S+=1.0/n;
```

И наконец, эта же самая задача с пустым телом цикла:

```
for(n=1,S=0;1.0/n>eps && n<INT_MAX;S+=1.0/n++);
```

Следующий фрагмент программы на Си++ содержит два вложенных цикла **for**. В нем запрограммировано получение на экране таблицы умножения.

```
for(x=2;x<=9;x++)
  for(y=2;y<=9;y++)
    cout<<"\n"<<x<<"*"<<y<<"="<<x*y;
```

На экране будет получен следующий результат:

```
2*2=4
2*3=6
. . .
9*8=72
9*9=81
```

Оператор continue. Если выполнение очередного шага цикла требуется завершить до того, как будет достигнут конец тела цикла, используется оператор **continue**. Следующий фрагмент программы обеспечивает вывод на экран всех четных чисел в диапазоне от 1 до 100.

```
for(i=1;i<=100;i++)
{if(i%2) continue; cout<<"\t"<<i;}
```

Для нечетных значений переменной *i* остаток от деления на 2 будет равен единице, этот результат воспринимается как значение «истина» в условии ветвления, и выполняется оператор **continue**. Он завершит очередной шаг цикла, выполнение цикла перейдет к следующему шагу.

Оператор goto. Оператор безусловного перехода **goto** существует в языке Си, как и во всех других языках программирования высокого уровня. Однако с точки зрения структурного подхода к программированию его использование рекомендуется ограничить. Формат оператора:

```
goto метка;
```

Метка представляет собой идентификатор с последующим двоеточием, ставится перед помечаемым оператором.

Одна из ситуаций, в которых использование **goto** является оправданным — это необходимость «досрочного» выхода из вложенного цикла. Вот пример такой ситуации:

```
for(...)
{ while(...)
  { for(...)
    {... goto exit ...}
  }
}
exit: cout<<"выход из цикла";
```

При использовании оператора безусловного перехода необходимо учитывать следующие ограничения:

- нельзя входить внутрь блока извне;
- нельзя входить внутрь условного оператора (`if...else...`);
- нельзя входить внутрь переключателя;
- нельзя входить внутрь цикла.

Упражнения

1. Используя циклы `while`, `do – while` и `for`, написать три варианта программы получения на экране таблицы синусов для значений аргумента в диапазоне от 0 до $\pi/2$ с заданным числом шагов.

2. Вычислить и вывести все члены числового ряда

$$1, \frac{1}{2!}, \frac{1}{3!}, \ldots, \frac{1}{N!},$$

значение которых превышает 10^{-5}.

3. Напечатать в возрастающем порядке все трехзначные числа, в десятичной записи которых нет одинаковых цифр.

4. Дано целое $n > 2$. Напечатать все простые числа из диапазона $[2, n]$.

5. Составить программу перевода целого десятичного числа в двоичную систему счисления.

6. Составить программу перевода целого десятичного числа в шестнадцатеричную систему счисления.

4.8. Функции

А теперь нам предстоит разобраться с вопросом о том, как в Си/Си++ реализуется механизм подпрограмм. Вспомним о том, что в Паскале существуют две разновидности подпрограмм: подпрограммы-процедуры и подпрограммы-функции. Тот и другой тип подпрограмм описывается внутри основной программы и компилируется вместе с ней. Реализация механизма подпрограмм в Си/Си++ существенно отличается от аналогичной реализации в Паскале.

Определение функции. Обращение к функции. В Си используется лишь один тип подпрограмм — функция. Здесь вообще не принято употреблять термин «подпрограмма», потому что *функция является основной программной единицей в Си*, минимальным исполняемым программным модулем. Всякая программа обязательно включает в себя основную функцию с именем **main**. Если в программе используются и другие функции, то они выполняют роль подпрограмм.

Рассмотрим пример. Требуется составить программу нахождения наибольшего значения из трех величин — $\max(a, b, c)$. Для ее решения можно использовать вспомогательный алгоритм нахождения максимального значения из двух, поскольку справедливо равенство: $\max(a, b, c) = \max(\max(a, b), c)$.

Вот программа решения этой задачи с использованием вспомогательной функции.

Пример 1.

```
#include <iostream.h>
//Определение вспомогательной функции
int MAX(int x, int y)
{ if (x>y) return x;
  else return y;
}
//Основная функция
void main()
{ int a,b,c,d;
  cout<<"Введите a,b,c:";
  cin>>a>>b>>c;
  d=MAX(MAX(a,b),c);
  cout<<"\nmax(a,b,c)="<<d;
}
```

Формат определения функции следующий:

```
тип имя_функции (спецификация_параметров)
{тело_ функции}
```

Тип функции — это тип возвращаемого функцией результата. Если функция не возвращает никакого результата, то для нее указывается тип **void**.

Имя функции — идентификатор, задаваемый программистом или `main` для основной функции.

Спецификации параметров — это либо «пусто», либо список имен формальных параметров функции с указанием типа для каждого из них.

Тело функции — это либо составной оператор, либо *блок*.

Здесь мы впервые встречаемся с понятием блока. Признаком блока является наличие описаний программных объектов (переменных, массивов и т.д.), которые действуют в пределах этого блока. Блок, как и составной оператор, ограничивается фигурными скобками.

В Си действует правило: *тело функции не может содержать в себе определения других функций*. Иначе говоря, недопустимы внутренние функции, как это делается в Паскале. Из всякой функции возможно обращение к другим функциям, однако они всегда являются внешними по отношению к вызывающей.

Оператором возврата из функции в точку ее вызова является оператор `return`. Он может использоваться в функциях в двух формах:

return; или **return** выражение;

В первом случае функция не возвращает никакого значения в качестве своего результата. Во втором случае результатом функции является значение указанного выражения. Тип этого выражения должен либо совпадать с типом функции, либо относиться к числу типов, допускающих автоматическое преобразование к типу функции.

Оператор return может в явном виде отсутствовать в теле функции. В таком случае его присутствие подразумевается перед закрывающей тело функции фигурной скобкой. Такая подстановка производится компилятором.

Формат обращения к функции (вызова функции) традиционный:

```
имя_функции(список_фактических_праметров)
```

Однако в Си обращение к функции имеет своеобразную трактовку: обращение к функции — это выражение. В этом выражении круглые скобки играют роль знака операции, для которой функция и фактические параметры (аргументы) являются операндами. Приоритет операции «скобки» самый высокий (см. табл. 4.2), поэтому вычисление функции в выражениях производится раньше других операций.

Между формальными и фактическими параметрами при вызове функции должны соблюдаться правила соответствия *по последовательности* и *по типам*. Фактический параметр — это выражение того же типа, что и у соответствующего ему формального параметра. Стандарт языка Си допускает автоматическое преобразование значений фактических параметров к типу формальных параметров. В Си++ такое преобразование не предусмотрено. Поэтому в дальнейшем мы будем строго следовать принципу соответствия типов.

Необходимо усвоить еще один важнейший принцип, действующий в Си/Си++: *передача параметров при вызове функции происходит только по значению*. Если снова проводить аналогию с Паскалем, то это значит, что в Си допустимы только параметры-значения (без `var`). Поэтому *выполнение функции не может изменить значения переменных, указанных в качестве фактических параметров.*

Правило соответствия по количеству, обязательное в Паскале, в Си в некоторых случаях может не соблюдаться. Более того, в Си возможны функции с переменным числом параметров. Примером таких функций являются библиотечные функции `printf()` и `scanf()`.

Прототип функции. Оказывается, совсем не обязательно было в предыдущем примере помещать полное определение функции `MAX()` перед основной частью программы. Вот другой вариант программы, решающей ту же самую задачу.

Пример 2.

```
#include <iostream.h>
//Прототип функции MAX
int MAX(int, int);
//Основная функция
void main()
{ int a,b,c,d;
  cout<<"Введите a,b,c:";
  cin>>a>>b>>c;
  d=MAX(MAX(a,b),c);
  cout<<"\nmax(a,b,c)="<<d;
}
//Определение функции MAX
int MAX(int x, int y)
{ if (x>y) return x;
  else return y;
}
```

Здесь использован *прототип* функции. Прототипом называется предварительное описание функции, в котором содержатся все необходимые сведения для правильного обращения к ней: имя и тип функции, типы формальных параметров. В прототипе имена

формальных параметров указывать необязательно (как это сделано в примере 2), хотя их указание не является ошибочным. Можно было написать и так, как в заголовке определения функции:

```
int MAX(int x, int y);
```

Точка с запятой в конце прототипа ставится обязательно!

Можно было бы записать прототип и в теле основной функции наряду с описаниями других программных объектов в ней. Вопрос о размещении описаний связан с понятием области видимости, который будет обсужден немного позже.

В следующей программе приводится пример использования функции, которая не имеет параметров и не возвращает никаких значений в точку вызова. Эта функция рисует на экране строку, состоящую из 80 звездочек.

Пример 3.

```
#include <iostream.h>
//Прототип функции line
void line(void);
//Основная функция
void main()
{  line(); //Вызов функции line
}
//Определение функции line
void line(void)
{  int i;
   for(i=0; i<80; i++) cout<<"*";
}
```

А теперь сопоставим программу на Паскале (разд. 3.13) для вычисления наибольшего общего делителя для суммы, разности и произведения двух чисел с аналогичной программой на Си++.

Пример 4.

Программа на Паскале

```
Program NOD3;
Var a,b,Rez: Integer;
Function
NOD2(M,N:Integer):
Integer;
Begin
  While M<>N Do
    If M>N
    Then M:=M-N
    Else N:=N-M;
  NOD2:=M
End;
```

Программа на Си++

```
#include <iostream.h>
#include <math.h>
int NOD2(int,int);
void main()
{ int a,b,Rez;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  Rez=NOD2(NOD2(a+b,
abs(a-b)),a*b);
  cout<<"NOD равен"<<Rez;
}
```

```
Begin
  Write("a=");
ReadLn(a);
  Write("b=");
ReadLn(b);
  Rez:=NOD2(NOD2(a+b,
       Abs(a-b)),a*b);
  WriteLn("NOD равен",
Rez)
End.
```

```
int NOD2(int M, int N)
{ while(M!=N)
  { if(M>N) M=M-N;
    else N=N-M;
  }
  return M;
}
```

У читателя может возникнуть вопрос: если основная часть программы является функцией, то кто (или что) ее вызывает? Ответ состоит в следующем: программу вызывает операционная система при запуске программы на исполнение. И в принципе `main`-функция совсем не обязательно должна иметь тип `void`. Например, она может возвращать операционной системе целое значение 1 в качестве признака благополучного завершения программы и 0 — в «аварийном» случае. Обработка этих сообщений будет осуществляться системными средствами.

Использование библиотечных функций. Библиотечными называются вспомогательные функции, хранящиеся в отдельных файлах. Стандартные библиотеки входят в стандартный комплект системы программирования на Си/Си++. Кроме того, программист может создавать собственные библиотеки функций. Ранее уже говорилось о том, что для использования стандартных функций необходимо подключать к программе заголовочные файлы соответствующих библиотек. Делается это с помощью директивы претранслятора `#include` с указанием имени заголовочного файла. Например, `#include <stdio.h>`. Все заголовочные файлы имеют расширение `h` (от английского *header*). Теперь должно быть понятно, что эти файлы содержат прототипы функций библиотеки. На стадии претрансляции происходит подстановка прототипов перед основной функцией, после чего компилятор в состоянии контролировать правильность обращения к функциям. Сами программы, реализующие функции, хранятся в форме объектного кода и подключаются к основной программе на стадии редактирования связей (при работе компоновщика).

Рассмотрим программу решения следующей задачи: зная декартовы координаты вершин выпуклого четырехугольника, вычислить его площадь (рис. 45).

Математическое решение этой задачи следующее. Обозначим координаты вершин четырехугольника так: ($x1$, $y1$), ($x2$, $y2$), ($x3$, $y3$), ($x4$, $y4$). Площадь четырехугольника можно вычислить как сумму площадей двух треугольников. В свою очередь, площадь каждого треугольника вычисляется по формуле Герона. Для примене-

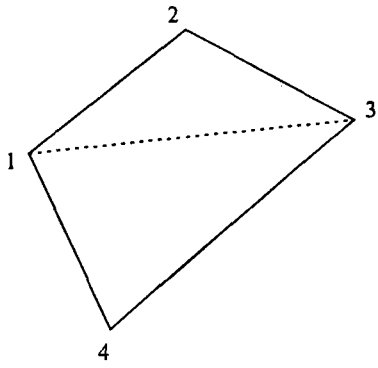


Рис. 45

ния формулы Герона нужно найти длины сторон. Длина стороны между первой и второй вершинами вычисляется по формуле:

$$L_{12} = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}.$$

Аналогично вычисляются длины других отрезков.

Таким образом, для решения основной задачи — вычисления площади четырехугольника — требуется вспомогательный алгоритм вычисления площади треугольника, для которого, в свою очередь, необходим вспомогательный алгоритм вычисления длины отрезка по координатам концов.

Ниже приведена программа решения поставленной задачи.

Пример 5.

```
//Площадь выпуклого четырехугольника
#include <iostream.h>
#include <math.h>
#include <conio.h>
typedef double D; //Переименование типа double
D Line(D,D,D,D); //Прототип функции Line
D Geron(D,D,D,D,D,D); // Прототип функции Geron
//Основная функция
void main()
{ D x1,y1,x2,y2,x3,y3,x4,y4,S1234;
  clrscr();
  cout<<"x1="; cin>>x1; cout<<"y1="; cin>>y1;
  cout<<"x2="; cin>>x2; cout<<"y2="; cin>>y2;
  cout<<"x3="; cin>>x3; cout<<"y3="; cin>>y3;
  cout<<"x4="; cin>>x4; cout<<"y4="; cin>>y4;
  S1234=Geron(x1,y1,x2,y2,x3,y3)
       +Geron(x1,y1,x3,y3,x4,y4);
  cout<<"Площадь четырехугольника="<<S1234;
}
//Определение функции Line
D Line(D a,D b,D c,D d)
{return sqrt((a-c)*(a-c)+(b-d)*(b-d));}
//Определение функции Geron
D Geron(D a1,D a2,D b1,D b2,D c1,D c2)
{D p,ab,bc,ca;
    ab=Line(a1,a2,b1,b2); bc=Line(b1,b2,c1,c2);
    ca=Line(c1,c2,a1,a2);
```

```
    p=(ab+bc+ca)/2;
    return sqrt(p*(p-ab)*(p-bc)*(p-ca));
}
```

В этой программе используются функции из трех стандартных библиотек с заголовочными файлами `iostream.h`, `math.h` и `conio.h`. С первыми двумя мы уже встречались раньше. Третья библиотека (файл `conio.h`) содержит функции, предназначенные для управления выводом на экран в символьном режиме. Она является аналогом модуля CRT в Турбо Паскале. В программе из этой библиотеки используется функция `clrscr()` — очистка экрана.

Еще одним новым элементом в приведенной программе является строка

```
typedef double D;
```

Служебное слово `typedef` представляет собой спецификатор типа, позволяющий определять синонимы для обозначения типов. В результате в рассматриваемой программе вместо длинного слова `double` для обозначения того же самого типа можно употреблять одну букву `D`. Данное описание действует глобально и распространяется как на основную, так и на вспомогательные функции.

Обратим внимание на еще одно обстоятельство. В функции `Geron` имеются обращения к функции `Line`, а в основной функции — обращение только к функции `Geron`. *Для компилятора важно, чтобы перед вызывающей функцией присутствовал или прототип, или определение вызываемой функции.* Поэтому если из данной программы убрать прототип функции `Line`, то ошибки не будет. Но если одновременно с этим поменять местами определения функций `Line` и `Geron`, то компилятор выдаст сообщение об ошибке.

Рекурсивные определения функций. Как и в Паскале, в языках Си/Си++ допускается рекурсивное определение функций. Проиллюстрируем определение рекурсивной функции на классическом примере вычисления факториала целого положительного числа.

```
long Factor(int n)
{ if (n<0) return 0;
  if (n==0) return 1;
  return n*Factor(n-1);
}
```

В случае если при вызове функции будет задан отрицательный аргумент, она вернет нулевое значение — признак неверного обращения. Выполнение программы происходит аналогично тому, как это описано в разд. 3.13.

Передача значений через глобальные переменные. *Областью действия описания программного объекта называется часть программы, в пределах которой действует (учитывается) это описание.* Если

переменная описана внутри некоторого блока, то она локализована в этом блоке и из других блоков, внешних по отношению к данному, «не видна». Если описание переменной находится вне блока и предшествует ему в тексте программы, то это описание действует внутри блока и называется глобальным. Глобальная переменная «видна» из блока. Например:

```
double x;
int func1()
{int y;… }
void main()
{float y;…}
```

Переменная x является глобальной по отношению к функциям func1, main и, следовательно, может в них использоваться. В функциях func1 и main имеются локальные переменные с одинаковым именем y. Однако это разные величины, никак не связанные друг с другом. Поскольку переменная x является общей для обеих функций, то они могут взаимодействовать через x друг с другом.

При обращении к функции передача значений возможна как через параметры, так и через глобальные переменные. Используя этот механизм, в программах на Си можно реализовывать функции, работающие подобно процедурам в Паскале. В следующей программе решается уже рассматриваемая нами задача получения наибольшего из трех значений.

Пример 6.

```
int z; //Описание глобальной переменной
void MAX(int x, int y)
{if (x>y) z=x; else z=y;}
#include <iostream.h>
void main()
{  int a,b,c;
   cout<<"a="; cin>>a;
   cout<<"b="; cin>>b;
   cout<<"c="; cin>>c;
   MAX(a,b);
   MAX(z,c);
   cout<<"max="<<z;
}
```

Результат выполнения функции MAX заносится в глобальную переменную z, которая «видна» также и из основной функции. Поэтому при втором обращении эта переменная играет одновременно роль аргумента и результата. Здесь оператор обращения к функции выглядит подобно обращению к процедуре в Паскале, а глобальная переменная z играет роль var-параметра.

Классы памяти. Под всякую переменную, используемую в программе, должно быть выделено место в памяти ЭВМ. Выделение памяти может происходить либо на стадии компиляции (компоновки) программы, либо во время ее выполнения. Существуют 4 класса памяти, выделяемой под переменные:

- автоматическая (ключевое слово `auto`);
- внешняя (`extern`);
- статическая (`static`);
- регистровая (`register`).

Под глобальные переменные выделяется место во внешней памяти (не нужно думать, что речь идет о магнитной памяти; это оперативная память класса `extern`). Глобальную переменную можно объявить либо вне программных блоков, либо внутри блока с ключевым словом `extern`. Обычно это делается в тех случаях, когда программный модуль хранится в отдельном файле и, следовательно, отдельно компилируется.

Пусть, например, основная и вспомогательная функции хранятся в разных файлах.

Пример 7.

Файл 1:

```
 ...
int var
void main()
{var=5;
 func();
 cout<<var;
}
```

Файл 2:

```
void func()
{ extern int var;
var=10*var;
}
```

Здесь обмен значениями между основной и вспомогательной функцией `func()` происходит через общую глобальную переменную `var`, для которой во время компиляции выделяется место во внешнем разделе памяти. В результате выполнения данной программы на экран выведется число 50.

Локальные переменные, объявленные внутри блоков, распределяются в автоматической памяти, работающей по принципу стека. Выделение памяти происходит при входе выполнения программы в блок, а при выходе из блока память освобождается. Ключевое слово `auto` писать необязательно (подразумевается по умолчанию).

Статическая память выделяется под переменные, локализованные внутри блока, но в отличие от автоматической памяти не освобождается при выходе из блока. Таким образом, при повторном вхождении в блок статическая переменная сохраняет свое прежнее значение. Пример объявления статической переменной:

```
f()
{static int schet=10; …}
```

Инициализация статической переменной происходит только при первом вхождении в блок. Если инициализация явно не указана, то переменной автоматически присваивается нулевое начальное значение. Статические переменные можно использовать, например, для организации счетчика числа вхождений в блок.

Регистровая память выделяется под локальные переменные. Регистры процессора — самый быстрый и самый маленький вид памяти. Они задействованы при выполнении практически всех операций в программе. Поэтому возможность распоряжаться регистровой памятью лучше оставить за компилятором.

Наконец рассмотрим пример, в котором используются различные способы описания переменных.

Пример 8.

```
int var     //описана внешняя переменная var
main()
{extern int var;  // та же внешняя переменная var
…
}
func1()
{extern int var1;//новая внешняя переменная var1
…             //внешняя var здесь также видна
}
func2()
{…     //здесь переменная var видна, а переменная
}           // var1 не видна
int var1;   //глобально описана переменная var1
func3()
{ int var;  //здесь var – локальная переменная,
…           //видна внешняя переменная var1
}
func4()
{auto int var1;   //здесь var1 – локальная переменная,
…                 //видна внешняя глобальная var
}
```

Упражнения

1. Найти ошибку в программе:

```
#include <iostream.h>
void main()
{int a=1, b=2, c;
 c=sum(a,b);
 cout<<c;
}
```

```
int sum(int x,int y)
{return x+y;}
```

2. Определить результат выполнения программы:

```
#include <iostream.h>
void mul(int,int);
int S;
void main()
{int a=2,b=3;
 mul(a,b);a=2*S;mul(a,b);
 cout<<S;
}
void mul(int x,int y)
{S=x*y;}
```

3. Составить программу для вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя функцию вычисления площади круга.

4. Даны три целых числа. Определить, сумма цифр которого из них больше. Подсчет суммы цифр организовать через функцию.

5. Составить функцию, определяющую, является ли ее целый аргумент простым числом. Использовать эту функцию для подсчета количества простых чисел в последовательности из десяти целых чисел, вводимых с клавиатуры.

6. Описать рекурсивную функцию `stepen(x,n)` от вещественного x ($x \neq 0$) и целого n, которая вычисляет величину x^n согласно формуле

$$x^n = \begin{cases} 1 & \text{при } n = 0, \\ \dfrac{1}{x^{-n}} & \text{при } n < 0, \\ x \cdot x^{n-1} & \text{при } n > 0. \end{cases}$$

7. Даны натуральные числа n и m; найти НОД(n, m). Составить рекурсивную функцию вычисления НОД, основанную на соотношении НОД(n, m)=НОД (m, r), где r — остаток от деления n на m ($n > m$).

4.9. Массивы

Понятие массива знакомо из Паскаля. *Массив — это структура однотипных элементов, занимающих непрерывную область памяти.* С массивом связаны следующие его свойства: имя, тип, размерность, размер.

Формат описания массива следующий:

```
тип_элементов имя [константное_выражение]
```

Константное выражение определяет размер массива, т.е. числ элементов этого массива. Например, согласно описанию

```
int A[10];
```

объявлен массив с именем A, содержащий 10 элементов целого типа. Элементы массива обозначаются индексированными именами. Нижнее значение индекса равно 0:

```
A[0], A[1], A[2], A[3], A[4], A[5], A[6], A[7],
A[8], A[9]
```

В отличие от Паскаля в Си нельзя определять произвольные диапазоны для индексов. Размер массива, указанный в описании, всегда на единицу больше максимального значения индекса.

Размер массива может явно не указываться, если при его объявлении производится инициализация значений элементов. Например:

```
int p[]={2, 4, 6, 10, 1};
```

В этом случае создается массив из пяти элементов со следующими значениями:

```
p[0]=2, p[1]=4, p[2]=6, p[3]=10, p[4]=1
```

В результате следующего объявления массива

```
int M[6]={5, 3, 2};
```

будет создан массив из шести элементов. Первые три элемента получат инициализированные значения. Значения остальных будут либо неопределенными, либо равны нулю, если массив внешний или статический.

Рассмотрим несколько примеров программ обработки одномерных массивов.

Пример 1. Ввод с клавиатуры и вывод на экран одномерного массива.

```
//Ввод и вывод массива
#include <iostream.h>
#include <conio.h>
void main()
{ int i, A[5];
  clrscr();
  for(i=0; i<5; i++)
    { cout<<"A["<<i<<"]="; cin>>A[i];}
  for(i=0; i<5; i++)
    cout<<"A["<<i<<"]="<<A[i]<<"  ";
}
```

Пример 2. Ввод вещественного массива и вычисление среднего значения.

```
//Среднее значение массива
#include <iostream.h>
#include <conio.h>
void main()
{ const n=10;
  int i; double A[n], SA;
  clrscr();
  for(i=0; i<n; i++) {cout<<"A["<<i<<"]=";
cin>>A[i];}
  SA=0;
  for(i=0; i<n;i++) SA=SA+A[i];
  SA=SA/n;
  cout<<"/nСреднее значение="<<SA;
}
```

В этой программе обратите внимание на определение размера массива через константу.

Пример 3. Сортировка массива «методом пузырька».

```
//Сортировка массива
#include <iostream.h>
#include <conio.h>
void main()
{ int X[]={6,4,9,3,2,1,5,7,8,10};
  int i,j,n,A;
  clrscr();
  n=sizeof(X)/sizeof(X[0]);
  for(i=0; i<n-1; i++)
  for(j=0; j<n-1-i; j++)
   if(X[j]>X[j+1]) {A=X[j]; X[j]=X[j+1]; X[j+1]=A;}
  for(i=0; i<n; i++) cout<<X[i]<<" ";
}
```

Алгоритм сортировки массива «методом пузырька» описан в разд. 3.17. В данной программе массив инициализирован. Его размер равен числу заданных значений. Чтобы сделать программу универсальной по отношению к размеру массива, значение размера вычисляется автоматически и заносится в переменную `n`. Для этого используется операция `sizeof()` — определение размера в байтах. Результат `sizeof(X)` равен размеру в памяти всего массива `X` — 20 байтам. Результат `sizeof(X[0])` равен размеру одного элемента массива — 2 байтам. Отношение этих величин равно 10 — числу элементов массива. Внимательно проанализируйте организацию перебора значений параметров вложенных циклов — `i, j`.

В результате выполнения этой программы на экран выведется упорядоченная числовая последовательность

```
1 2 3 4 5 6 7 8 9 10
```

Многомерные массивы. Двумерный массив трактуется как одномерный массив, элементами которого является массив с указанным в описании типом элементов. Например, оператор

```
float R[5][10];
```

объявляет массив из пяти элементов, каждый из которых есть массив из десяти вещественных чисел. Отдельные величины этого массива обозначаются именами с двумя индексами: `R[0][0]`, `R[0][1]`, ..., `R[4][9]`. Объединять индексы в одну пару скобок нельзя, т.е. запись `R[2,3]` ошибочна.

Пример описания трехмерного массива:

```
double X[3][7][20];
```

Как и в Паскале, порядок расположения элементов многомерного массива в памяти такой, что прежде всего меняется последний индекс, затем предпоследний и т.д., и лишь один раз пробегает свои значения первый индекс.

При описании многомерных массивов их также можно инициализировать. Делать это удобно так:

```
int M[3][3]={ 11,12,13,
              21,22,23,
              31,32,33 };
```

Рассмотрим примеры программ обработки матриц — числовых двумерных массивов.

Пример 4. Вычисление и вывод на экран таблицы умножения в форме матрицы Пифагора.

```
// Матрица Пифагора
#include <stdio.h>
#include <conio.h>
void main()
{ int i,j, A[10][10];
  clrscr();
  for(i=1; i<=9; i++)
  {  for(j=1; j<=9; j++)
     { A[i][j]=i*j;
        printf("%5d",A[i][j]);
     }
     printf("\n");
  }
}
```

По данной программе в двумерном массиве A не будут заполнены нулевая строка и нулевой столбец. По-прежнему интерпретируем первый индекс двумерного массива как номер строки матрицы, а второй индекс — как номер столбца.

Пример 5. Заполнение матрицы случайными числами в диапазоне от 0 до 99 и поиск в ней максимального значения.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <stdlib.h>
#define n 5
void main()
{ int i,j,ImaxA,JmaxA,A[n][n];
  clrscr();
  randomize(); //Установка датчика случайных чисел
  for(i=0; i<n; i++)
  {  for(j=0; j<n; j++)
     { A[i][j]=rand()%100;
       cout<<setw(6)<<A[i][j];
     }
     cout<<endl;
  }
 ImaxA=JmaxA=0;
 for(i=0; i<n; i++)
 {  for(j=0; j<n; j++)
     if (A[i][j]>A[ImaxA][JmaxA])
     { ImaxA=i; JmaxA=j; }
 }
cout<<"Максимальное_значение:["<<ImaxA<<"]["
<<JmaxA<<"]=" <<A[ImaxA][JmaxA];
}
```

В результате тестирования этой программы получен следующий результат:

```
46    23    57    35    18
8     48    68     4    70
56    98    16    71    40
70    84    66    67    11
20    44    37    57    38

Максимальное значение: A[2][1]=98
```

В данной программе имеются новые элементы, использование которых требует пояснения. В стандартной библиотеке с заголовочным файлом stdlib.h содержится функция, прототип которой имеет вид: **int** rand(void).

Результатом этой функции является целое случайное число из диапазона от 0 до RAND_MAX. Значение константы RAND_MAX определено в заголовочном файле stdlib.h и обычно равно 32767 — максимально допустимому целому числу. Для получения случайных чисел в диапазоне от 0 до $N-1$ достаточно вычислить остаток от целого деления rand() на N. Функция с прототипом void randomize(void) выполняет первоначальную настройку датчика случайных чисел так, чтобы последовательность чисел не повторялась при повторном выполнении программы.

Другим новым элементом в данной программе является использование *манипуляторов* для управления потоковым выводом с помощью стандартного объекта cout. Манипуляторы объявляются в заголовочном файле iomanip.h. Манипулятор setw(n) влияет на формат следующего элемента выходного потока. Он указывает на то, что значение будет выводиться в n позиций на экране (в программе n = 6). Другой использованный манипулятор — endl — обозначает конец строки и переводит экранный курсор на начало новой строки. Его действие аналогично действию управляющего символа \n.

Упражнения

1. Дан вектор $\{z_i\}$, $i = 1, \ldots, 50$. Вычислить длину этого вектора:

$$L = \sqrt{z_1^2 + z_2^2 + \ldots + z_{50}^2}.$$

2. Вычислить полином 10-й степени по формуле Горнера:

$$a_{10}x^{10} + a_9x^9 + \ldots + a_1x + a_0 = ((\ldots(a_{10}x + a_9)x + a_8)x + \ldots + a_1)x + a_0.$$

3. Для вектора $\{x_i\}$, $i = 1, \ldots, 20$, подсчитать количество компонент, значения которых лежат в интервале [0,1].

4. Даны два вектора $\{x_i\}$, $\{y_i\}$, $i = 1, \ldots, 10$, упорядоченные по возрастанию. Слить их в один вектор $\{z_i\}$, $i = 1, \ldots, 20$ так, чтобы сохранилась упорядоченность.

5. Дан массив, состоящий из 100 целых чисел.

а) Вывести все числа, которые встречаются в этом массиве несколько раз.

б) Вывести все числа, которые встречаются в массиве только по одному разу.

6. С помощью датчика случайных чисел заполнить двоичную матрицу 5×10. Определить номер строки с наибольшим количеством нулей.

7. Транспонировать целочисленную матрицу 5×5, т.е. отразить относительно главной диагонали.

8. В двоичной матрице 10×10 найти совпадающие строки.

4.10. Указатели

Понятие указателя знакомо читателю из разд. 3.21, в котором описывается ссылочный тип данных в Паскале. Смысл этого понятия в Си/Си++ остается тем же: *указатель — это адрес поля памяти, занимаемого программным объектом.*

Пусть в программе определены три переменные разных типов:

```
int a=5;
char c='G';
float r=1.2E8;
```

Эти величины разместились в памяти компьютера следующим образом:

Память	FFC0	FFC1	FFC2	FFC3	FFC4	FFC5	FFC6
Переменные	a		c	r			
Значения	5		'G'	$1.2*10^8$			

Операция & — адрес. Применение этой операции к имени переменной дает в результате ее адрес в памяти. Для переменных из данного выше примера: `&a` равно `FFC0`, `&c` – `FFC2`, `&r` – `FFC3`.

Описание указателей. Для хранения адресов используются переменные типа «указатель». Формат описания таких переменных следующий:

```
тип *имя_переменной
```

Примеры описания указателей:

```
int *pti;   char *ptc;   float *ptf;
```

После такого описания переменная `pti` может принимать значение указателя на величину целого типа; переменная `ptc` предназначена для хранения указателя на величину типа `char`; переменная `ptf` — на величину типа `float`.

Указателям могут присваиваться значения адресов объектов только того типа, с которым они описаны. В нашем примере допустимы операторы

```
pti=&a; ptc=&c; ptf=&r;
```

В результате указатели примут следующие значения:

`pti` – `FFC0`, `ptc` – `FFC2`, `ptf` – `FFC3`.

Как и для других типов данных, значения указателей могут инициализироваться при описании. Например:

```
int a=5;  int *pti=&a;
char c='G';  char *ptc=&c;
float r=1.2E8;  float *ptf=&r;
```

В заголовочном файле `stdio.h` определена константа — нулевой указатель с именем `NULL`. Ее значение можно присваивать указателю. Например:

```
ptf=NULL;
```

Не надо думать, что после этого указатель `ptf` будет ссылаться на нулевой байт памяти. Нулевой указатель обозначает отсутствие конкретного адреса ссылки.

Использованный в описаниях указателей символ * (звездочка) в данном контексте является знаком *операции разадресации.* С ее помощью можно сослаться через указатель на соответствующую переменную.

После приведенных выше описаний в записи выражений этой программы взаимозаменяемыми становятся `a` и `*pti`, `c` и `*ptc`, `r` и `*ptf`. Например, два оператора

```
x=a+2;   и   x=*pti+2;
```

тождественны друг другу. В результате выполнения оператора

```
cout<<*pti<<a;
```

на экран выведется `55`.

Операции над указателями. Записывая выражения и операторы, изменяющие значения указателей, необходимо помнить главное правило: *единицей изменения значения указателя является размер соответствующего ему типа.*

Продемонстрируем это правило на определенных выше указателях. Выполнение операторов

```
pti=pti+1;   или     pti++;
```

изменит значение указателя `pti` на 2, в результате чего он примет значение `FFC2`. В результате выполнения оператора `pti--;` значение указателя уменьшится на 2 и станет равным `FFBE`.

Аналогично для указателей других типов:

`ptc++;` увеличит значение указателя на 1;

`ptf++;` увеличит значение указателя на 4.

Использование указателей для передачи параметров функции. Рассматривая ранее правила использования функций, мы обращали внимание на то, что в языке Си возможна только односторонняя передача значений фактических параметров из вызывающей программы к формальным параметрам вызываемой функции. Возвращаемое значение несет сама функция, используемая в качестве операнда в выражении. Отсюда, казалось бы, следует неукоснительное правило: в процессе выполнения функции не могут изменяться значения переменных в вызывающей программе. Однако это правило можно обойти, если в качестве параметров функции использовать указатели.

В следующем примере функция swap() производит обмен значениями двух переменных величин, заданных своими указателями в аргументах.

```
void swap(int *a,int *b)
{ int c;
  c=*a; *a=*b; *b=c;
}
```

Если в основной программе имеется следующий фрагмент:

```
int x=1,y=2;
swap(&x,&y);
printf("x=%d y=%d",x,y);
```

то на экран будет выведено:

```
x=2   y=1
```

т. е. переменные x и y поменялись значениями.

Все выглядит очень похоже на то, как если бы в Паскале использовали процедуру обмена с var-параметрами. И тем не менее передача параметров здесь тоже происходит по значению, только этими значениями являются указатели. После обращения к функции указатель a получил адрес переменной x, указатель b — адрес переменной y. После этого переменная x в основной программе и разадресованный указатель *a в функции оказываются связанными с одной ячейкой памяти; так же — y и *b.

Таким образом, можно сделать вывод о том, что *использование указателей в параметрах функции позволяет моделировать работу процедур.*

Указатели и массивы. Сейчас мы обсудим одно неожиданное обстоятельство в языке Си (неожиданное, с точки зрения человека, изучающего Си после Паскаля).

Имя массива трактуется как указатель-константа на массив.

Пусть, например, в программе объявлен массив:

```
int X[10];
```

В таком случае X является указателем на нулевой элемент массива в памяти компьютера. В связи с этим истинным является отношение

```
X==&X[0]
```

Отсюда следует, что для доступа к элементам массива кроме индексированных имен можно использовать разадресованные указатели по принципу:

имя[индекс] тождественно *(имя + индекс)

Например, для описанного выше массива X взаимозаменяемы следующие обозначения элементов:

X[5], или *(X+5), или *(5+X).

Напоминаем, что для указателей работают свои правила сложения. Поскольку X — указатель на величину целого типа, то X+5 увеличивает значение адреса на 10.

В языке Си символ [*играет роль знака операции сложения адреса массива с индексом элемента массива.*

Из сказанного должно быть понятно, почему индекс первого элемента массива всегда нуль. Его адрес должен совпадать с адресом массива:

```
X[0]==*(X+0)
```

Поскольку имя массива является указателем-константой, то его нельзя изменять в программе, т. е. ему нельзя ничего присваивать. Например, если описаны два одинаковых по структуре массива

```
int X[10],Y[10];
```

то оператор присваивания X=Y будет ошибочным. Такое возможно в Паскале, но недопустимо в Си. Пересылать значения одного массива в другой можно только поэлементно.

Теперь рассмотрим двумерные массивы. Пусть в программе присутствует описание:

```
int P[5][10];
```

Это матрица из пяти строк и десяти чисел в каждой строке. Двумерный массив расположен в памяти в последовательности по строкам. По-прежнему P является указателем-константой на массив, т. е. на элемент P[0][0]. Индексированное имя P[i] обозначает *i*-ю строку. Ему тождественно следующее обозначение в форме разадресованного указателя:

```
*(P+i*10)
```

Обращение к элементу массива P[2][4] можно заменить на *(P+2*10+4). В общем случае эквивалентны обозначения:

```
P[i][j]    и    *(P+i*10+j)
```

Здесь дважды работает операция «квадратная скобка». Последнее выражение можно записать иначе, без явного указания на длину строки матрицы P:

```
*(*(P+i)+j).
```

Очевидно, что по индукции для ссылки на элемент трехмерного массива A[i][j][k] справедливо выражение

```
*(*(*(A+i)+j)+k) и т.д.
```

Массив как параметр функции. Обсудим эту тему на примерах.

Пример 1. Составим программу решения следующей задачи. Дана вещественная матрица A[M][N]. Требуется вычислить и вывести евклидовы нормы строк этой матрицы.

Евклидовой нормой вектора называют корень квадратный из суммы квадратов его элементов:

$$L = \sqrt{\sum_{i=1}^{n} x_i^2} .$$

Если строку матрицы рассматривать как вектор, то данную формулу надо применить к каждой строке. В результате получим M чисел.

Определение функции вычисления нормы произвольного вектора:

```
double Norma(int n, double X[])
{ int i;
  double S=0;
  for(i=0; i<n; i++) S+=X[i]*X[i];
  return sqrt(S);
}
```

Заголовок этой функции можно было записать и в такой форме:

```
double Norma(int n, double *X)
```

В обоих случаях в качестве второго параметра функции используется указатель на начало массива. Во втором варианте это более очевидно, однако оба варианта тождественны.

При вызове функции `Norma()` в качестве второго фактического параметра должен передаваться адрес начала массива (вектора).

Рассмотрим фрагмент основной программы, использующей данную функцию для обработки матрицы размером 5×10.

```
void main()
{  double A[5][10];  int i;
   //Ввод матрицы
. . . . . .
   //Вычисление и вывод нормы строк
   for(i=0; i<5; i++)
   cout<<"Норма"<<i<<"-й строки=" <<Norma(10,A[i]);
}
```

В обращении к функции второй фактический параметр `A[i]` является указателем на начало i-й строки матрицы `A`.

Пример 2. Заполнить двумерную матрицу случайными целыми числами в диапазоне от 0 до 99. Отсортировать строки полученной матрицы по возрастанию значений. Отсортированную матрицу вывести на экран.

```
#include <iostream.h>
#include <iomanip.h>
```

```
#include <conio.h>
#include <stdlib.h>
const n=5;//Глобальное объявление константы
//Прототипы функций
void Matr(int M[][n]);
void Sort(int, int X[]);
//Основная программа
void main()
{
  int i,j, A[n][n];
  clrscr();
  cout<<"\n"<<"Матрица до сортировки:"<<"\n";
  Matr(A);
  for(i=0; i<n; i++) Sort(n, A[i]);
  cout<<"\n"<<"Матрица после сортировки:"<<"\n";
  for(i=0; i<n; i++)
  {  for(j=0; j<n; j++)
     cout<<setw(6)<<A[i][j];
     cout<<endl;}
}
// Функция сортировки вектора
void Sort(int k, int X[])
{int i,j, Y;
 for(i=0; i<k-1; i++)
   for(j=0; j<k-i-1; j++)
    if(X[j]>X[j+1]) {Y=X[j]; X[j]=X[j+1];
X[j+1]=Y;}
}
 //Функция заполнения матрицы и вывода на экран
 void Matr(int M[][n])
 {int i,j;
  randomize(); //Установка датчика случайных чисел
  for(i=0; i<n; i++)
  {  for(j=0; j<n; j++)
     { M[i][j]=rand()%100; cout<<setw(6)<<M[i][j];}
     cout<<endl;
   }
 }
```

Здесь все выглядит совсем как при использовании процедур на Паскале. Обратите внимание на прототип и заголовок функции `Matr()`. В них явно указывается вторая размерность параметра-матрицы. Первую тоже можно указать, но это необязательно. Как уже говорилось выше, двумерный массив рассматривается как одномерный массив, элементами которого являются массивы (в данном случае — строки матрицы). Компилятору необходимо «знать»

размер этих элементов. Для массивов большей размерности (3, 4 и т.д.) в заголовках функций необходимо указывать все размеры, начиная со второго.

При обращении к функции `Matr()` фактическим параметром является указатель на начало двумерного массива `A`, а при обращении к функции `Sort()` — указатели на начало строк.

В итоге тестирования программы получен следующий результат.

Матрица до сортировки:

```
46   23   57   35   18
 8   48   68    4   70
56   98   16   71   40
70   84   66   67   11
20   44   37   57   38
```

Матрица после сортировки:

```
18   23   35   46   57
 4    8   48   68   70
16   40   56   71   98
11   66   67   70   84
20   37   38   44   57
```

Упражнения

1. В оперативной памяти вектор `int X[10]` расположен, начиная с адреса `B7F0`. Какие значения примут выражения:

а) `X+1`; б) `X+5`; в) `X–4`?

2. В программе объявлен массив:

```
int P[]={0,2,4,5,6,7,9,12};
```

Какие значения примут выражения:

а) `P[3]`; б) `*P`; в) `*(P+4)`; г) `*(P+P[2])`?

3. Составить функцию сортировки значений трех переменных *a*, *b*, *c* в порядке возрастания.

4. Составить функцию заполнения целочисленного одномерного массива случайными значениями в диапазоне от 0 до *N*.

5. Составить функцию вычисления среднего значения элементов вещественного одномерного массива. Использовать эту функцию в основной программе, определяющей в матрице номер строки с наибольшим средним значением.

4.11. Обработка символьных строк

В языках Си/Си++ нет специально определенного строкового типа данных, как в Турбо Паскале. Символьные строки организуются как массивы символов, последним из которых является символ `\0`, внутренний код которого равен нулю. Отсюда следует одно важное преимущество перед строками в Турбо Паскале, где размер строки не может превышать 255 (длина указывается в первом байте), — на длину символьного массива в Си нет ограничения.

Строка описывается как символьный массив. Например:

```
char STR[20];
```

Одновременно с описанием строка может инициализироваться. Возможны два способа инициализации строки — с помощью строковой константы и в виде списка символов:

```
char S[10]="строка";
char S[]="строка";
char S[10]={'с','т','р','о','к','а','\0'};
```

По результату первого описания под строку `S` будет выделено 10 байт памяти, из них первые 7 получат значения при инициализации (седьмой — нулевой символ). Второе описание сформирует строку из семи символов. Третье описание по результату равнозначно первому. Конечно, можно определить символьный массив и так:

```
char S[10]={'с','т','р','о','к','а'};
```

т.е. без нулевого символа в конце. Но это приведет к проблемам с обработкой такой строки, так как будет отсутствовать ориентир на его окончание.

Отдельные символы строки идентифицируются индексированными именами. Например, в описанной выше строке `S[0]='с'`, `S[5]='а'`.

Обработка строк обычно связана с перебором всех символов от начала до конца. Признаком конца такого перебора является обнаружение нулевого символа. В следующей программе производятся последовательная замена всех символов строки на звездочки и подсчет длины строки.

Пример 1.

```
 //Замена символов на звездочки
#include <stdio.h>
#include <conio.h>
void main()
{  char S[]="fh5j";
   int i=0;
   clrscr();
   puts(S);
```

```
    while(S[i])
    {S[i++]='*'; puts(S);}
    printf("\n,Длина строки=",i);
}
```

В результате выполнения программы на экране получим:

```
fh5j
*h5j
**5j
***j
****
Длина строки=4
```

В этой программе цикл повторяет свое выполнение, пока `S[i]` не получит значение нулевого символа.

Для вывода строки на экран в стандартной библиотеке `stdio` имеется функция `puts()`. Аргументом этой функции указывается имя строки. В этой же библиотеке есть функция ввода строки с клавиатуры с именем `gets()`. В качестве аргумента указывается имя строки, в которую производится ввод.

Среди стандартных библиотек Си/Си++ существует библиотека функций для обработки строк. Ее заголовочный файл — `string.h`. В следующем примере используется функция определения длины строки из этой библиотеки. Имя функции — `strlen()`. В качестве аргумента указывается имя строки.

Пример 2. Ввести символьную строку. Перевернуть (обратить) эту строку. Например, если ввели строку «abcdef», то в результате в ней должны получить «fedcba».

```
//Обращение строки
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{  char C,S[10];
   int i;
   clrscr();
   printf("Введите строку");
   gets(S);
   for(i=0; i<=(strlen(S)-1)/2; i++)
   { C=S[i]; S[i]=S[strlen(S)-i-1];
S[strlen(S)-i-1]=C;}
   printf("\nПеревернутая строка:");
   puts(S);
}
```

Идея алгоритма состоит в перестановке символов, расположенных на одинаковом расстоянии от начала и конца строки. Пе-

ребор элементов строки доходит до ее середины. Составляя подобные программы, не надо забывать, что индекс первого символа строки — 0, а индекс последнего на единицу меньше длины строки.

Строка как параметр функции. Использование строк в качестве параметра функции аналогично рассмотренному выше использованию массивов других типов. Необходимо помнить, что имя массива есть указатель на его начало. Однако для строк имеется одно существенное отличие от массивов других типов: *имя строки является указателем-переменной*, и, следовательно, его значение может подвергаться изменению. Стандарт Си рекомендует в заголовках функций, работающих со строками, явно использовать обозначение символьного указателя.

Пример 1. Запишем определение функции вычисления длины строки (аналог стандартной функции `strlen()`).

```
int length(char *s)
{ int k;
  for(k=0; *s++!='\0'; k++);
  return k;
}
```

Здесь функция использует явный механизм работы с указателем. Изменение значения указателя `s` допустимо благодаря тому, что он является переменной. Еще раз напомним, что для числовых массивов этого делать нельзя! Если соблюдать данное ограничение и для строк, то условное выражение в операторе `for` следовало бы писать так: `*(s+k)!='\0'` или `s[k]!='\0'`.

Обдумайте это обстоятельство!

Пример 2. Оформим программу обращения строки в виде функции и напишем основную программу, использующую ее. Алгоритм обращения реализуем иначе, чем в рассмотренной выше программе. Для определения длины строки не будем пользоваться стандартной функцией. Для вывода строки на экран применим функцию `printf()` со спецификатором `%s` (работает аналогично функции `puts()`).

```
//Обращение строки
#include <stdio.h>
#include <conio.h>
//Прототипы функций
int length(char*str);
void invers(char*str);
// Основная программа
void main()
{  char S[]="123456789";
   clrscr();
```

```
   invers(S);//Вызов функции обращения строки
   printf("\n%s",S);
}
//Функция вычисления длины строки
int length(char *s)
{ int k;
  for(k=0; *s++!='\0'; k++);
  return k;
}
//Функция обращения строки
void invers(char *e)
{ char c;
  int i,j,m;
  m=length(e); //Вызов функции length
  for(i=0, j=m-1; i<j; i++, j--)
  { c=e[i]; e[i]=e[j]; e[j]=c; }
}
```

В результате выполнения этой программы на экране получим строку:

```
9 8 7 6 5 4 3 2 1 0
```

И снова обратим внимание на то, что здесь функция `invers()` работает аналогично паскалевской процедуре. Обмен параметрами через указатели имитирует обмен по имени.

Пример 3. Описать функцию вставки символа в строку. Параметры функции: строка, позиция вставки, вставляемый символ. Использовать эту функцию в основной программе, где исходная строка, номер позиции и вставляемый символ задаются вводом.

```
// Вставка символа в строку
#include <stdio.h>
#include <string.h>
#include <conio.h>
void INSERT(char *str, int p, char c)
{ int i;
   for(i=strlen(str); i>=p; i--)
   str[i+1]=str[i];
   str[p]=c;
}
void main()
{  char c, S[100]; int n;
   clrscr();
   puts("Введите строку:"); gets(S);
   puts("Введите позицию вставки:");
    scanf("%d",&n);
   puts("Введите символ:"); c=getche();
   INSERT(S,n,c);
```

```
    puts("\nРезультат:"); puts(S);
}
```

Вот вариант диалога на экране при выполнении этой программы:

```
Введите строку:
0123456789
Введите позицию вставки:
4
Введите символ:
*
Результат:
0123*456789
```

В этой программе наряду с рассмотренными ранее функциями для ввода и вывода строки `gets()` и `puts()` используется функция чтения символа с клавиатуры `getche()` из библиотеки `stdio.h`. Ее прототип имеет вид: `int getche(void)`. Она возвращает значение символа, введенного с клавиатуры, которое может быть присвоено символьной переменной.

Упражнения

1. Составить программу подсчета количества цифр в данной строке.
2. Составить программу, которая по данной символьной строке формирует числовой массив, содержащий коды символов, составляющих строку.
3. Составить функцию, определяющую тождественность двух данных строк.
4. Составить программу, удаляющую в данной строке каждый символ с четным номером.
5. Составить функцию конкатенации (слияния) двух строк. В основной программе использовать эту функцию для слияния четырех строк.
6. Составить функцию, переводящую десятичное целое число, представленное в символьном виде, в соответствующую величину целого типа.
7. Составить функцию, переводящую десятичное вещественное число, представленное в символьном виде, в соответствующую величину плавающего типа.

4.12. Структуры и объединения

В языках Си/Си++ понятие структуры аналогично понятию записи (`record`) в Паскале. Это структурированный тип данных, представляющий собой поименованную совокупность разнотипных элементов. Тип `структура` обычно используется при разработке информационных систем, баз данных.

Правила использования структур обсудим на примере, аналогичном тому, который рассматривался в разделе 3.20 применительно к Паскалю.

Сведения о выплате студентам стипендии требуется организовать в виде, показанном на рис. 46.

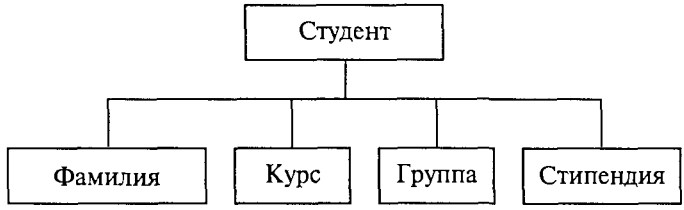


Рис. 46

Элементы такой структуры (фамилия, курс, группа, стипендия) называются полями. Каждому полю должно быть поставлено в соответствие имя и тип.

Формат описания структурного типа следующий:

```
struct имя_типа
          {определения_элементов};
```

В конце обязательно ставится точка с запятой (это оператор).

Для рассмотренного примера определение соответствующего структурного типа может быть следующим:

```
struct student {char fam[30];
                int kurs;
                char grup[3];
                float stip;
               };
```

После этого `student` становится именем структурного типа, который может быть назначен некоторым переменным. В соответствие со стандартом Си это нужно делать так:

```
struct student stud1, stud2;
```

Правила Си++ разрешают в этом случае служебное слово `struct` опускать и писать

```
student stud1, stud2;
```

Здесь `stud1` и `stud2` — переменные структурного типа.

Допускаются и другие варианты описания структурных переменных. Можно вообще не задавать имя типа, а описывать сразу переменные:

```
struct {char fam[30];
        int kurs;
```

```
    char grup[3];
    float stip;
} stud1, stud2, *pst;
```

В этом примере кроме двух переменных структурного типа объявлен указатель `pst` на такую структуру. В данном описании можно было сохранить имя структурного типа `student`.

Обращение к элементам (полям) структурной величины производится с помощью *уточненного* имени следующего формата:

```
имя_структуры.имя_элемента
```

Снова все похоже на Паскаль. Примеры уточненных имен для описанных выше переменных:

```
stud1.fam;    stud1.stip
```

Значения элементов структуры могут определяться вводом, присваиванием, инициализацией. Пример инициализации в описании:

```
student stud1={"Кротов", 3, "Ф32", 350};
```

Пусть в программе определен указатель на структуру

```
student *pst, stud1;
```

Тогда после выполнения оператора присваивания

```
pst=&stud1;
```

к каждому элементу структурной переменной `stud1` можно обращаться тремя способами. Например, для поля `fam`

`stud1.fam` или `(*pst).fam` или `pst->fam`

В последнем варианте используется *знак операции доступа к элементу структуры*: —>. Аналогично можно обращаться и к другим элементам этой переменной:

```
pst->FIO, pst->grup, pst->stip.
```

Поля структуры могут сами иметь структурный тип. Такие величины представляют многоуровневые деревья.

Допускается использование массивов структур. Например, сведения о 100 студентах могут храниться в массиве, описанном следующим образом:

```
student stud[100];
```

Тогда сведения об отдельных студентах будут обозначаться, например, так: `stud[1].fam`, `stud[5].kurs` и т.п. Если нужно взять первую букву фамилии 25-го студента, то следует писать: `stud[25].fam[0]`.

Пример 1. Ввести сведения об N студентах. Определить фамилии студентов, получающих самую высокую стипендию.

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

```
#include <conio.h>
void main()
{
 const N=30; int i; float maxs;
 struct student {char fam[15];
                 int kurs;
                 char grup[3];
                 float stip;
                };
 student stud[N];
 clrscr();
 for(i=0;i<N;i++)
 { printf("%d-й студент",i);
   printf("\n"Фамилия:");scanf("%s",&stud[i].fam);
   printf("Курс:"); scanf("%d",&stud[i].kurs);
   printf("Группа:"); scanf("%s",&stud[i].grup);
   printf("Стипендия:"); scanf("%f",&stud[i].stip);
 }
maxs=0;
for(i=0;i<N;i++)
 if(stud[i].stip>maxs) maxs=stud[i].stip;
printf("\n Студенты,получающие максимальную
стипендию %f руб.",maxs);
 for(i=0; i<N; i++)
 if(stud[i].stip==maxs) printf("\n%s",stud[i].fam);
}
```

Элемент структуры типа поля битов. Использование структуры в программе на Си позволяет работать с отдельными битами, т.е. с разрядами двоичного кода. Для этого используются элементы структуры типа поля битов. Формат структуры, содержащий поля битов, следующий:

```
struct имя_структуры
       { тип имя_поля_1: длина_в_битах;
         тип имя_поля_2: длина_в_битах;
          . . . . .
         тип имя_поля_N: длина_в_битах;
       };
```

В качестве типа полей могут использоваться спецификаторы `int, unsigned, signed`. Минимальной величиной такого типа может быть структура, состоящая всего из одного битового поля. Пример описания такой структуры:

```
struct onebit
        { unsigned bit:1;
        } cod;
```

Конечно, для переменной cod в памяти будет выделено 8 бит (1 байт), но использоваться будет только один первый бит.

Объединение. Объединение — это еще один структурированный тип данных. Объединение похоже на структуру и в своем описании отличается от структуры тем, что вместо ключевого слова struct используется слово union.

```
union имя_типа
        {определения_элементов};
```

Объединение отличается от структуры способом организации во внутренней памяти. *Все элементы объединения в памяти начинаются с одного байта.*

Пусть в программе описана структура:

```
struct S
     { int i;
       char ch;
       long int L;
     };
```

Расположение ее элементов в памяти будет следующим:

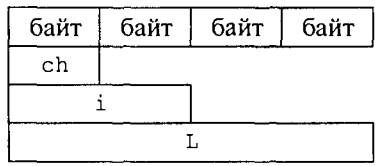
байт	байт	байт	байт	байт	байт	байт
i		ch	L			

Элементы структуры занимают последовательные ячейки памяти с размером, соответствующим типу. Общий размер структуры равен сумме длин полей.

А теперь рассмотрим объединение со следующим описанием:

```
union S
     { int i;
       char ch;
       long int L;
     };
```

Величина с таким типом в памяти будет расположена следующим образом:



Поля объединения накладываются друг на друга. Общий объем занимаемой памяти равен размеру самого большого поля.

Изменение значения любого поля объединения меняет значения других полей.

Пример 2. Составим программу решения следующей задачи: с клавиатуры вводится символ. Вывести на экран двоичный код этого символа.

```
//Получение двоичного кода символа
#include <stdio.h>
#include <conio.h>
//Структура битовых полей
struct byte{int b1:1;
            int b2:1;
            int b3:1;
            int b4:1;
            int b5:1;
            int b6:1;
            int b7:1;
            int b8:1;
};
//Объединение - переменная
union bits
      {char ch;
       byte cod;} u;
//Прототип функции декодирования
void decode(bits x);
//Основная программа
void main()
{ do{u.ch=getche();  //Ввод символа с клавиатуры
      printf(":");
      decode(u);
     } while(u.ch!='q'); //Символ q- признак конца
                         // ввода
 }
//Функция декодирования
 void decode(bits u)
 { if (u.cod.b8) printf("1"); else printf("0");
   if (u.cod.b7) printf("1"); else printf("0");
   if (u.cod.b6) printf("1"); else printf("0");
   if (u.cod.b5) printf("1"); else printf("0");
   if (u.cod.b4) printf("1"); else printf("0");
   if (u.cod.b3) printf("1"); else printf("0");
   if (u.cod.b2) printf("1"); else printf("0");
   if (u.cod.b1) printf("1"); else printf("0");
   printf("\n");
 }
```

В этой программе переменная-объединение u содержит два элемента: символьное поле ch и битовую структуру cod, которые накладываются друг на друга. Таким образом, оказывается воз-

можным получить доступ к каждому биту кода символа. Работа программы заканчивается после ввода символа q.

Вот вариант результатов работы данной программы:

```
s:01110011
d:01100100
J:01101010
a:01100001
b:01100010
c:01100011
d:01100100
q:01110001
```

Упражнения

1. Сведения о каждом химическом элементе из периодической таблицы Д. И. Менделеева представить в виде структуры. Написать программу ввода таблицы в память компьютера.
2. Представить координаты точки в трехмерном пространстве в виде структуры, состоящей из трех вещественных полей. Написать программу ввода координат двух точек и вычисления расстояния между ними.
3. Рассматривая комплексное число как структуру, состоящую из двух вещественных полей, составить функции выполнения четырех арифметических операций с комплексными числами.

4.13. Потоковый ввод-вывод в стандарте Си

Под вводом-выводом в программировании понимается процесс обмена информацией между оперативной памятью и внешними устройствами: клавиатурой, дисплеем, магнитными накопителями и т. п. Ввод — это занесение информации с внешних устройств в оперативную память, а вывод — вынос информации из оперативной памяти на внешние устройства. Такие устройства, как дисплей и принтер, предназначены только для вывода; клавиатура — устройство ввода. Магнитные накопители (диски, ленты) используются как для ввода, так и для вывода.

Основным понятием, связанным с информацией на внешних устройствах ЭВМ, является понятие файла. Всякая операция ввода-вывода трактуется как операция обмена с файлами: ввод — это чтение из файла в оперативную память; вывод — запись информации из оперативной памяти в файл. Поэтому вопрос об организации в языке программирования ввода-вывода сводится к вопросу об организации работы с файлами.

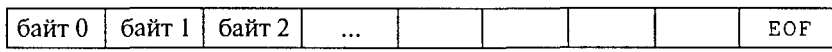
Вспомним, что в Паскале мы использовали представления о внутреннем и внешнем файле. Внутренний файл — это перемен-

ная файлового типа, являющаяся структурированной величиной. Элементы файловой переменной могут иметь разный тип и, соответственно, разную длину и форму внутреннего представления. Внутренний файл связывается с внешним (физическим) файлом с помощью стандартной процедуры `Assign`. Один элемент файловой переменной становится отдельной записью во внешнем файле и может быть прочитан или записан с помощью одной команды. Попытка записать в файл или прочитать из него величину, не совпадающую по типу с типом элементов файла, приводит к ошибке.

Аналогом понятия внутреннего файла в языках Си/Си++ является понятие потока. Отличие от файловой переменной Паскаля состоит в том, что потоку в Си не ставится в соответствие тип. *Поток — это байтовая последовательность, передаваемая в процессе ввода-вывода.*

Поток должен быть связан с каким-либо внешним устройством или файлом на диске. В терминологии Си это звучит так: *поток должен быть направлен на какое-то устройство или файл.*

Основные отличия файлов в Си состоят в следующем: здесь отсутствует понятие типа файла и, следовательно, фиксированной структуры записи файла. Любой файл рассматривается как байтовая последовательность:



Стрелочкой обозначен указатель файла, определяющий текущий байт файла. `EOF` является стандартной константой — признаком конца файла.

Существуют стандартные потоки и потоки, объявляемые в программе. Последние обычно связываются с файлами на диске, создаваемыми программистом. Стандартные потоки назначаются и открываются системой автоматически. С началом работы любой программы открываются 5 стандартных потоков, из которых основными являются следующие:

- `stdin` — поток стандартного ввода (обычно связан с клавиатурой);
- `stdout` — поток стандартного вывода (обычно связан с дисплеем);
- `stderr` — вывод сообщений об ошибках (связан с дисплеем).

Кроме этого, открывается поток для стандартной печати и дополнительный поток для последовательного порта.

Работая ранее с программами на Си, используя функции ввода с клавиатуры и вывода на экран, мы уже неявно имели дело с первыми двумя потоками. А сообщения об ошибках, которые си-

стема выводила на экран, относились к третьему стандартному потоку. Поток для работы с дисковым файлом должен быть открыт в программе.

Работа с файлами на диске. Работа с дисковым файлом начинается с объявления указателя на поток. Формат такого объявления:

```
FILE *имя_указателя;
```

Например:

```
FILE *fp;
```

Слово `FILE` является стандартным именем структурного типа, объявленного в заголовочном файле `stdio.h`. В структуре `FILE` содержится информация, с помощью которой ведется работа с потоком, в частности: указатель на буфер, указатель (индикатор) текущей позиции в потоке и т.д.

Следующий шаг — открытие потока, которое производится с помощью стандартной функции `fopen()`. Эта функция возвращает конкретное значение для указателя на поток и поэтому ее значение присваивается объявленному ранее указателю. Соответствующий оператор имеет формат:

```
имя_указателя=fopen(имя_файла, режим_открытия);
```

Параметры функции `fopen()` являются строками, которые могут быть как константами, так и указателями на символьные массивы. Например:

```
fp=fopen("test.dat","r");
```

Здесь `test.dat` — это имя физического файла в текущем каталоге диска, с которым теперь будет связан поток с указателем `fp`. Параметр режима `r` означает, что файл открыт для чтения. Что касается терминологии, то допустимо употреблять как выражение «открытие потока», так и выражение «открытие файла».

Существуют следующие режимы открытия потока и соответствующие им параметры:

Параметр	*Режим*
`r`	открыть для чтения
`w`	создать для записи
`a`	открыть для добавления
`r+`	открыть для чтения и записи
`w+`	создать для чтения и записи
`a+`	открыть для добавления или создать для чтения и записи

Как уже отмечалось при изучении Паскаля, надо хорошо понимать, что открытие уже существующего файла для записи ведет к

потере прежней информации в нем. Если такой файл еще не существовал, то он создастся. Открывать для чтения можно только существующий файл.

Поток может быть открыт либо для текстового, либо для двоичного (бинарного) режима обмена.

Понятие текстового файла обсуждалось в разделе 3.19. Смысл понятия остается прежним: это последовательность символов, которая делится на строки специальными кодами — возврат каретки (код 13) и перевод строки (код 10). Если файл открыт в текстовом режиме, то при чтении из такого файла комбинация символов «возврат каретки — перевод строки» преобразуется в один символ `\n` — переход к новой строке.

При записи в файл осуществляется обратное преобразование. При работе с двоичным файлом никаких преобразований символов не происходит, т.е. информация переносится без всяких изменений.

Указанные выше параметры режимов открывают текстовые файлы. Если требуется указать на двоичный файл, то к параметру добавляется буква `b`. Например: `rb`, или `wb`, или `r+b`. В некоторых компиляторах текстовый режим обмена обозначается буквой `t`, т.е. записывается `a+t` или `rt`.

Если при открытии потока по какой-либо причине возникла ошибка, то функция `fopen()` возвращает значение константы `NULL`. Эта константа также определена в файле `stdio.h`. Ошибка может возникнуть из-за отсутствия открываемого файла на диске, нехватки места в динамической памяти и т.п. Поэтому желательно контролировать правильность прохождения процедуры открытия файла. Рекомендуется следующий способ открытия:

```
FILE *fp;
if (fp=fopen("test.dat","r")==NULL
{puts("Не могу открыть файл\n");
 return;
}
```

В случае ошибки программа завершит выполнение с закрытием всех ранее открытых файлов.

Закрытие потока (файла) осуществляет функция `fclose()`, прототип которой имеет вид:

```
int fclose(FILE *fptr);
```

Здесь `fptr` обозначает формальное имя указателя на закрываемый поток. Функция возвращает ноль, если операция закрытия прошла успешно. Другая величина означает ошибку.

Запись и чтение символов. Запись символов в поток производится функцией `putc()` с прототипом

```
int putc(int ch, FILE *fptr);
```

Если операция прошла успешно, то возвращается записанный символ. В случае ошибки возвращается константа EOF.

Считывание символа из потока, открытого для чтения, производится функцией gets() с прототипом

```
int gets(FILE *fptr);
```

Функция возвращает значение считываемого из файла символа. Если достигнут конец файла, то возвращается значение EOF. Заметим, что это происходит лишь в результате чтения кода EOF.

Исторически сложилось так, что gets() возвращает значение типа int. То же можно сказать и про аргумент ch в описании функции puts(). Используется же в обоих случаях только младший байт. Поэтому обмен при обращении может происходить и с переменными типа char.

Пример 1. Составим программу записи в файл символьной последовательности, вводимой с клавиатуры. Пусть признаком завершения ввода будет символ *.

```
//Запись символов в файл
#include <stdio.h>
void main()
{ FILE *fp;
  char c;
  if((fp=fopen("test.dat","w"))==NULL)
  {puts("Не могу открыть файл!\n"); return;}
  puts("Вводите символы. Признак конца - *");
  while((c=getchar())!='*')
  putc(c,fp);
  fclose(fp);
}
```

В результате на диске (в каталоге, определяемом системой) будет создан файл с именем test.dat, который заполнится вводимыми символами. Символ * в файл не запишется.

Пример 2. Файл, созданный в результате работы предыдущей программы, требуется последовательно прочитать и содержимое вывести на экран.

```
//Чтение символов из файла
#include <stdio.h>
#include <conio.h>
void main()
{ FILE *fp;
  char c;
  clrscr();
  if((fp=fopen("test.dat","r"))==NULL)
  {puts("Не удалось открыть файл!\n"); return;}
```

```
  while((c=getc(fp))!=EOF) putchar(c);
  fclose(fp);
}
```

Связь между результатом работы предыдущей программы и данной программой осуществляется через имя физического файла, которое в обоих случаях должно быть одним и тем же.

Запись и чтение целых чисел. Запись целых чисел в поток без преобразования их в символьную форму производится функцией `putw()` с прототипом

```
int putw(int, FILE *fptr);
```

Если операция прошла успешно, то возвращается записанное число. В случае ошибки возвращается константа `EOF`.

Считывание целого числа из потока, открытого для чтения, производится функцией `getw()` с прототипом

```
int getw(FILE *fptr);
```

Функция возвращает значение считываемого из файла числа. Если прочитан конец файла, то возвращается значение `EOF`.

Пример 3. Составим программу, по которой в файл запишется последовательность целых чисел, вводимых с клавиатуры, а затем эта последовательность будет прочитана и выведена на экран. Пусть признаком конца ввода будет число 9999.

```
//Запись и чтение целых чисел
#include <stdio.h>
#include <conio.h>
void main()
{FILE *fp;
  int x;
  clrscr();
  //Открытие потока для записи
  if((fp=fopen("test.dat","w"))==NULL)
  {puts("Ошибка открытия файла для записи!\n");
   return;
  }
  puts("Вводите числа. Признак конца - 9999");
  scanf("%d",&x);
  while(x!=9999)
  {putw(x,fp);scanf("%d",&x);
  }
  fclose(fp);//Закрытие потока в режиме записи
  //Открытие потока для чтения
  if((fp=fopen("test.dat","r"))==NULL)
  {puts("Ошибка открытия файла для чтения!\n");
   return;
  }
```

```
  while((x=getw(fp))!=EOF) printf("\n%d",x);
  fclose(fp);
}
```

После завершения ввода чисел в файл его необходимо закрыть. При этом происходит сброс накопленных в буфере значений на диск. Только после этого можно открывать файл для чтения. Указатель устанавливается на начало потока, и дальнейшее чтение будет происходить от начала до конца файла.

Запись и чтение блоков данных. Специальные функции обмена с файлами имеются только для символьного и целого типов данных. В общем случае используются функции чтения и записи блоков данных. С их помощью можно записывать в файл и читать из файла вещественные числа, массивы, строки, структуры. При этом, как и для ранее рассмотренных функций, сохраняется форма внутреннего представления данных.

Функция записи блока данных имеет прототип

```
int fread(void*buf, int bytes, int n, FILE*fptr);
```

Здесь

`buf` — указатель на адрес данных, записываемых в файл;

`bytes` — длина в байтах одной единицы записи (блока данных);

`n` — число блоков, передаваемых в файл;

`fptr` — указатель на поток.

Если запись выполнилась благополучно, то функция возвращает число записанных блоков (значение `n`).

Функция чтения блока данных из файла имеет прототип

```
int fwrite(void*buf, int bytes, int n, FILE*fptr);
```

По аналогии с предыдущим описанием легко понять смысл параметров.

Пример 4. Следующая программа организует запись блоком в файл строки (символьного массива), а также чтение и вывод на экран записанной информации.

```
#include <string.h>
#include <stdio.h>
void main()
{  FILE *stream;
   char msg[]="this is a test";
   char buf[20];
   if ((stream=fopen("DUMMY.FIL", "w+"))==NULL)
   { puts("Ошибка открытия файла\n");
     return;
   }
   //Запись строки в файл
   fwrite(msg, strlen(msg)+1, 1, stream);
```

```
    //Установка указателя на начало файла
    fseek(stream, 0, SEEK_SET);
    //Чтение строки из файла
    fread(buf, strlen(msg)+1, 1, stream);
    printf("%s\n", buf);
    fclose(stream);
}
```

В этой программе поток открывается в режиме `w+` (создание для записи с последующим чтением). Поэтому закрывать файл после записи не потребовалось. Новым элементом данной программы по сравнению с предыдущими является использование функции установки указателя потока в заданную позицию. Ее формат:

```
int fseek(указатель_на поток, смещение,
          начало_отсчета);
```

Начало отсчета задается одной из констант, определенных в файле `stdio.h`:

`SEEK_SET` (имеет значение 0) — начало файла;
`SEEK_CUR` (имеет значение 1) — текущая позиция;
`SEEK_END` (имеет значение 2) — конец файла.

Смещение определяет число байт, на которое надо сместить указатель относительно заданного начала отсчета. Смещение может быть как положительным, так и отрицательным числом. Оба параметра имеют тип `long`.

Форматный обмен с файлами. С помощью функции форматного вывода можно формировать на диске текстовый файл с результатами вычислений, представленными в символьном виде. В дальнейшем этот файл может быть просмотрен на экране, распечатан на принтере, отредактирован с помощью текстового редактора. Общий вид функции форматного вывода:

```
int fprintf(указатель_на_поток, форматная_строка,
            список_переменных);
```

Использовавшаяся нами ранее функция `printf()` для организации вывода на экран является частным вариантом функции `fprintf()`. Функция `printf()` работает лишь со стандартным потоком `stdin`, который всегда связывается системой с дисплеем. Не будет ошибкой, если в программе вместо `printf()` написать `fprintf(stdin,...)`.

Правила использования спецификаторов форматов при записи в файлы на диске точно такие же, как и при выводе на экран (см. разд. 4.4).

Пример 5. Составим программу, по которой будет рассчитана и записана в файл таблица квадратных корней для целых чисел от 1 до 10. Для контроля эта же таблица выводится на экран.

```
//Таблица квадратных корней
#include <stdio.h>
#include <iostream.h>
#include <math.h>
void main()
{ FILE *fp;
  int x;
  fp=fopen("test.dat","w");
  //Вывод на экран и в файл шапки таблицы
  printf("\t Таблица квадратных корней\n");
  fprintf(fp,"\t Таблица квадратных корней\n");
  printf("\t x\t\tsqrt(x)\n");
  fprintf(fp,"\t x\t\tsqrt(x)\n");
  //Вычисление и вывод таблицы квадратных корней
  //на экран и в файл
  for(x=1; x<=10; x++)
  { printf("\t%f\t%f\n",float(x),sqrt(float(x)));
    fprintf(fp,"\t%f\t%f\n",float(x),
    sqrt(float(x)));
   }
   fclose(fp);
   }
```

Если теперь с помощью текстового редактора (например, входящего в систему программирования) открыть файл `test.dat`, то на экране увидим:

```
Таблица квадратных корней
    x              sqrt(x)
    1.000000       1.000000
    2.000000       1.414214
    3.000000       1.732051
    4.000000       2.000000
    5.000000       2.236068
    6.000000       2.449490
    7.000000       2.645751
    8.000000       2.828427
    9.000000       3.000000
    10.000000      3.162278
```

Теперь эти результаты можно распечатать, включить в текст отчета и т.п.

Форматный ввод из текстового файла осуществляется с помощью функции `fscanf()`, общий формат которой выглядит следующим образом:

```
int fscanf(указатель_на_поток, форматная_строка,
           список_адресов_переменных);
```

Данной функцией удобно пользоваться в тех случаях, когда исходные данные заранее подготавливаются в текстовом файле.

В следующем примере числовые данные из файла `test.dat`, полученного в результате выполнения предыдущей программы, вводятся в числовые массивы `X` и `Y`. Для контроля значения элементов массивов выводятся на экран. Предварительно с помощью текстового редактора в файле `test.dat` удаляются две первые строки с заголовками. В результате в файле останутся только числа.

Пример 6.

```
//Ввод чисел из файла
#include <stdio.h>
#include <iostream.h>
#include <math.h>
void main()
{ FILE *fp;
  int i;
  float x[10], y[10];
  fp=fopen("test.dat","r");
  for(i=0; i<10; i++)
  { fscanf(fp,"%f%f",&x[i],&y[i]);
   printf("%f  %f\n",x[i],y[i]);
  }
   fclose(fp);
}
```

Упражнения

1. Составить программу, которая формирует файл целых чисел, получаемых с помощью датчика случайных чисел.

2. Составить программу, которая в файле, сформированном программой из предыдущей задачи, находит наибольшее и наименьшее значения.

3. Составить программу, которая формирует файл из строчных латинских букв, выбираемых случайным образом.

4. Составить программу, которая в файле, сформированном программой из предыдущей задачи, подсчитает количество букв *z*.

5. Составить программу, записывающую на диск таблицу Менделеева.

6. Составить программу, которая в файле, сформированном в результате решения предыдущей задачи, будет отыскивать сведения о заданном химическом элементе.

7. Сведения о деталях, хранящихся на складе, содержат следующие атрибуты: название, количество, стоимость одной детали. Составить программы, решающие следующие задачи:

а) заполнить файл с информацией о деталях на складе;

б) вычислить общую стоимость деталей;

в) выяснить, какие детали имеются в наибольшем количестве, какие — в наименьшем;

г) вывести информацию о наличии на складе деталей данного типа и их количестве;

д) внести изменения в файл после выдачи со склада определенного количества данного вида деталей. Если какой-то тип деталей полностью выбран со склада, то уничтожить запись о ней в файле.

4.14. Объектно-ориентированное программирование в Си++

Основным отличием языка Си++ от Си является наличие в нем средств объектно-ориентированного программирования (ООП). Часто в литературе язык Си++ определяют именно как язык объектно-ориентированного программирования. Ранее в разд. 3.23 мы уже обсуждали основные понятия и приемы ООП на примере Турбо Паскаля. Для Си++ базовые понятия ООП, естественно, остаются теми же: это *инкапсуляция, наследование* и *полиморфизм*. Реализация ООП на Си++ несколько более гибкая, чем в Турбо Паскале. Существуют определенные терминологические отличия. Первое такое отличие заключается в следующем: вместо понятия «объектный тип данных», применяемого в Турбо Паскале, в Си++ используется понятие «класс».

Класс — это структурированный тип, включающий в себя в качестве элементов типизированные данные и функции, применяемые по отношению к этим данным. Таким образом, инкапсуляция (объединение параметров и методов) заложена в составе элементов класса: типизированные данные — это параметры, а методы реализованы через функции.

Тип «класс» устанавливается для *объектов*. Принято говорить: однотипные объекты принадлежат одному классу.

Синтаксис объявления класса подобен синтаксису объявления структуры. Объявление начинается с ключевого слова `class`, за которым следует имя класса. В простейшем случае объявление класса имеет следующий формат:

```
class имя
{   тип1 переменная1
    тип2 переменная2
    . . . . . .
    public:
    функция1;
    функция2;
    . . . . . .
};
```

Основное отличие класса от структур состоит в том, что все члены класса по умолчанию считаются закрытыми и доступ к ним

могут получить только функции — члены этого же класса. Однако *режим доступа* к элементам класса может быть изменен путем его явного указания. Для этого перед элементами класса записывается соответствующий спецификатор доступа. Существуют три таких спецификатора:

- `private` (частный);
- `public` (общедоступный);
- `protected` (защищенный).

Режим доступа `private` обозначает, что соответствующий элемент может использоваться только функциями данного класса. Этот режим доступа устанавливается по умолчанию. Элементы с режимом доступа `public` доступны в других частях программы. О режиме `protected` будет сказано немного позже. Чаще всего режим доступа к данным (переменным) бывает `private`, а к функциям — `public`. Это отражено в приведенном выше формате объявления класса.

В качестве примера рассмотрим объявление класса обыкновенных дробей (см. пример 3 в разд. 3.23) с именем `Drob`, в котором значение дроби определено через структуру двух целых чисел (числитель и знаменатель), а к методам работы с дробью отнесены ввод дроби — функция `Vvod`; вычисление наибольшего общего делителя числителя и знаменателя — функция `NOD`; сокращение дроби — функция `Sokr`; возведение дроби в целую степень — функция `Stepen` — и вывод дроби на экран — функция `Print`. Объявление соответствующего класса выглядит так:

```
class Drob{
      Frac A;
      public:
      void Vvod(void);
      int NOD(void);
      void Sokr(void);
      void Stepen(int N);
      void Print(void);
      };
```

Имеется в виду, что глобально по отношению к этому классу объявлена структура с именем `Frac`:

```
struct Frac{int P; int Q;};
```

Таким образом, пять методов реализованы пятью функциями, которые в объявлении класса представлены своими прототипами. Описания функций — членов класса производится отдельно. При этом нужно указывать, к какому классу принадлежит данная функция. Для этого используется операция принадлежности, знак которой `::`. Например:

```
void Drob::Vvod(void)
```

```
{cout<<"Числитель?"; cin>>A.P;
 cout<<"Знаменатель?"; cin>>A.Q;
}
```

В основной части программы (основной функции) класс Drob будет поставлен в соответствие определенным переменным в качестве типа. Например:

```
Drob Y;
```

После этого переменная Y воспринимается в программе как объект соответствующего класса. Для основной программы открыты только функции этого объекта. Следовательно, воздействовать на параметры объекта можно только через эти функции. Аналогично элементам структуры обращение к элементам объекта производится с помощью составного имени (через точку). Например: Y.Vvod().

Пример 1. Следующая программа на Си++ является аналогом программы на Турбо Паскале из примера 1 в разд. 3.23.

```
//В программе объявлен класс простых дробей,
//описана переменная этого класса, выполнена
//обработка переменной
#include <iostream.h>
#include <math.h>
struct Frac{int P; int Q;};
Frac F;
//Объявление класса
class Drob{
      Frac A;
      public:
      void Vvod(void);
      int NOD(void);
      void Sokr(void);
      void Stepen(int N);
      void Print(void);
      };
//Описания функций – членов класса
       void Drob::Vvod(void)
       { cout<<"Числитель?"; cin>>A.P;
         cout<<"Знаменатель?"; cin>>A.Q;
       }
       int Drob::NOD(void)
       { int M,N;
         M=abs(A.P); N=A.Q;
         while(M!=N)
         { if(M>N)
           if(M%N!=0) M=M%N; else M=N;
         else if(N%M!=0) N=N%M; else N=M;
```

```
        }
    return M;
     }
    void Drob::Sokr(void)
    { int X;
      X=NOD();
      if(A.P!=0)
      { A.P=A.P/X; A.Q=A.Q/X;}
      else A.Q=1;
    }
    void Drob::Stepen(int N)
    { int i;
     F.P=F.Q=1;
     for(i=1;i<=N;i++)
     { F.P*=A.P; F.Q*=A.Q;}
     }
    void Drob::Print(void)
    { cout<<"\n"<<A.P<<"/"<<A.Q<<"\n";}
//Основная функция
    void main(void)
    { Drob Y;  //Объявление объекта
       cout<<"Вводите дробь!"<<"\n";
       Y.Vvod();
        Y.Sokr();
       cout<<"Дробь после сокращения:"<<"\n";
        Y.Print();
        Y.Stepen(2);
       cout<<"Дробь, возведенная в
       квадрат:"<<"\n";
       cout<<F.P<<"/"<<F.Q<<"\n";
    }
```

В результате выполнения этой программы на экране получим:

```
Вводите дробь!
Числитель? 6
Знаменатель? 15
Дробь после сокращения:
3/5
Дробь, возведенная в квадрат:
9/25
```

Наследование — второе фундаментальное понятие ООП. Механизм наследования позволяет формировать иерархии классов. Класс-наследник получает свойства класса-предка. Как и в Турбо Паскале, в классе-наследнике могут быть объявлены новые дополнительные элементы. Элементы-данные должны иметь имена, отличные от имен предка. Элементы-функции могут быть новыми

относительно предка, но могут и повторять имена функций своих предков. Как и в Турбо Паскале, здесь действует принцип «снизу вверх» при обращении к функции: функция потомка перекрывает одноименную функцию своего предка.

Формат объявления класса-потомка следующий:

```
class имя_потомка: режим_доступа имя_предка
                                {новые_элементы}
```

Для того чтобы элементы-данные класса-предка были доступны функциям класса-потомка, этим элементам должен быть поставлен в соответствие режим доступа `protected` (защищенный).

Пример 2. Следующая программа на Си++ является аналогом программы на Турбо Паскале из примера 2 в разд. 3.23.

```
//В программе объявлен исходный класс
//четырехугольников и классы-наследники
//параллелограммов, ромбов и квадратов.
#include <iostream.h>
#include <math.h>
//Объявление базового класса четырехугольников
class FourAngle
    { protected:
      double x1,y1,x2,y2,x3,y3,x4,y4,
       A,B,C,D,D1,D2,
       Alpha,Beta,Gamma,Delta,
       P,S;
      public:
      void Init(void);
      void Storony(void);
      void Diagonali(void);
      void Angles(void);
      void Perimetr(void);
      void Ploshad(void);
      void PrintElements(void);
    };
//Объявление класса параллелограммов — наследника
//четырехугольников
class Parall: public FourAngle
    {public:
      void Storony(void);
      void Perimetr(void);
      void Ploshad(void);
    };
//Объявление класса ромбов — наследника
//параллелограммов
class Romb: public Parall
    {public:
```

```
      void Storony(void);
      void Perimetr(void);
    };
//Объявление класса квадратов – наследника ромбов
class Kvadrat: public Romb
    { public:
      void Angles(void);
      void Ploshad(void);
    };
//Описания функции – членов класса
    void FourAngle::Init(void)
    { cout<<"\n,Введите координаты вершин: \n";
      cin>>x1>>y1>>x2>>y2>>x3>>y3>>x4>>y4;
    }
    void FourAngle::Storony(void)
    { A=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
      B=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
      C=sqrt((x4-x3)*(x4-x3)+(y4-y3)*(y4-y3));
      D=sqrt((x4-x1)*(x4-x1)+(y4-y1)*(y4-y1));
    }
     void FourAngle::Diagonali(void)
    { D1=sqrt((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3));
      D2=sqrt((x2-x4)*(x2-x4)+(y2-y4)*(y2-y4));
    }
//Функция Ugol не является членом какого-либо
//класса.Она выполняет вспомогательную роль для
//функции Angles. Эта функция может независимым
//образом использоваться и в основной программе для
//определения углов треугольника, заданного длинами
//сторон
     double Ugol(double Aa, double Bb, double Cc)
     { double VspCos, VspSin, Pi;
       Pi=4*atan(1.0);
       VspCos=(Aa*Aa+Bb*Bb-Cc*Cc)/2/Aa/Bb;
       VspSin=sqrt(1-VspCos*VspCos);
       if(abs(VspCos)>1e-7)
       return (atan(VspSin/VspCos)+Pi*(VspCos<0))/
        Pi/180;
       else return 90.0;
     }
     void FourAngle::Angles(void)
     { Alpha=Ugol(D,A,D2); Beta=Ugol(A,B,D1);
       Gamma=Ugol(B,C,D2); Delta=Ugol(C,D,D1);
    }
     void FourAngle::Perimetr(void)
     { P=A+B+C+D; }
```

```
void FourAngle::Ploshad(void)
{ double Per1, Per2;
  Per1=(A+D+D2)/2; Per2=(B+C+D1)/2;
  S=sqrt(Per1*(Per1-A)*(Per1-D)*(Per1-D2))+
  sqrt(Per2*(Per2-B)*(Per2-C)*(Per2-D1));
}
void FourAngle::PrintElements(void)
{ cout<<"Стороны:\n"<<A<<" "<<B<<" "<<C<<" "
  <<D<<"\n";
  cout<<"Углы:\n"
  <<Alpha<<" "<<Beta<<" "<<Gamma<<" "
  <<Delta<<"\n";
  cout<<"Периметр:\n"<<P<<"\n";
  cout<<"Площадь:\n"<<S<<"\n";
  cout<<"Диагонали:\n"<<D1<<" "<<D2<<"\n";
}
void Parall::Storony(void)
{ A=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
  B=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
  C=A; D=B;
}
void Parall::Perimetr(void)
{ P=2*(A+B); }
void Parall::Ploshad(void)
{ double Per;
  Per=(A+D+D2)/2;
  S=2*sqrt(Per*(Per-A)*(Per-D)*(Per-D2));
}
void Romb::Storony(void)
{ A=B=C=D=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-
 y1)); }
void Romb::Perimetr(void)
{ P=4*A; }
void Kvadrat::Angles(void)
{ Alpha=Beta=Gamma=Delta=90.0;}
void Kvadrat::Ploshad(void)
{ S=A*A; }
// Основная функция. По координатам вершин квадрата
// вычисляет и выводит все его параметры
void main(void)
{ Kvadrat obj;//Объявление объекта класса «квадрат»
  obj.Init();
  obj.Storony();
  obj.Diagonali();
  obj.Angles();
  obj.Perimetr();
```

```
  obj.Ploshad();
  obj.PrintElements();
}
```

В результате выполнения теста на экране будет получено:

```
Введите координаты вершин:
0 0 1 1 2 0 1 -1
Стороны:
1.414214 1.414214 1.414214 1.414214
Углы:
90 90 90 90
Периметр:
5.656854
Площадь:
2
Диагонали:
2 2
```

Конструкторы и деструкторы. Смысл этих понятий аналогичен их смыслу в Турбо Паскале. Основное назначение конструктора — инициализация элементов-данных объекта и выделение динамической памяти под данные. Конструктор срабатывает при выполнении оператора определения типа «класс для объекта». Деструктор освобождает выделенную конструктором память при удалении объекта.

Области памяти, занятые данными базовых типов, таких, как `int, float, double` и т.п., выделяются и освобождаются системой автоматически и не нуждаются в помощи конструктора и деструктора. Именно поэтому в программах, рассмотренных в примерах 1 и 2, конструкторы и деструкторы не объявлялись (система все равно создает их автоматически).

Конструктор и деструктор объявляются как члены-функции класса. Имя конструктора совпадает с именем класса. Имя деструктора начинается с символа ~ (тильда), за которым следует имя класса.

Пример 3. Объявляется класс для строковых объектов. В этом примере конструктор с помощью оператора `new` резервирует блок памяти для указателя `stringl`. Освобождение занятой памяти выполняет деструктор с помощью оператора `delete`.

```
class string_operation
      { char *stringl;
        int string_len;
        public:
        string_operatoin(char*)     //Конструктор
        { stringl=new char[string_len];}
        ~string_operation()         //Деструктор
        { delete stringl; }
```

```
    void input_data(char*);
    void output_data(char*);
  };
```

В основной программе явного обращения к конструктору и деструктору не требуется. Они выполняются автоматически.

Полиморфизм допускает использование функций с одним и тем же именем (а также операций) применительно к разным наборам аргументов и операндов, а также к разным их типам, в зависимости от контекста программы. В Си++ полиморфизм реализован через *механизм перегрузки.*

Внутри класса допускается существование нескольких функций с одинаковым именем, но различающимися типами результатов и наборами формальных параметров. При обслуживании обращения к такой функции компилятор выбирает подходящий вариант в зависимости от количества и типов аргументов.

Пример 4. В следующей программе определяется перегруженная функция `modul()` класса `absolute`, которая возвращает абсолютное значение как целочисленного, так и вещественного аргумента. В первом случае для этого используется библиотечная функция `abs()`, принимающая аргумент типа `int`, во втором случае — `fabs()`, принимающая аргумент типа `double`.

```
#include <iostream.h>
#include <math.h>
class absolute
      { public:
          int modul(int);
          double modul(double);
      };
int absolute:: modul(int X)
{ return(abs(X)); }
double absolute:: modul(double X)
{ return(fabs(X)); }
void main()
{ absolute number;
  cout<<"Абсолютное значение числа –765 равно:"
      <<number.modul(-765)<<endl;
  cout<<"Абсолютное значение числа –23.987 равно:"
      <<number.modul(-23.987)<<endl;
}
```

В результате работы программы получим:

```
Абсолютное значение числа –765 равно 765
Абсолютное значение числа –23.987 равно 23.987
```

Перегрузка операций. Полиморфизм в Си++ реализуется не только через механизм перегрузки функций, но и через перегрузку операций. Применительно к объектам определенного класса могут быть определены специфические правила выполнения некоторой операции. При этом сохраняется возможность ее традиционного применения в другом контексте.

Для перегрузки операции применительно к классу в число членов класса должна быть включена специальная *функция операции*, которая определяет действия, выполняемые по этой операции. Формат определения функции-операции:

```
тип_возвращаемого_значения operator знак_операции
(спецификации_параметров_операции) {тело_функции-
операции}
```

Рассмотрим пример программы, в которой используются перегруженные операции.

Пример 5. Класс `vector` определяет трехмерный вектор в евклидовом пространстве. В этом классе будут использоваться перегруженные операции сложения (+) и присваивания (=) как операции с трехмерными векторами. Сумма двух векторов вычисляется как вектор, компоненты которого равны суммам соответствующих компонент слагаемых. Операция = выполняет покомпонентное присваивание векторов.

```
#include <iostream.h>
class vector
{ int x,y,z; //Компоненты вектора
  public:
  vector operator+(vector t);
  vector operator=(vector t);
       void show(void);
       void assign(int mx, int my, int mz);
}
//Перегрузка операции +
vector vector::operator+(vector t)
{vector temp;
   temp.x=x+t.x;
   temp.y=y+t.y;
   temp.z=z+t.z;
}
//Перегрузка операции=
vector vector::operator=(vector t)
{  x=t.x; y=t.y; z=t.z;
   return *this; //Использован this (см. ниже)
}
void vector::show(void)
{cout<<x<<", ";
```

```
   cout<<y<<", ";
   cout<<x<<\n';
}
void vector::assign(int mx, int my, int mz)
{ x=mx; y=my; z=mz;}
//Основная программа
void main(void)
{ vector a, b, c;
   a.assign(1, 2, 3);
   b.assign(10, 10, 10);
   a.show();
   b.show();
   c=a+b;//Работают перегруженные операции +, =
   c.show();
   c=a+b+c;
   c.show;
   c=b=a;
   c.show;
   b.show;
}
```

В результате выполнения программы на экране получим:

```
1, 2, 3
10, 10, 10
11, 12, 13
22, 24, 26
1, 2, 3
1, 2, 3
```

Здесь естествен вопрос: почему бинарные операции-функции + и = имеют в описании только по одному аргументу? Дело в том, что другой аргумент всегда передается неявно с использованием this-указателя. Оператор `temp.x=x+t.x;` аналогичен строке `temp.x=this->x+t.x`, т.е. x ссылается на `this->x`. Здесь this ассоциируется с объектом, предшествующим знаку операции. Объект справа от знака операции передается как параметр функции.

Если аналогичным образом определять унарные операции как функции-члены класса, то для них не требуется указания параметра. Объект, к которому относится операция, передается в функцию неявно через указатель this. Например, для класса vector можно добавить объявление унарной постфиксной операции ++:

```
vector operator++(void);
```

Описание этой функции будет следующим:

```
vector vector::operator++(void)
{ x++;
```

```
        y++;
        x++;
        return *this;
}
```

Упражнения

1. Определить класс `Line`, содержащий в качестве полей данных координаты начала и конца линии, а также содержащий методы для чтения и установки координат.

2. Определить класс `DAY`, содержащий в себе перечислимый тип, определяющий день недели (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday), а также методы для установки, чтения и показа на экране информации о том, какой день недели находится в данном объекте. При вызове из объекта метода показа на экран должны выводиться название дня недели и справка о том, рабочий он или выходной.

3. Опишите любой базовый и производный от него классы так, чтобы при срабатывании в них конструкторов и деструкторов на экран выдавались соответствующие надписи типа «Сработал такой-то метод из такого-то класса».

4. Определите класс `Integer`, хранящий в поле данных целое число. Перегрузите одноместную операцию ! для этого класса, выводящую в зависимости от знака (+ или —) этого числа результат 0 (отрицательный) или 1 (положительный). Перегрузите также двухместную операцию + так, чтобы при сложении объекта с целым числом значения поля данных увеличивалось на это число, а при сложении объекта с другим объектом этого же класса поле данных уменьшалось.

5. Используя средства ООП в Си++, выполните упражнения, приведенные в конце разд. 3.23.

4.15. Форматированный ввод-вывод в Си++

Для организации ввода-вывода в Си++ можно использовать средства языка Си (`conio.h`). Однако в Си++ существует стандартная библиотека классов, ориентированная на организацию потокового ввода-вывода. Классы ввода-вывода образуют иерархию по принципу наследования. Базовым в этой иерархии является класс `ios` (исключение составляют лишь классы буферизированных потоков). В классе `ios` объединены базовые данные и методы для ввода-вывода. Прямыми потомками класса `ios` являются классы `istream` и `ostream`. Класс `istream` — это класс входных потоков; `ostream` — класс выходных потоков. Потомком этих двух классов является `iostream` — класс двунаправленных потоков ввода-вывода. С этим классом мы уже много раз

имели дело, подключая его к программам с помощью головного файла iostream.h.

Объект cout принадлежит к классу ostream и представляет собой поток вывода, связанный с дисплеем. Объект cin принадлежит классу istream и является потоком ввода, связанным с клавиатурой. Оба эти объекта наследуются классом iostream.

Знак << обозначает перегруженную операцию вставки символов в поток вывода cout, а >> — знак операции извлечения из потока ввода cin.

Для организации форматированного потокового ввода-вывода в Си++ существуют два средства:

- применение функций-членов класса ios для управления флагами форматирования;
- применение функций-манипуляторов.

Управление флагами форматирования. Флаги форматирования — двоичные коды, управляющие форматом выводимых значений. В заголовочном файле iostream.h определено следующее перечисление, задающее флаги форматирования:

```
enum{
skipws     = 0x0001 отбрасывание пробелов
left       = 0x0002 выравнивание по левому краю
                    поля
right      = 0x0004 выравнивание по правому краю
                    поля
internal   = 0x0008 заполнение пустых позиций
dec        = 0x0010 выдача в десятичном виде
oct        = 0x0020 выдача в восьмеричном виде
hex        = 0x0040 выдача в шестнадцатеричном виде
showbase   = 0x0080 выдача основания сист.
                    счисления
showpoint  = 0x0100 выдача позиции точки
uppercase  = 0x0200 выдача в формате xx.xxxx Exx
showpos    = 0x0400 выдача знака у положит. числа
scientific = 0x0800 выдача в форме с плавающ.
                    точкой
fixed      = 0x1000 выдача в форме с фиксир. точкой
unibuf     = 0x2000 улучшенная выдача
stdio      = 0x4000 освобождение потока
};
```

Фактически в этом списке содержатся имена констант, определяющие флаги соответствующих назначений. Коду формата соответствуст целый тип long.

Изменить состояние флагов формата можно с помощью функции-члена класса ios, имеющей прототип

```
long setf(long flags)
```

Например, чтобы установить флаг showbase в активный режим (включить) применительно к стандартному потоку вывода cout, используется оператор

```
cout.setf(ios::showbase);
```

Для установки флагов можно использовать побитовые операции. Например:

```
cout.setf(ios::left|ios::hex);
```

В результате включатся одновременно флаги, управляющие выравниванием по левому краю и выводом целых значений в шестнадцатеричной системе.

Для выключения используется функция

```
unsetf(long flags);
```

Например, для отмены вывода основания системы счисления используется оператор:

```
cout.unsetf(ios::showbase);
```

Вот еще некоторые функции-члены класса ios:

`long flags(void)` — возвращает текущее состояние флагов;

`int width(int len)` — возвращает текущую ширину поля вывода и устанавливает значение ширины, равное `len`;

`char fill(char ch)` — возвращает текущий символ заполнения и устанавливает новый символ заполнения `ch`;

`int precision(int num)` — возвращает текущее число десятичных знаков после точки и устанавливает значение этого параметра равным `num`.

Пример 1. Следующая программа иллюстрирует применение рассмотренного способа управления форматным выводом.

```
#include <iostream.h>
void main(void)
{
long fl;
fl=cout.flags();
cout<<"Исходное состояние флагов:"<<fl<<"\n";
//Выведется целое число – код флагов,
//установленных по умолчанию
cout.set(ios::shoupos);
cout.set(ios::scientific);
cout<<123<<" "<<1.2345678<<"\n";
//Выведется:
//+123 +1.23456e+00
cout.set(ios::hex|ios::showbase);
cout.unsetf(ios::showpos);
```

```
cout.width(15);
cout.precision(10);
cout<<123<<" "<<123.456<<" "1.2345678<<"\n";
//Выведется:
//0x7B 1.23456e+02 1.2345678e+00
cout<<"Новое состояние флагов:"<<cout.flags()
<<"\n";
//Выведется:
//Новое состояние флагов:0x28C1
cout.flags(fl);//Возврат к исходному состоянию
cout<<"После восстановления исходных флагов:\n";
cout<<123<<" "<<123.456<<1.2345678<<"\n";
//Выведется:
//После восстановления исходных флагов:
//123 123.456 1.234567
}
```

Использование манипуляторов. Для управления форматами потокового вывода можно использовать специальные функции, называемые манипуляторами. Доступ в программе к стандартным манипуляторам можно получить, подключив файл `iomanip.h`. Список стандартных манипуляторов:

`dec`	Десятичный формат
`endl`	Вывод `"\n"` и освобождение буфера
`ends`	Вывод `NULL`
`flush`	Освободить поток
`hex`	Шестнадцатеричный формат числа
`resetiosflags(long f)`	Отключить флаги, определенные `f`
`setbase(int base)`	Установить основание системы счисления
`setfill(char ch)`	Установить символ заполнения
`setiosflags(long f)`	Включить флаги, указанные `f`
`setprecision(int p)`	Установить `p` цифр в дробной части
`setw(int w)`	Установить ширину поля выдачи
`ws`	Режим пропуска символов пробела

Пример 2. В следующей программе вычисляется и выводится на экран таблица значений функций $\sin x$ и $\cos x$ на n шагах в интервале от 0 до p. Для форматирования таблицы результатов используются манипуляторы.

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
void main()
{ double a,b,x;
  int n=20;
  a=0; b=4*atan(1.);
  cout<<"  x            sin(x)         cos(x)"<<endl;
  cout<<"- - - - - - - - - - - - - - - - - - - - - -
- "<<endl;
  for(x=a;x<=b;x=x+(b-a)/n)
   cout<<setprecision(4)<<setw(10)<<x<<"       "
       <<setprecision(4)<<setw(10)<<sin(x)<<"      "
       <<setprecision(4)<<setw(10)<<cos(x)<<endl;
}
```

Начальная часть таблицы, выводимой по этой программе, имеет вид:

```
      x              sin(x)              cos(x)
 - - - - - - - - - - - - - - - - - - - - - - -
       0                   0                   1
  0.1571              0.1564              0.9877
  0.3142               0.309              0.9511
  0.4712               0.454               0.891
  0.6283              0.5878               0.809
  0.7854              0.7071              0.7071
   .   .   .   .   .   .   .   .   .   .   .   .
```

Под каждое число выделяется по 10 позиций на экране. По умолчанию число занимает крайнюю правую позицию в отведенном под него поле. Оставшиеся слева позиции занимает символ-заполнитель. По умолчанию символом-заполнителем является пробел. Однако с помощью манипулятора `setfill()` его можно заменить. Если в крайних правых позициях оказываются нули, то они не выводятся. Действие манипулятора распространяется только на значение, непосредственно следующее за ним в потоке вывода.

ГЛАВА 5. МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ

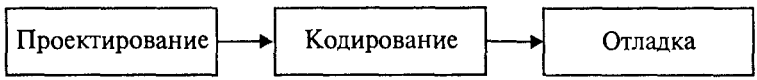
5.1. Основные понятия структурного программирования

Прошло уже более полувека со времени появления первой ЭВМ. Все это время вычислительная техника бурно развивалась. Менялась элементная база ЭВМ, росли быстродействие, объем памяти, менялись средства взаимодействия человека с машиной. Безусловно, эти изменения сказывались самым непосредственным образом на работе программиста. Определенный общепринятый способ производства чего-либо (в данном случае — программ) называют технологией. Далее мы будем говорить о *технологии программирования*.

На первых ЭВМ с «тесной» памятью и небольшим быстродействием основным показателем качества программы была ее экономичность по занимаемой памяти и времени счета. Чем программа получалась короче, тем класс программиста считался выше. Такое сокращение программы часто давалось большими усилиями. Иногда программа получалась настолько «хитрой», что могла «перехитрить» самого автора. Возвращаясь через некоторое время к собственной программе, желая что-то изменить, программист мог запутаться в ней, забыв свою «гениальную идею».

Так как вероятность выхода из строя сложного технического устройства больше, чем простого, очень сложный алгоритм всегда увеличивает вероятность ошибки в программе.

В процессе изготовления программного продукта программист должен пройти определенные этапы.



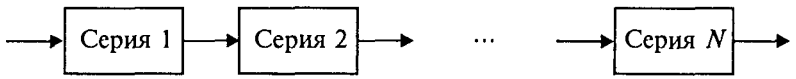
На стадии проектирования строится алгоритм будущей программы, например, в виде блок-схемы. Кодирование — это составление текста программы на языке программирования. Отладка осуществляется с помощью тестов, т.е. программа выполняется с некоторым заранее продуманным набором исходных данных, для которого известен результат. Чем сложнее программа, тем большее число тестов требуется для ее проверки. Очень «хитрую» программу трудно протестировать исчерпывающим образом. Всегда есть шанс, что какой-то «подводный камень» остался незамеченным.

С ростом памяти и быстродействия ЭВМ, с совершенствованием языков программирования и трансляторов с этих языков проблема экономичности программы становится менее острой. Все более важной качественной характеристикой программ становится их простота, наглядность, надежность. С появлением машин третьего поколения эти качества стали основными.

В конце 60-х — начале 70-х гг. XX столетия вырабатывается дисциплина, которая получила название структурного программирования. Ее появление и развитие связаны с именами Э. В. Дейкстры, Х. Д. Милса, Д. Е. Кнута и других ученых. Структурное программирование до настоящего времени остается основой технологии программирования. Соблюдение его принципов позволяет программисту быстро научиться писать ясные, безошибочные, надежные программы.

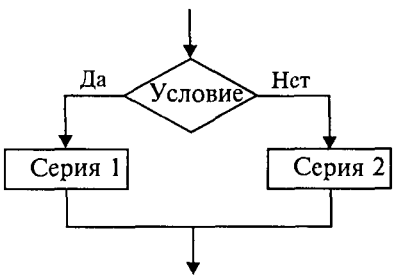
В основе структурного программирования лежит теорема, которая была строго доказана в теории программирования. Суть ее в том, что *алгоритм для решения любой логической задачи можно составить только из структур «следование, ветвление, цикл»*. Их называют базовыми алгоритмическими структурами. Из предыдущих разделов учебника вы уже знакомы с этими структурами. По сути дела, мы и раньше во всех рассматриваемых примерах программ придерживались принципов структурного программирования.

Следование — это линейная последовательность действий:



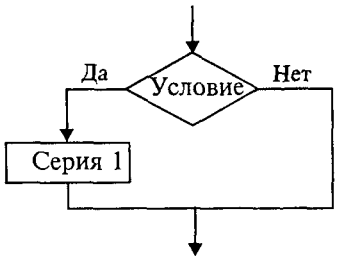
Каждый блок может содержать в себе как простую команду, так и сложную структуру, но обязательно должен иметь один вход и один выход.

Ветвление — алгоритмическая альтернатива. Управление передается одному из двух блоков в зависимости от истинности или ложности условия. Затем происходит выход на общее продолжение:



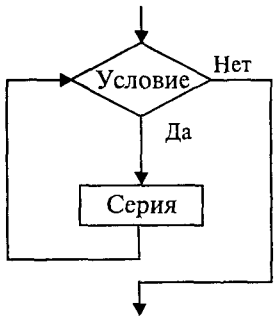

```
если условие
то   серия 1
иначе серия 2
кв
```

Неполная форма ветвления имеет место, когда на ветви «нет» пусто:




```
если условие
то   серия 1
иначе
кв
```

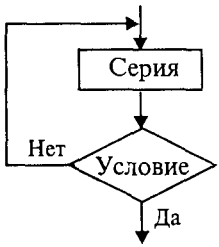
Цикл — повторение некоторой группы действий по условию. Различаются два типа цикла. Первый — цикл с предусловием (цикл-пока):




```
пока условие
повторять
нц
  серия
кц
```

Пока условие истинно, выполняется серия, образующая тело цикла.

Второй тип циклической структуры — цикл с постусловием (цикл-до):




```
повторять
      серия
до условие
```

Здесь тело цикла предшествует условию цикла. Тело цикла повторяет свое выполнение, если условие ложно. Повторение кончается, когда условие станет истинным.

Теоретически необходимым и достаточным является лишь первый тип цикла — цикл с предусловием. Любой циклический алго-

ритм можно построить с его помощью. Это более общий вариант цикла, чем цикл-до. В самом деле, тело цикла-до хотя бы один раз обязательно выполнится, так как проверка условия происходит после завершения его выполнения. А для цикла-пока возможен такой вариант, когда тело цикла не выполнится ни разу. Поэтому в любом языке программирования можно было бы ограничиться только циклом-пока. Однако в ряде случаев применение цикла-до оказывается более удобным, и поэтому он используется.

Иногда в литературе структурное программирование называют программированием без GOTO. Действительно, при таком подходе нет места безусловному переходу. Неоправданное использование в программах оператора GOTO лишает ее структурности, а значит, всех связанных с этим положительных свойств: прозрачности и надежности алгоритма. Хотя во всех процедурных языках программирования этот оператор присутствует, однако, придерживаясь структурного подхода, его употребления следует избегать.

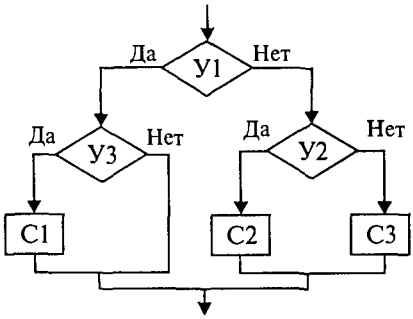
Сложный алгоритм состоит из соединенных между собой базовых структур. Соединяться эти структуры могут двумя способами: *последовательным* и *вложенным*. Эта ситуация аналогична тому, что мы наблюдаем в электротехнике, где любая сколь угодно сложная электрическая цепь может быть разложена на последовательно и параллельно соединенные участки.

Вложенные алгоритмические структуры не являются аналогом параллельно соединенных проводников. Здесь больше подходит аналогия с матрешками, помещенными друг в друга. Если блок, составляющий тело цикла, сам является циклической структурой, то, значит, имеют место вложенные циклы. В свою очередь, внутренний цикл может иметь внутри себя еще один цикл и т.д. В связи с этим вводится представление о *глубине вложенности* циклов. Точно так же и ветвления могут быть вложенными друг в друга.

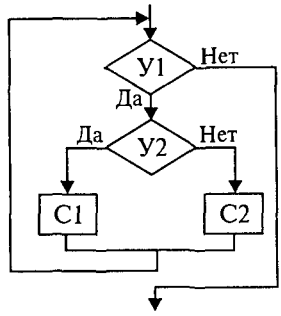
Структурный подход требует соблюдения стандарта в изображении блок-схем алгоритмов. Чертить их нужно так, как это делалось во всех приведенных примерах. Каждая базовая структура должна иметь один вход и один выход. Нестандартно изображенная блок-схема плохо читается, теряется наглядность алгоритма. Вот несколько примеров структурных блок-схем алгоритмов (рис. 47).

Такие блок-схемы легко читаются. Их структура хорошо воспринимается зрительно. Структуре каждого алгоритма можно дать название. У приведенных на рис. 47 блок-схем следующие названия:

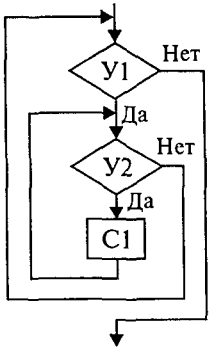
1. Вложенные ветвления. Глубина вложенности равна единице.
2. Цикл с вложенным ветвлением.
3. Вложенные циклы-пока. Глубина вложенности — единица.
4. Ветвление с вложенной последовательностью ветвлений на положительной ветви и с вложенным циклом-пока на отрицательной ветви.



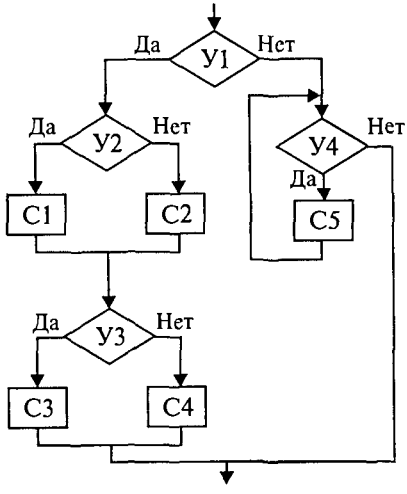
1



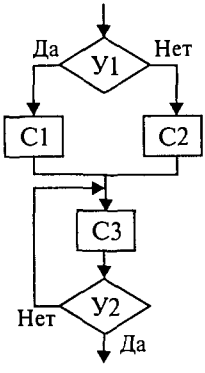
2



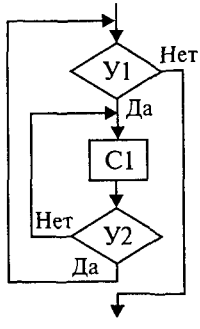
3



4



5



6

Рис. 47

5. Следование ветвления и цикла-до.

6. Вложенные циклы. Внешний — цикл-пока, внутренний — цикл-до.

Наряду с блок-схемами для описания алгоритмов часто используются псевдокоды. Учебный алгоритмический язык школьной информатики является примером такого псевдокода. Учебный АЯ — структурный псевдокод. В нем вообще отсутствует безусловный переход. Обучение составлению алгоритмов на этом языке способствует «структурному воспитанию» программиста.

Языки программирования Паскаль и Си называют языками структурного программирования. В них есть все необходимые управляющие конструкции для структурного построения программы. Наглядность такому построению придает структуризация внешнего вида текста программы. Основной используемый для этого прием — сдвиги строк, которые должны подчиняться следующим правилам:

- конструкции одного уровня вложенности записываются на одном вертикальном уровне (начинаются с одной позиции в строке);
- вложенная конструкция записывается смещенной по строке на несколько позиций вправо относительно внешней для нее конструкции.

Для приведенных выше блок-схем структура текста программы на Паскале должна быть следующей:

1.
```
if <U1>
then if <U2>
     then <C1>
else if <U2>
     then <C2>
     else <C3>
```

2.
```
while <U1> do
   if <U2>
   then <C1>
   else <C2>
```

3.
```
while <U1> do
   while <U2> do
    <C1>
```

4.
```
if <U1>
then  begin if <U2>
            then <C1>
            else <C2>;
            if<U3>
            then <C3>
            else <C4>
      end
else  while <U4> do
       <C5>
```

5.	6.

`if <U1>`
`then <C1>`
`else <C2>;`
`repeat`
`  C3`
`  until <U2>`
	`while <U1> do`
	`  repeat`
	`  <C1>`
	`  until <U2>`

Структурная методика алгоритмизации — это не только форма описания алгоритма, но это еще и *способ мышления программиста.* Создавая алгоритм, нужно стремиться составлять его из стандартных структур. Если использовать строительную аналогию, можно сказать, что структурная методика построения алгоритма подобна сборке здания из стандартных секций в отличие от складывания по кирпичику.

Еще одним важнейшим технологическим приемом структурного программирования является *декомпозиция решаемой задачи на подзадачи* — более простые с точки зрения программирования части исходной задачи. Алгоритмы решения таких подзадач называются *вспомогательными алгоритмами.* В связи с этим возможны два пути в построении алгоритма:

- «сверху вниз»: сначала строится основной алгоритм, затем вспомогательные алгоритмы;
- «снизу вверх»: сначала составляются вспомогательные алгоритмы, затем основной.

Первый подход еще называют методом *последовательной детализации*, второй — *сборочным* методом.

Сборочный метод предполагает накопление и использование библиотек вспомогательных алгоритмов, реализованных в языках программирования в виде подпрограмм, процедур, функций. При последовательной детализации сначала строится основной алгоритм, а затем в него вносятся обращения к вспомогательным алгоритмам первого уровня. После этого составляются вспомогательные алгоритмы первого уровня, в которых могут присутствовать обращения к вспомогательным алгоритмам второго уровня, и т.д. Вспомогательные алгоритмы самого нижнего уровня состоят только из простых команд.

Метод последовательной детализации применяется в любом конструировании сложных объектов. Это естественная логическая последовательность мышления конструктора: постепенное углубление в детали. В нашем случае речь идет тоже о конструировании, но только не технических устройств, а алгоритмов. Достаточно сложный алгоритм другим способом построить практически невозможно.

Методика последовательной детализации позволяет организовать работу коллектива программистов над сложным проектом. На-

пример, руководитель группы строит основной алгоритм, а разработку вспомогательных алгоритмов и написание соответствующих подпрограмм поручает своим сотрудникам. Участники группы должны лишь договориться об интерфейсе (т.е. взаимосвязи) между разрабатываемыми программными модулями, а внутренняя организация программы — личное дело программиста.

Пример разработки программы методом последовательной детализации будет рассмотрен в следующем разделе.

5.2. Метод последовательной детализации

Суть метода была описана выше. Сначала анализируется исходная задача. В ней выделяются подзадачи. Строится иерархия таких подзадач (рис. 48).

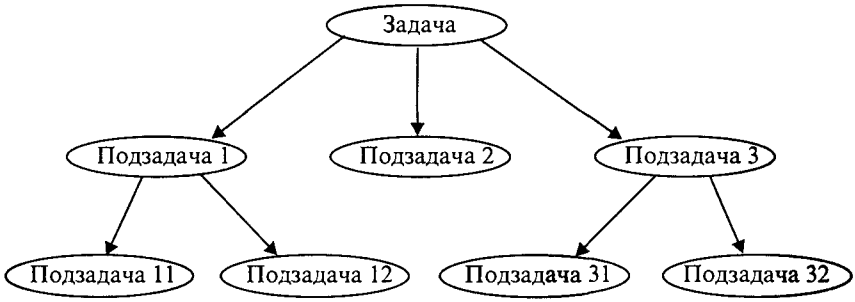


Рис. 48

Затем составляются алгоритмы (или программы), начиная с основного алгоритма (основной программы), далее — вспомогательные алгоритмы (подпрограммы) с последовательным углублением уровня, пока не получим алгоритмы, состоящие из простых команд.

Вернемся к задаче «Интерпретатор», которая рассматривалась в разд. 3.16. Напомним условие: дана исходная символьная строка, имеющая следующий вид:

```
a ⊕ b =
```

На месте `a` и `b` стоят десятичные цифры; значком ⊕ обозначен один из знаков операций: +, –, *. Нужно, чтобы машина вычислила это выражение и после знака = вывела результат. Операция деления не рассматривается для того, чтобы иметь дело только с целыми числами.

Сформулируем требования к программе *Interpretator*, которые сделают ее более универсальной, чем вариант, рассмотренный в разд. 3.16:

1. Операнды `a` и `b` могут быть многозначными целыми положительными числами в пределах `MaxInt`.

2. Между элементами строки, а также в начале и в конце могут стоять пробелы.

3. Программа осуществляет синтаксический контроль текста.

Ограничимся простейшим вариантом контроля: строка должна состоять только из цифр, знаков операций, знака = и пробелов.

4. Проводится семантический контроль: строка должна быть построена по схеме a ⊕ b =. Ошибка, если какой-то элемент отсутствует или нарушен их порядок.

5. Осуществляется контроль диапазона значений операндов и результата (не должны выходить за пределы `MaxInt`).

Уже из перечня требований становится ясно, что программа будет непростой. Составлять ее мы будем, используя метод последовательной детализации. Начнем с того, что представим в самом общем виде алгоритм как линейную последовательность этапов решения задачи:

1. Ввод строки.

2. Синтаксический контроль (нет ли недопустимых символов?).

3. Семантический контроль (правильно ли построено выражение?).

4. Выделение операндов. Проверка операндов на допустимый диапазон значений. Перевод в целые числа.

5. Выполнение операции. Проверка результата на допустимый диапазон.

6. Вывод результата.

Этапы 2, 3, 4, 5 будем рассматривать как подзадачи первого уровня, назвав их (и будущие подпрограммы) соответственно `Sintax`, `Semantika`, `Operand`, `Calc`. В свою очередь, для их реализации потребуется решение следующих подзадач: пропуск лишних пробелов (`Propusk`), преобразование символьной цифры в целое число (`Cifra`). Кроме того, при выделении операндов понадобится распознавать операнд, превышающий максимально допустимое значение (`Error`). Обобщая все сказанное в схематической форме, получаем некоторую структуру подзадач. Этой структуре будет соответствовать аналогичная структура программных модулей (рис. 49).

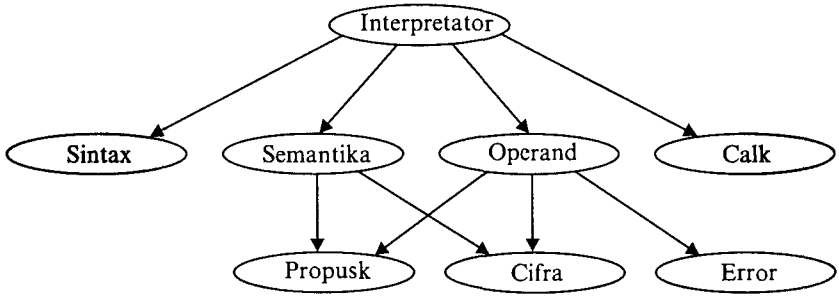


Рис. 49

Первый шаг детализации. Сначала наметим все необходимые подпрограммы, указав лишь их заголовки (спецификации). На месте тела подпрограмм запишем поясняющие комментарии (такой вид подпрограммы называется «заглушкой»). Напишем основную часть программы. А потом вернемся к детальному программированию процедур и функций. На первом этапе программирования вместо тела подпрограммы опишем ее назначение в форме комментария.

```
Program Interpretator;
Type PozInt=0..MaxInt;
Var Line: String;
      A,B: PozInt;
      Znak: Char;
      Flag: Boolean;
      Rezult: Integer;
Procedure Propusk(Stroka: String; M: Byte; Var N
                  :Byte);
Begin
{----------------------------------------
Пропускает подряд стоящие пробелы в строке Stroka,
начиная с символа номер M. В переменной N получает
номер первого символа, не являющегося пробелом
----------------------------------------}
End;
Function Cifra(C: Char): Boolean;
Begin
{----------------------------------------
Получает значение True, если символ с является
цифрой, и False - в противном случае.
----------------------------------------}
End;
Function Sintax(Stroka: String): Boolean;
Begin
{----------------------------------------
Синтаксический контроль. Получает значение True,
если в строке нет других символов, кроме цифр,
знаков операций, знака "=" и пробелов. В противном
случае - False.
----------------------------------------}
End;
Function Semantika(Stroka: String): Boolean;
Begin
{----------------------------------------
Проверяет, соответствует ли структура строки
формату "a⊕b=". Если да, то получает значение
True, если нет - False.
```

```
-----------------------------------------}
End;
Procedure Operand(Stroka: String; Var A,B: PozInt;
Var Z: Char; Var Flag: Boolean);
Begin
{-----------------------------------------
Выделяет операнды. Проверяет их на допустимый
диапазон значений. Переменная Flag получает
значение True, если результат проверки
положительный, и False - в противном случае.
Преобразует операнды в целые положительные числа A,
B. В переменной Z получает знак операции.
-----------------------------------------}
End;
Procedure Calc(A,B: PozInt; Znak: Char; Var Rez:
Integer; Var Flag: Boolean);
Begin
{-----------------------------------------
Вычисляет результат операции. Проверяет
принадлежность результата диапазону от - MaxInt до
MaxInt. Flag - признак результата проверки.
Результат операции получается в переменной Rez.
-----------------------------------------}
End;
Begin {--- Начало основной части программы ---}
WriteLn("Введите выражение:");
WriteLn;
Read(Line);
If Not Sintax(Line)
Then WriteLn("Недопустимые символы")
Else If Not Semantika(Line)
     Then WriteLn("Неверное выражение")
     Else
       Begin
         Operand(Line,A,B,Znak,Flag);
         If Not Flag
         Then WriteLn("Слишком большие операнды")
         Else
           Begin
             Calc(A,B,Znak,Rezult,Flag);
             If Not Flag
             Then WriteLn("Большой результат")
             Else WriteLn(Rezult)
           End
       End
End.
```

Второй шаг детализаци. Теперь будем составлять подпрограммы.

```
Procedure Propusk(Stroka: String; M: Byte; Var N
:Byte);
Begin
   N:=M;
   While (Stroka[N]=' ') And (N<Length(Stroka)) Do
   N:=N+1
End;
Function Cifra(C: Char): Boolean;
Begin
   Cifra:=(C>='0') And (C<='9')
End;
Function Sintax(Stroka: String): Boolean;
Var I: Byte;
Begin
   Sintax:=True;
   For I:=1 To Length(Stroka) Do
   If Not ((Stroka[I]=" ") Or Cifra(Stroka[I]) Or
   (Stroka[I]='+' Or (Stroka[I]='-') Or
   (Stroka[I]='*') Or (Stroka[I]='='))
   Then Sintax:=False
End;
Function Semantika(Stroka: String): Boolean;
Var I: Byte;
Begin
   Semantika:=True;
   Propusk(Stroka,1,I);
   If Not Cifra(Stroka[I])
   Then
   Begin
      Semantika:=False;
      Exit
   End;
   While Cifra(Stroka[I]) Do
   I:=I+1;
   Propusk(Stroka,I,I);
   If Not((Stroka[I]='+') Or (Stroka[I]='-')
   Or(Stroka[I]='*'))
   Then
   Begin
      Semantika:=False;
      Exit
   End;
   I:=I+1;
   Propusk(Stroka,I,I);
```

```
  If Not Cifra(Stroka[I])
  Then
  Begin
     Semantika:=False;
     Exit
  End;
  While Cifra(Stroka[I]) Do
  I:=I+1;
  Propusk(Stroka,I,I);
  If Stroka[I]<>'='
  Then
  Begin
     Semantika:=False;
     Exit
  End;
  I:=I+1;
  Propusk(Stroka,I,I);
  If I<Length(Stroka)
  Then
  Begin
     Semantika:=False;
     Exit
  End
End;
Procedure Operand(Stroka: String; Var A,B: PozInt;
Var Z: Char; Var Flag: Boolean);
Var P,Q: String;
  I: Byte;
  Code: Integer;
Function Error(S: String): Boolean;
{ Функция принимает значение True, если в строке S
находится число, превышающее максимальное целое, и
False – в противном случае }
Const Lim='32767';
Var I:Byte;
Begin
  If Length(S)>5
  Then Error:=True
  Else If Length(s)<5
       Then Error:=False
       Else Error:=S>Lim
End;
Begin
  Flag:=True;
  I:=1;
  Propusk(Stroka,I,I);
```

```
  Q:='';
  While Cifra(Stroka[I]) Do
  Begin
       Q:=Q+Stroka[I];
       I:=I+1
  End;
  If Error(Q)
  Then
  Begin
       Flag:=False;
       Exit
  End;
  Propusk(Stroka,I,I);
  Z:=Stroka[I];
  I:=I+1;
  Propusk(Stroka,I,I);
  P:='';
  While Cifra(Stroka[I]) Do
  Begin
       P:=P+Stroka[I];
       I:=I+1
  End;
  If Error(P)
  Then
  Begin
       Flag:=False;
       Exit
  End;
  Val(Q,A,Code);
  Val(P,B,Code)
{Стандртная процедура Val преобразует цифровую
строку в соответствующее число; Code - код ошибки}
End;
Procedure Calc(A,B:PozInt;Znak:Char;
Var Rez: Integer; Var Flag:Boolean);
Var X,Y,Z: Real;
Begin
  Flag:=True;
  Y:=A;
  Z:=B;
  Case  Znak Of
        '+':X:=Y+Z;
        '-':X:=Y-Z;
        '*':X:=Y*Z
  End;
If Abs(X)>MaxInt
```

```
    Then
    Begin
        Flag:=False;
        Exit
    End;
    Rez:=Round(X)
End;
```

Обратите внимание на то, что функция `Error` определена как внутренняя в процедуре `Operand`. Это объясняется тем, что указанная функция используется только в данной процедуре. Другим программным модулям она «не нужна».

Окончательно объединив тексты подпрограмм с основной программой, получаем рабочий вариант программы *Interpretator*. Теперь ее можно вводить в компьютер.

Отладка и тестирование программы. Никогда нельзя быть уверенным, что одним махом написанная программа будет верной (хотя такое и возможно, но с усложнением программы становится все менее вероятным). До окончательного рабочего состояния программа доводится в процессе *отладки*.

Ошибки могут быть «языковые», могут быть алгоритмические. Первый тип ошибок, как правило, помогает обнаружить компилятор с Паскаля. Это ошибки, связанные с нарушением правил языка программирования. Их еще называют ошибками *времени компиляции*, ибо обнаруживаются они именно во время компиляции. Сам компилятор в той или иной форме выдает пользователю сообщение о характере ошибки и ее месте в тексте программы. Исправив очередную ошибку, пользователь повторяет компиляцию. И так продолжается до тех пор, пока не будут ликвидированы все ошибки этого уровня.

Алгоритмические ошибки приводят к различным последствиям. Во-первых, могут возникнуть невыполнимые действия. Например, деление на нуль, корень квадратный из отрицательного числа, выход индекса за границы строки и т. п. Это ошибки *времени исполнения*. Они приводят к прерыванию выполнения программы. Как правило, имеются системные программные средства, помогающие в поиске таких ошибок.

Другая ситуация, когда алгоритмические ошибки не приводят к прерыванию выполнения программы. Программа выполняется до конца, получаются какие-то результаты, но они не являются верными. Для окончательной отладки алгоритма и анализа его правильности производится тестирование. *Тест* — это такой вариант решения задачи, для которого заранее известны результаты. Как правило, один тестовый вариант не доказывает правильность программы. Программист должен придумать систему тестов, построить план тестирования для исчерпывающего испытания всей программы.

Мы уже говорили о том, что качественная программа ни в каком варианте не должна завершаться аварийно. Тесты программы *Interpretator* должны продемонстрировать, что при правильном вводе исходной строки будут всегда получаться верные результаты, а при наличии ошибок (синтаксических, семантических, выхода за диапазон) будут получены соответствующие сообщения. План тестирования нашей программы может быть, например, таким:

№	Что вводим	Что должно получиться
1	`25+73=`	`98`
2	`5274–12315=`	`-7041`
3	`614*25=`	`15350`
4	`25,5+31,2=`	`Недопустимые символы`
5	`5=6–1`	`Неверное выражение`
6	`72315–256=`	`Слишком большие операнды`
7	`32000*100=`	`Большой результат`

Успешное прохождение всех тестов есть необходимое условие правильности программы. Заметим, что при этом оно необязательно является достаточным. Чем сложнее программа, тем труднее построить исчерпывающий план тестирования. Опыт показывает, что даже в «фирменных» программах в процессе эксплуатации обнаруживаются ошибки. Поэтому проблема тестирования программы — очень важная и одновременно очень сложная проблема.

5.3. Рекурсивные методы

Суть рекурсивных методов — сведение задачи к самой себе. Вы уже знаете, что как в Паскале, так и в Си существует возможность рекурсивного определения функций и процедур. Эта возможность представляет собой способ программной реализации рекурсивных алгоритмов. Однако увидеть рекурсивный путь решения задачи (рекурсивный алгоритм) часто очень непросто.

Рассмотрим классическую задачу, известную в литературе под названием «Ханойская башня» (рис. 50).

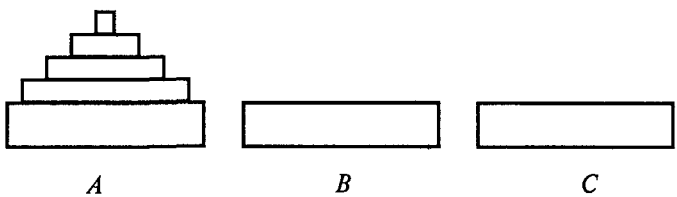


Рис. 50

На площадке (назовем ее A) находится пирамида, составленная из дисков уменьшающегося от основания к вершине размера.

Эту пирамиду в том же виде требуется переместить на площадку B. При выполнении этой работы необходимо соблюдать следующие ограничения:

- перекладывать можно только по одному диску, взятому сверху пирамиды;
- класть диск можно либо только на основание площадки, либо на диск большего размера;
- в качестве вспомогательной можно использовать площадку C.

Название «Ханойская башня» связано с легендой, согласно которой в давние времена монахи одного ханойского храма взялись переместить по этим правилам башню, состоящую из 64 дисков. С завершением их работы наступит конец света. Монахи все еще работают и, надеемся, еще долго будут работать!

Нетрудно решить эту задачу для двух дисков. Обозначая перемещения диска, например, с площадки A на B так: $A \to B$, напишем алгоритм для этого случая

$$A \to C;\ A \to B;\ C \to B.$$

Всего 3 хода! Для трех дисков алгоритм длиннее:

$$A \to B;\ A \to C;\ B \to C;\ A \to B;\ C \to A;\ C \to B;\ A \to B.$$

В этом случае уже требуются 7 ходов.

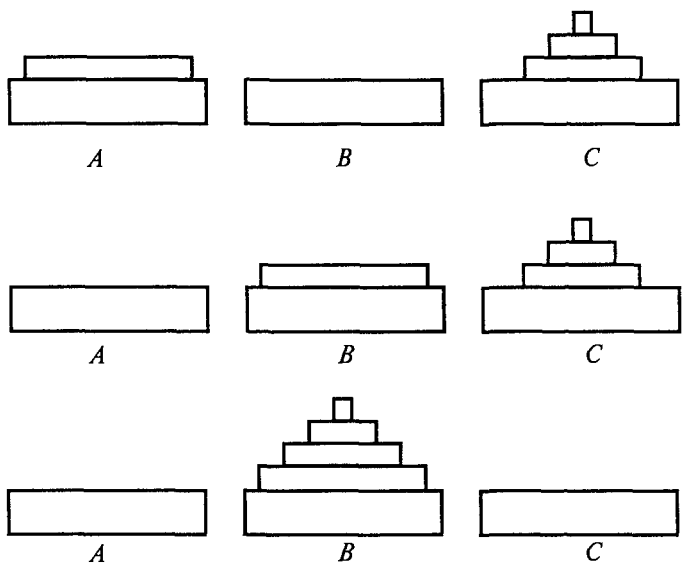


Рис. 51

Подсчитать количество ходов (N) для k дисков можно по следующей рекуррентной формуле:

$$N(1) = 1;\ N(k) = 2 \times N(k-1) + 1.$$

Например, $N(10) = 1023$, $N(20) = 104857$. А вот сколько перемещений нужно сделать ханойским монахам:

$$N(64) = 18446744073709551615.$$

Попробуйте прочитать это число.

Теперь составим программу, по которой машина рассчитает алгоритм работы монахов и выведет его для любого значения n (количества дисков). Пусть на площадке A находятся n дисков. Алгоритм решения задачи будет следующим:

1. Если $n = 0$, то ничего не делать.
2. Если $n > 0$, то переместить $n - 1$ диск на C через B;
   переместить диск с A на B ($A \rightarrow B$)
   переместить $n - 1$ диск с C на B через A.

При выполнении пункта 2 последовательно будем иметь три состояния (рис. 51).

Описание алгоритма имеет явно рекурсивный характер. Перемещение n дисков описывается через перемещение $n - 1$ диска. А где же выход из этой последовательности рекурсивных ссылок алгоритма самого на себя? Он в пункте 1, каким бы ни показалось странным его тривиальное содержание.

А теперь построим программу на Паскале. В ней имеется рекурсивная процедура *Hanoy*, выполнение которой заканчивается только при $n = 0$. При обращении к процедуре используются фактические имена площадок, заданные их номерами: 1, 2, 3. Поэтому на выходе цепочка перемещений будет описываться в таком виде:

$$1 \rightarrow 2 \quad 1 \rightarrow 3 \quad 2 \rightarrow 3 \text{ и т.д.}$$

```
Program Monahi;
Var N: Byte;
Procedure Hanoi(N: Byte;A,B,C:Char);
Begin
   If N>0 Then
   Begin Hanoi(N-1,A,C,B);
   WriteLn(A,'=>',B);
   Hanoi(N-1,C,B,A)
   End
End;
Begin
   WriteLn("Укажите число дисков:");
   ReadLn(N);
   Hanoi(N,'1','2','3')
End.
```

Это одна из самых удивительных программ! Попробуйте воспроизвести ее на машине. Проследите, как изменяется число ходов с ростом *n*. Для этой цели можете сами добавить в программу счетчик ходов и в конце вывести его значение или печатать ходы с порядковыми номерами.

5.4. Методы перебора в задачах поиска

В данном разделе мы рассмотрим некоторые задачи, связанные с проблемой поиска информации. Это огромный класс задач, достаточно подробно описанный в классической литературе по программированию (см., например, книги Н. Вирта, Д. Кнута и другие).

Общий смысл задач поиска сводится к следующему: из данной информации, хранящейся в памяти ЭВМ, выбрать нужные сведения, удовлетворяющие определенным условиям (критериям).

Подобные задачи мы уже рассматривали. Например, поиск максимального числа в числовом массиве, поиск нужной записи в файле данных и т. п. Такой поиск осуществляется перебором всех элементов структуры данных и их проверкой на удовлетворение условию поиска. Перебор, при котором просматриваются все элементы структуры, называется *полным* перебором.

Полный перебор является «лобовым» способом поиска и, очевидно, не всегда самым лучшим.

Рассмотрим пример. В одномерном массиве *X* заданы координаты *n* точек, лежащих на вещественной числовой оси. Точки пронумерованы. Их номера соответствуют последовательности в массиве *X*. Определить номер первой точки, наиболее удаленной от начала координат.

Легко понять, что это знакомая нам задача определения номера наибольшего по модулю элемента массива *X*. Она решается путем полного перебора следующим образом:

```
Const N=100;
Var X: Array[1..N] Of Real; I,Number: 1..N;
Begin
{____Ввод массива X_____}
.......................
Number:=1;
For I:=2 To N Do
    If Abs(X[I])>Abs(X[Number])
    Then Number:=I;
WriteLn(Number)
End.
```

Полный перебор элементов одномерного массива производится с помощью одной циклической структуры.

А теперь такая задача: исходные данные — те же, что и в предыдущей; требуется определить пару точек, расстояние между которыми наибольшее.

Применяя метод перебора, эту задачу можно решать так: перебрать все пары точек из N данных и определить номера тех, расстояние между которыми наибольшее (наибольший модуль разности координат). Такой полный перебор реализуется через два вложенных цикла:

```
Number1:=1;
Number2:=2;
For I:=1 To N Do
For J:=1 To N Do
  If Abs(X[I]-X[J])>Abs(X[Number1]-X[Number2])
  Then
  Begin
    Number1:=I;
    Number2:=J
 End;
```

Очевидно, что такое решение задачи нерационально. Здесь каждая пара точек будет просматриваться дважды, например при $i = 1$, $j = 2$ и $i = 2, j = 1$. Для случая $n = 100$ циклы повторят выполнение $100 \times 100 = 10000$ раз.

Выполнение программы ускорится, если исключить повторения. Исключить также следует и случай совпадения значений i и j. Тогда число повторений цикла будет равно $\frac{n(n-1)}{2}$. При $n = 100$ получается 4950.

Для исключения повторений нужно в предыдущей программе изменить начало внутреннего цикла с 1 на $i + 1$. Программа примет вид:

```
Number1:=1;
Number2:=2;
For I:=1 To N Do
For J:=I+1 To N Do
  If Abs(X[I]-X[J])>Abs(X[Number1]-X[Number2])
  Then
  Begin
    Number1:=I;
    Number2:=J
  End;
```

Рассмотренный вариант алгоритма назовем перебором *без повторений.*

Замечание. Конечно, эту задачу можно было решить и другим способом, но в данном случае нас интересовал именно алгоритм,

связанный с перебором. В случае точек, расположенных не на прямой, а на плоскости или в пространстве, поиск альтернативы такому алгоритму становится весьма проблематичным.

В следующей задаче требуется выбрать все тройки чисел без повторений, сумма которых равна десяти, из массива *X*.

В этом случае алгоритм будет строиться из трех вложенных циклов. Внутренние циклы имеют переменную длину.

```
For I:=1 To N Do
For J:=I+1 To N Do
For K:=J+1 To N Do
If X[I]+X[J]+X[K]=10
Then WriteLn(X[I],X[J],X[K]);
```

А теперь представьте, что из массива *X* требуется выбрать все группы чисел, сумма которых равна десяти. В группах может быть от 1 до *n* чисел. В этом случае количество вариантов перебора резко возрастает, а сам алгоритм становится нетривиальным.

Казалось бы, ну и что? Машина работает быстро! И все же посчитаем. Число различных групп из *n* объектов (включая пустую) составляет 2^n. При $n = 100$ это будет $2^{100} \approx 10^{30}$. Компьютер, работающий со скоростью миллиард операций в секунду, будет осуществлять такой перебор приблизительно 10 лет. Даже исключение перестановочных повторений не сделает такой переборный алгоритм практически осуществимым.

Путь практической разрешимости подобных задач состоит в нахождении способов исключения из перебора бесперспективных с точки зрения условия задачи вариантов. Для некоторых задач это удается сделать с помощью алгоритма, описанного в следующем разделе.

Перебор с возвратом. Рассмотрим алгоритм перебора с *возвратом* на примере задачи о прохождении лабиринта (рис. 52).

Дан лабиринт, оказавшись внутри которого нужно найти выход наружу. Перемещаться можно только в горизонтальном и вер-

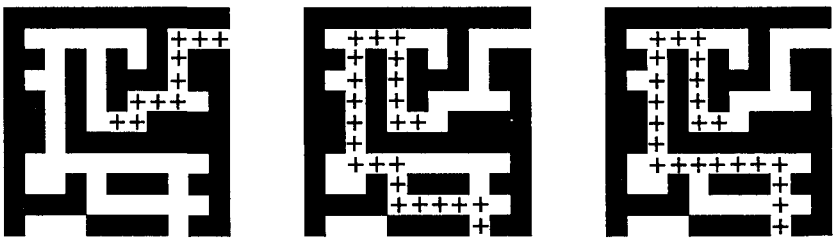


Рис. 52

тикальном направлениях. На рисунке показаны все варианты путей выхода из центральной точки лабиринта.

Для получения программы решения этой задачи нужно решить две проблемы:

- как организовать данные;
- как построить алгоритм.

Информацию о форме лабиринта будем хранить в квадратной матрице LAB символьного типа размером $N \times N$, где N — нечетное число (чтобы была центральная точка). На профиль лабиринта накладывается сетка так, что в каждой ее ячейке находится либо стена, либо проход.

Матрица отражает заполнение сетки: элементы, соответствующие проходу, равны пробелу, а стене — какому-нибудь символу (например, букве M). Путь движения по лабиринту будет отмечаться символами +.

Например, приведенный выше рисунок (в середине) соответствует следующему заполнению матрицы LAB:

```
M  M  M  M  M  M  M  M  M  M  M
M     +  +  +        M
M  M  +  M  +  M     M     M  M
M     +  M  +  M  M  M     M  M
M  M  +  M  +  M                M
M  M  +  M  +  +     M  M  M  M
M  M  +  M  M  M  M  M  M  M  M
M  M  +  +  +                   M
M        M  +  M  M  M     M  M  M
M  M  M  M  +  +  +  +  +     M
M           M  M  M  M  +  M  M
```

Исходные данные — профиль лабиринта (исходная матрица LAB без крестиков); результат — все возможные траектории выхода из центральной точки лабиринта (для каждого пути выводится матрица LAB с траекторией, отмеченной крестиками).

Алгоритм перебора с возвратом еще называют *методом проб*. Суть его в следующем:

1. Из каждой очередной точки траектории просматриваются возможные направления движения в одной и той же последовательности; договоримся, что просмотр будет происходить каждый

раз против часовой стрелки — справа-сверху-слева-снизу; шаг производится в первую же обнаруженную свободную соседнюю клетку; клетка, в которую сделан шаг, отмечается крестиком.

2. Если из очередной клетки дальше пути нет (тупик), то следует возврат на один шаг назад и просматриваются еще не испробованные пути движения из этой точки; при возвращении назад покинутая клетка отмечается пробелом.

3. Если очередная клетка, в которую сделан шаг, оказалась на краю лабиринта (на выходе), то на печать выводится найденный путь.

Программу будем строить методом последовательной детализации. Первый этап детализации:

```
Program Labirint;
Const N=11; {размер лабиринта NxN клеток}
Type Field=Array[1..N,1..N] Of Char;
Var LAB: Field;
Procedure GO(LAB: Field; X,Y: Integer);
Begin
{Поиск путей из центра лабиринта до края
- каждый найденный путь печатается}
End;
Begin
{Ввод лабиринта}
GO(LAB, N Div 2+1, N Div 2+1) {начинаем с середины}
End.
```

Процедура GO пытается сделать шаг в клетку с координатами *x*, *y*. Если эта клетка оказывается на выходе из лабиринта, то пройденный путь выводится на печать. Если нет, то в соответствии с установленной выше последовательностью делается шаг в соседнюю клетку. Если клетка тупиковая, то выполняется шаг назад. Из сказанного выше следует, что процедура носит рекурсивный характер.

Запишем сначала общую схему процедуры без детализации:

```
Procedure GO(LAB: Field; X,Y: Integer);
Begin
    If {клетка (x,y) свободна}
    Then
    Begin
        {шаг на клетку (x,y)}
        If {дошли до края лабиринта}
        Then {печатается найденный путь}
        Else {попытка сделать шаг в соседние клетки
        в условленной последовательности}
        {возвращение на один шаг назад}
    End
End;
```

Для вывода найденных траекторий составляется процедура PRINTLAB.

В окончательном виде программа будет выглядеть так:

```
Program Labirint;
Const N=11; {размер лабиринта NxN клеток}
Type Field=Array[1..N, 1..N] Of Char;
Var LAB: Field; X,Y: Integer;
Procedure PRINTLAB(LAB: Field);
{Печать найденного пути в лабиринте}
Var X,Y: Integer;
Begin
     For X:=1 To N Do
     Begin
            For Y:=1 To N Do
              Write(LAB[X,Y]);
              WriteLn
      End;
     WriteLn
End; {печати}
Procedure GO(LAB: Field; X,Y: Integer);
Begin
     If LAB[X,Y]=' ' {если клетка свободна}
     Then
     Begin
       LAB[X,Y]:='+'; {делается шаг}
       If (X=1)Or(X=N)Or(Y=1)Or(Y=N) {край}
       Then
        PRINTLAB(LAB){печатается найденный путь}
       Else
          Begin {поиск следующего шага}
            GO(LAB,X+1,Y);
            GO(LAB,X, Y+1);
            GO(LAB,X-1,Y);
            GO(LAB,X,Y-1)
          End;
       LAB[X,Y]:='' {возвращение назад}
     End
End; {процедуры GO}
Begin {основной программы}
      {ввод лабиринта}
       For X:=1 To N Do
       Begin
          For Y:=1 To N Do
              Read(LAB[X,Y]);
              ReadLn
       End;
```

```
        GO(LAB,N Div 2+1,N Div 2+1)  {начинаем с
                                      середины}
End.
```

Еще один пример красивой программы с использованием рекурсивного определения процедуры (вспомните ханойскую башню!).

Схема алгоритма данной программы типична для метода перебора с возвратом. По аналогичным алгоритмам решаются, например, популярные задачи об обходе шахматной доски фигурами или о расстановке фигур на доске так, чтобы они «не били» друг друга; множество задач оптимального выбора (задачи о коммивояжере, об оптимальном строительстве дорог и т.п.).

Замечание. Из-за использования массива `LAB` в качестве параметра-значения в процедуре `GO` могут возникнуть проблемы с памятью при реализации программы на ЭВМ. В таком случае можно перейти к глобальной передаче массива.

Упражнения

1. Даны декартовы координаты N точек на плоскости. Составить программы решения следующих задач:

а) найти две самые близкие друг к другу точки;

б) найти две самые удаленные друг от друга точки;

в) найти три точки, лежащие в вершинах треугольника с наибольшим периметром;

г) найти две ближайшие точки, отрезок между которыми может служить радиусом окружности, заключающей внутри себя все остальные точки; указать, какая из них является центральной.

2. Изменить программу `Labirint` таким образом, чтобы на печать выводился лишь кратчайший путь из центра лабиринта до края.

3. Составить программу, в соответствии с которой шахматный конь обойдет всю доску, побывав на каждом поле всего один раз.

4. Составить программу расстановки на шахматной доске восьми ферзей так, чтобы они не угрожали друг другу.

5.5. Эвристические методы

Под эвристическими понимаются такие методы, правильность которых строго не доказывается. Они выглядят правдоподобными; кажется, что в большинстве случаев они должны давать верные решения. На уровне экспертной оценки алгоритма часто не удается придумать контрпример, доказывающий ошибочность или неуниверсальность метода. Это, разумеется, не является строгим обоснованием правильности метода. Тем не менее практика использования эвристических методов дает положительные результаты.

Эвристические методы разнообразны, поэтому нельзя описать какую-то общую схему их разработки. Чаще всего они применяются совместно с методами перебора для сокращения числа проверяемых вариантов. Некоторые варианты согласно выбранной эвристике считаются заведомо бесперспективными и не проверяются. Такой подход ускоряет работу алгоритма по сравнению с полным перебором. Платой за это является отсутствие гарантии того, что выбрано правильное или наилучшее из всех возможных решение.

5.6. Сложность алгоритмов

Традиционно принято оценивать степень сложности алгоритма по объему используемых им основных ресурсов компьютера: процессорного времени и оперативной памяти. В связи с этим вводятся такие понятия, как временная сложность алгоритма и объемная сложность алгоритма.

Параметр временной сложности становится особенно важным для задач, предусматривающих интерактивный режим работы программы, или для задач управления в режиме реального времени. Часто программисту, составляющему программу управления каким-нибудь техническим устройством, приходится искать компромисс между точностью вычислений и временем работы программы. Как правило, повышение точности ведет к увеличению времени.

Объемная сложность программы становится критической, когда объем обрабатываемых данных оказывается на пределе объема оперативной памяти ЭВМ. На современных компьютерах острота этой проблемы снижается благодаря как росту объема ОЗУ, так и эффективному использованию многоуровневой системы запоминающих устройств. Программе оказывается доступной очень большая, практически неограниченная область памяти (виртуальная память). Недостаток основной памяти приводит лишь к некоторому замедлению работы из-за обменов с диском. Используются приемы, позволяющие минимизировать потери времени при таком обмене. Это использование кэш-памяти и аппаратного просмотра команд программы на требуемое число ходов вперед, что позволяет заблаговременно переносить с диска в основную память нужные значения. Исходя из сказанного можно заключить, что минимизация емкостной сложности не является первоочередной задачей. Поэтому в дальнейшем мы будем интересоваться в основном временной сложностью алгоритмов.

Время выполнения программы пропорционально числу исполняемых операций. Разумеется, в размерных единицах времени (секундах) оно зависит еще и от скорости работы процессора (так-

товой частоты). Для того чтобы показатель временной сложности алгоритма был инвариантен относительно технических характеристик компьютера, его измеряют в относительных единицах. Обычно временная сложность оценивается числом выполняемых операций.

Как правило, временная сложность алгоритма зависит от исходных данных. Это может быть зависимость как от величины исходных данных, так и от их объема. Если обозначить значение параметра временной сложности алгоритма α символом T_α, а буквой V обозначить некоторый числовой параметр, характеризующий исходные данные, то временную сложность можно представить как функцию $T_\alpha(V)$. Выбор параметра V зависит от решаемой задачи или от вида используемого алгоритма для решения данной задачи.

Пример 1. Оценим временную сложность алгоритма вычисления факториала целого положительного числа.

```
Function Factorial(x:Integer): Integer;
Var m,i: Integer;
Begin  m:=1;
    For i:=2 To x Do m:=m*i;
    Factorial:=m
End;
```

Подсчитаем общее число операций, выполняемых программой при данном значении x. Один раз выполняется оператор `m:=1`; тело цикла (в котором две операции: умножение и присваивание) выполняется $x - 1$ раз; один раз выполняется присваивание `Factorial:=m`. Если каждую из операций принять за единицу сложности, то временная сложность всего алгоритма будет $1 + 2(x - 1) + 1 = 2x$. Отсюда понятно, что в качестве параметра V следует принять значение x. Функция временной сложности получилась следующей:

$$T_\alpha(V) = 2V.$$

В этом случае можно сказать, что временная сложность зависит линейно от параметра данных — величины аргумента функции факториал.

Пример 2. Вычисление скалярного произведения двух векторов $A = (a_1, a_2, ..., a_k)$, $B = (b_1, b_2, ..., b_k)$.

```
AB:=0;
For i:=1 To k Do  AB:=AB+A[i]*B[i];
```

В этой задаче объем входных данных $n = 2k$. Количество выполняемых операций $1 + 3k = 1 + 3(n/2)$. Здесь можно взять $V = k = n/2$. Зависимости сложности алгоритма от значений элементов векторов A и B нет. Как и в предыдущем примере, здесь можно говорить о линейной зависимости временной сложности от параметра данных.

С параметром временной сложности алгоритма обычно связывают две теоретические проблемы. Первая состоит в поиске ответа на вопрос: до какого предела значения временной сложности можно дойти, совершенствуя алгоритм решения задачи? Этот предел зависит от самой задачи и, следовательно, является ее собственной характеристикой.

Вторая проблема связана с классификацией алгоритмов по временной сложности. Функция $T_\alpha(V)$ обычно растет с ростом V. Как быстро она растет? Существуют алгоритмы с линейной зависимостью T_α от V (как это было в рассмотренных нами примерах), с квадратичной зависимостью и с зависимостью более высоких степеней. Такие алгоритмы называются полиномиальными. А существуют алгоритмы, сложность которых растет быстрее любого полинома. Проблема, которую часто решают теоретики — исследователи алгоритмов, заключается в следующем вопросе: возможен ли для данной задачи полиномиальный алгоритм?

5.7. Методы сортировки данных

Существует традиционное деление алгоритмов на численные и нечисленные. Численные алгоритмы предназначены для математических расчетов: вычисления по формулам, решения уравнений, статистической обработки данных и т.п. В таких алгоритмах основным видом обрабатываемых данных являются числа. Нечисленные алгоритмы имеют дело с самыми разнообразными видами данных: символьной, графической, мультимедийной информацией. К этой категории относятся многие алгоритмы системного программирования (трансляторы, операционные системы), систем управления базами данных, сетевого программного обеспечения и т.д.

Для программных продуктов второй категории наиболее часто используемыми являются алгоритмы сортировки данных — упорядочения информации по некоторому признаку. От эффективности, прежде всего скорости, их выполнения во многом зависит эффективность работы всей программы.

Различают алгоритмы внутренней сортировки — во внутренней памяти и алгоритмы внешней сортировки — сортировки файлов. Далее мы будем рассматривать только внутреннюю сортировку.

Как правило, сортируемые данные располагаются в массивах. В простейшем случае это числовые массивы. Однако для нечисленных алгоритмов более характерна ситуация, когда сортируется массив записей (в терминологии Паскаля) или массив структур (в терминологии Си). Поле, по значению которого производится сортировка, называется *ключом сортировки*. Обычно оно имеет числовой тип. Например, массив сортируемых записей содержит

два поля: наименование товара и количество товара на складе. В программе на Паскале он описан так:

```
Const n=1000;
Type tovar=Record
             name: String;
             key: Integer
           End;
Var A: array[1..n] Of tovar;
```

Сортировка производится либо по возрастанию, либо по убыванию значения ключа A[i].key.

Во всех дальнейших примерах программ предполагается, что приведенные выше описания в программе присутствуют глобально и область их действия распространяется на процедуры сортировки. Хотя все примеры приводятся на Паскале, но по тому же принципу можно разработать функции сортировки на Си/Си++.

Алгоритм сортировки «методом пузырька» рассматривался в разделе 3.17. Здесь мы обсудим два алгоритма: сортировку простым включением и быструю сортировку.

Сортировка простым включением. Предположим, что на некотором этапе работы алгоритма левая часть массива с 1-го по $(i-1)$-й элемент включительно является отсортированной, а правая часть с i-го по n-й элемент остается такой, какой она была в первоначальном, неотсортированном массиве. Очередной шаг алгоритма заключается в расширении левой части на один элемент и, соответственно, сокращении правой части. Для этого берется первый элемент правой части (с индексом i) и вставляется на подходящее ему место в левую часть так, чтобы упорядоченность левой части сохранилась.

Процесс начинается с левой части, состоящей из одного элемента A[1], а заканчивается, когда правая часть становится пустой.

```
Procedure StraightInsertion;
Var i,j: Integer; x: tovar;
Begin  For i:=2 To n Do
Begin x:=A[i]; {В переменной x запоминается
значение, которое нужно поставить на свое место в
левой части}
      j:=i-1; {Правый край левой части}
      While (x.key<A[j].key) and (j>=1) Do
      Begin A[j+1]:=A[j]; j:=j-1; {Продвижение
"дырки" в левой части массива справа налево до той
позиции, в которую должен быть включен элемент
      A[i]}
      End;
      A[j+1]:=x {включение A[i] в "дырку" в левой
части}
```

```
End
End;
```

Теперь оценим сложность алгоритма сортировки простым включением. Очевидно, что временная сложность зависит как от размера сортируемого массива, так и от его исходного состояния в смысле упорядоченности элементов. Временная сложность будет минимальной, если исходный массив уже отсортирован в нужном порядке значений ключа (в данном случае — по возрастанию). Максимальное значение сложности будет соответствовать противоположной упорядоченности исходного массива, т.е. упорядоченности исходного массива по убыванию значений ключа. Обычно для алгоритмов сортировки временная сложность оценивается количеством пересылок элементов.

Оценим величину минимальной временной сложности алгоритма. Если массив уже отсортирован, то тело цикла `while` не будет выполняться ни разу. Выполнение процедуры сведется к работе следующего цикла:

```
For i:=2 To n do
Begin x:=A[i];
      j:=i-1;
      A[j+1]:=x
End;
```

Поскольку тело цикла `for` исполняется $n-1$ раз, то число пересылок элементов массива

$$M_{\min} = 2(n-1),$$

а число сравнений ключей равно

$$C_{\min} = n-1.$$

Сложность алгоритма будет максимальной, если исходный массив упорядочен по убыванию. Тогда каждый элемент `A[i]` будет «прогоняться» к началу массива, т.е. устанавливаться в первую позицию. Цикл `while` выполнится 1 раз при $i=2$, 2 раза при $i=3$ и т.д., $n-1$ раз при $i=n$. Таким образом, общее число пересылок записей равно:

$$M_{\max} = 2(n-1) + \sum_{i=2}^{n}(i-1) = 2(n-1) + n(n-1)/2 = \frac{1}{2}(n^2+3n-4).$$

Более подходящей для реальной ситуации является *средняя оценка сложности*. Для ее вычисления надо предположить, что все элементы исходного массива — случайные числа и их значения никак не связаны с их номерами. В таком случае результат очередной проверки условия `x.key<A[j].key` в цикле `While` также является случайным.

Разумно допустить, что среднее число выполнений цикла While для каждого конкретного значения i равно $i/2$, т.е. в среднем каждый раз приходится просматривать половину последовательности до тех пор, пока не найдется подходящее место для очередного элемента. Тогда формула для среднего числа пересылок (средняя оценка сложности) будет следующей:

$$M_{ср} = 2(n-1) + \sum_{i=2}^{n} i/2 = 2(n-1) + (n-2)(n-1)/4 = \frac{1}{4}(n^2 + 9n - 10).$$

Как максимальная, так и средняя оценка сложности алгоритма квадратична (является полиномом второй степени) по параметру n — размеру сортируемого массива.

Алгоритм быстрой сортировки. Этот алгоритм был разработан Э. Хоаром. В алгоритме быстрой сортировки используются три идеи:

- разделение сортируемого массива на 2 части, левую и правую;
- взаимное упорядочение двух частей (подмассивов) так, чтобы все элементы левой части не превосходили элементов правой части;
- рекурсия, при которой подмассив упорядочивается точно таким же способом, как и весь массив.

Для разделения массива на две части нужно выбрать некоторое «барьерное» значение ключа. Это значение должно удовлетворять единственному условию: лежать в диапазоне значений для данного массива (т.е. между минимальной и максимальной величиной). За «барьер» можно выбрать значение ключа любого элемента массива, например первого, или последнего, или находящегося в середине.

Далее нужно сделать так, чтобы в левом подмассиве оказались все элементы с ключом, меньшим барьера, а в правом — с большим. Затем, просматривая массив слева направо, необходимо найти позицию первого элемента с ключом, большим барьера, а просматривая справа налево — найти первый элемент с ключом, меньшим барьера. Следует поменять эти значения, затем продолжить встречное движение до следующей пары элементов, предназначенных для обмена. Необходимо повторять эту процедуру, пока индексы левого и правого просмотров не совпадут. Место совпадения станет границей между двумя взаимно упорядоченными подмассивами. Далее алгоритм рекурсивно применяется к каждому из подмассивов (левому и правому). В конечном счете приходим к совокупности из n взаимно упорядоченных одноэлементных массивов, которые делить дальше невозможно. Эта совокупность образует один полностью упорядоченный массив. Сортировка завершена!

```
Procedure Quicksort;
  Procedure Sort(L,R:Integer);
```

```
Var i,j,bar: Integer; w:tovar;
Begin i:=L; j:=R;
    bar:=A[(L+R)div2].key;{Установка барьера}
    Repeat
      {Поиск элемента слева для обмена}
      While A[i].key<bar Do i:=i+1;
      {Поиск элемента справа для обмена}
      While A[j].key>bar Do j:=j-1;
      {Обмен элементов и смещение по массиву}
      If i<=j Then
      Begin w:=A[i]; A[i]:=A[j]; A[j]:=w;
            i:=i+1; j:=j-1
      End;
    Until i>j;
{сформированы взаимно упорядоченные подмассивы}
    {Сортировка левого подмассива}
    If L<j Then Sort(L,j);
    {Сортировка правого подмассива}
    If i<R Then Sort(i,R);
End; {Sort}
Begin Sort(1,n)
End. {Quicksort}
```

Сложность алгоритма быстрой сортировки. Исследование временной сложности алгоритма быстрой сортировки является очень трудоемкой задачей, и поэтому мы здесь приводить его не будем. Рассмотрим лишь окончательный результат этого анализа. Временная сложность T как функция от n — размера массива — по порядку величины выражается следующей формулой:

$$T(n) = O(n \ln(n)).$$

Здесь использовано принятое в математике обозначение: $O(x)$ обозначает величину порядка x. Следовательно, временная сложность алгоритма быстрой сортировки есть величина порядка $n \ln(n)$. Эта величина для целых положительных n меньше, чем n^2 (вспомним, что алгоритм сортировки простым включением имеет сложность порядка n^2). И чем больше значение n, тем эта разница существеннее. Например:

$n = 10;$	$n^2 = 100;$	$n \ln(n) = 23{,}03;$
$n = 100;$	$n^2 = 10\,000;$	$n \ln(n) = 460{,}52.$

ГЛАВА 6. ЗАДАЧИ ПО ПРОГРАММИРОВАНИЮ

В этой главе представлена большая подборка задач, которые могут быть использованы для организации практических занятий по программированию как на Паскале, так и на Си++.

Задачи систематизированы по конструкциям языка, охватывают 17 тем и сгруппированы в разделы. Нумерация упражнений в каждом разделе начинается с номера один. Кроме того, предлагается набор «больших проектов», которые могут быть реализованы в конце изучения курса программирования для закрепления и развития навыков программирования.

В большинстве разделов задачи разделены по трудности на три уровня:

A — содержит простые задачи, решения которых сводятся к типовым алгоритмам;

B — содержит более сложные задачи, требующие сочетания типовых алгоритмов и определенного творческого подхода;

C — содержит задачи, рекомендованные для выполнения наиболее подготовленными учащимися.

Задачи составлены с учетом опыта проведения практических и лабораторных занятий по программированию на математическом факультете и факультете информатики и экономики Пермского государственного педагогического университета. Использованы также задачи из сборников, приведенных в списке литературы. Раздел «Задачи по теме "Модули"» содержит только авторские разработки.

6.1. Задачи по теме «Линейные программы»

6.1.1. Вычисления по формулам

Вычислить значение выражения по формуле (все переменные принимают действительные значения):

1. $\dfrac{b+\sqrt{b^2+4ac}}{2a} - a^3c + b^{-2}$;

2. $\dfrac{a}{c}\cdot\dfrac{b}{d} - \dfrac{ab-c}{cd}$;

3. $\dfrac{\sin x + \cos y}{\cos x - \sin y}\operatorname{tg} xy$;

4. $\dfrac{x+y}{y+1} - \dfrac{xy-12}{34+x}$;

5. $\dfrac{3+e^{y-1}}{1+x^2|y-\operatorname{tg}x|}$;

6. $x-\dfrac{x^3}{3}+\dfrac{x^5}{5}$;

7. $\ln\left|\left(y-\sqrt{|x|}\right)\left(x-\dfrac{y}{x+\dfrac{x^2}{4}}\right)\right|$

8. $(1-\operatorname{tg}x)^{\operatorname{ctg}x}+\cos(x-y)$;

9. $\dfrac{\ln|\cos x|}{\ln(1+x^2)}$;

10. $\left(\dfrac{x+1}{x-1}\right)^x+18xy^2$;

11. $\left(1+\dfrac{1}{x^2}\right)^x-12x^2y$;

12. $\dfrac{x^2-7x+10}{x^2-8x+12}$;

13. $\dfrac{\cos x}{\pi-2x}+16x\cos(xy)-2$;

14. $2^{-x}-\cos x+\sin(2xy)$;

15. $2\operatorname{ctg}(3x)-\dfrac{1}{12x^2+7x-5}$;

16. $|x^2-x^3|-\dfrac{7x}{x^3-15x}$;

17. $x\ln x+\dfrac{y}{\cos x-\dfrac{x}{3}}$;

18. $\sin\sqrt{x+1}-\sin\sqrt{x-1}$;

19. $e^x-\dfrac{y^2+12xy-3x^2}{18y-1}$;

20. $\dfrac{1+\sin\sqrt{x+1}}{\cos(12y-4)}$;

21. $2\operatorname{ctg}(3x)-\dfrac{\ln\cos x}{\ln(1+x^2)}$;

22. $e^x-x-2+(1+x)^x$;

23. $3^x-4x+\left(y-\sqrt{|x|}\right)$;

24. $x-10\sin x+|x^4-x^5|$;

25. $x-10^{\sin x}+\cos(x-y)$;

26. $\dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}+x$;

27. $\cos^2\left(\sin\dfrac{1}{z}\right)$;

28. $\dfrac{\cos^2 x}{\sin x}-xyz+\dfrac{ax^2+bx+c}{dx^3-f}$.

6.1.2. Вычисления в математических задачах

1. Вычислить периметр и площадь прямоугольного треугольника по длинам a и b двух катетов.

2. Заданы координаты трех вершин треугольника (x_1, y_1), (x_2, y_2), (x_3, y_3). Найти его периметр и площадь.

3. Вычислить длину окружности и площадь круга одного и того же заданного радиуса R.

4. Найти произведение цифр заданного четырехзначного числа.

5. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.

6. Вычислить расстояние между двумя точками с данными координатами (x_1, y_1) и (x_2, y_2).

7. Даны два действительных числа x и y. Вычислить их сумму, разность, произведение и частное.

8. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

9. Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоту, радиусы вписанной и описанной окружностей.

10. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

11. Найти площадь кольца, внутренний радиус которого равен r, а внешний — R ($R > r$).

12. Треугольник задан величинами своих углов и радиусом описанной окружности. Найти стороны треугольника.

13. Найти площадь равнобедренной трапеции с основаниями a и b и углом α при большем основании a.

14. Вычислить корни квадратного уравнения $ax^2 + bx + c = 0$ с заданными коэффициентами a, b и c (предполагается, что $a \neq 0$ и что дискриминант уравнения неотрицателен).

15. Дано действительное число x. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить за минимальное число операций

$$2x^4 - 3x^3 + 4x^2 - 5x + 6.$$

16. Дано значение x. Получить значения $-2x + 3x^2 - 4x^3$ и $1 + 2x + 3x^2 + 4x^3$. Позаботиться об экономии операций.

17. Найти площадь треугольника, две стороны которого равны a и b, а угол между этими сторонами γ.

18. Дано значение a. Не используя никаких функций и никаких операций, кроме умножения, получить значение a^8 за три операции и a^{10} за четыре операции.

19. Написать программу, которая выводит на экран первые четыре степени числа π.

20. Найти сумму членов арифметической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.

21. Найти (в радианах в градусах) все углы треугольника со сторонами a, b, c.

22. Составить программу перевода радианной меры угла в градусы, минуты и секунды.

23. Три сопротивления R_1, R_2, R_3 соединены параллельно. Найдите сопротивление соединения.

24. Составить программу для вычисления пути, пройденного лодкой, если ее скорость в стоячей воде v км/ч, скорость течения реки v_1 км/ч, время движения по озеру t_1 ч, а против течения реки — t_2 ч.

25. Текущее показание электронных часов: m ч ($0 \le m \le 23$) n мин ($0 \le n \le 59$) k с ($0 \le k \le 59$). Какое время будут показывать часы через p ч q мин r с?

26. Вычислить высоты треугольника со сторонами a, b, c.

27. Полторы кошки за полтора часа съедают полторы мышки. Сколько мышек съедят X кошек за Y часов?

28. Составить программу вычисления объема цилиндра и конуса, которые имеют одинаковую высоту H и одинаковый радиус основания R.

29. Ввести любой символ и определить его порядковый номер, а также указать предыдущий и последующий символы.

30. Дана величина A, выражающая объем информации в байтах. Перевести A в более крупные единицы измерения информации.

31. Даны натуральные числа M и N. Вывести старшую цифру дробной части и младшую цифру целой части числа M/N.

32. Дано натуральное число T, которое представляет длительность прошедшего времени в секундах. Вывести данное значение длительности в часах, минутах и секундах в следующей форме: HH ч MM мин SS с.

33. Дано действительное число R вида **nnn.ddd** (три цифровых разряда в дробной и целой частях). Поменять местами дробную и целую части числа и вывести полученное значение числа.

34. Заданы два вектора с координатами (X_1, Y_1, Z_1) и (X_2, Y_2, Z_2). Определить угол между векторами.

35. Вычислить площадь и периметр правильного N-угольника, описанного около окружности радиуса R (рассмотреть N — целого типа, R — вещественного типа).

36. Определить, во сколько раз площадь круга радиуса R больше площади сегмента, отсеченного хордой длины A.

37. Найти частное произведений четных и нечетных цифр четырехзначного числа.

38. Задан вектор с координатами (x, y, z). Найти углы наклона этого вектора к координатным осям.

39. Найти площадь круга, вписанного в треугольник с заданными сторонами.

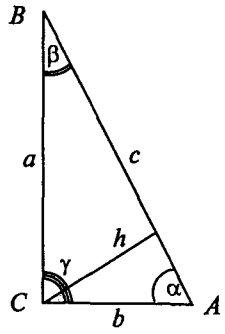


Рис. 53

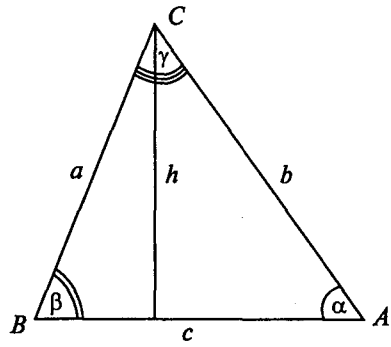


Рис. 54

40. Окружность вписана в квадрат заданной площади. Найти площадь квадрата, вписанного в эту окружность. Во сколько раз площадь вписанного квадрата меньше площади заданного?

41. Представить комплексное число $A + Bi$ (A, B — вещественные) в тригонометрическом виде.

42. Треугольник задан величинами своих углов и радиусом вписанной окружности. Найти стороны треугольника.

43. Дан прямоугольный треугольник ABC ($\gamma = 90°$), для которого определен следующий набор характерных параметров (рис. 53): a, b, c — стороны треугольника; α, β — острые углы (в градусах); h — высота, опущенная на гипотенузу c; S — площадь; P — периметр треугольника. По двум заданным параметрам вычислить все остальные. Возможные сочетания данных параметров:

а) a, b; б) a, c; в) a, h; г) b, α; д) h, β; е) c, β.

44. Дан произвольный треугольник ABC (рис. 54), для которого определен следующий набор характерных параметров: a, b, c — стороны треугольника; α, β, γ — углы (в градусах); h — высота, опущенная на сторону c; S — площадь; P — периметр треугольника. По трем заданным параметрам вычислить все остальные. Возможные сочетания параметров:

1) a, b, c;	2) a, b, γ;	3) c, α, β;
4) h, c, b;	5) h, c, α;	6) S, h, β;
7) S, h, α;	8) a, b, h;	9) a, b, S;
10) a, b, P;	11) a, h, α;	12) a, h, γ;
13) S, c, α;	14) h, α, β;	15) h, α, γ.

6.1.3. Задачи на составление логических выражений

Составить линейную программу, печатающую значение `true`, если указанное высказывание является истинным, и `false` — в противном случае.

1. Сумма двух первых цифр заданного четырехзначного числа равна сумме двух его последних цифр.

2. Сумма цифр данного трехзначного числа N является четным числом.

3. Точка с координатами (x, y) принадлежит части плоскости, лежащей между прямыми $x = m$, $x = n$ $(m < n)$.

4. Квадрат заданного трехзначного числа равен кубу суммы цифр этого числа.

5. Целое число N является четным двузначным числом.

6. Треугольник со сторонами a, b, c является равносторонним.

7. Треугольник со сторонами a, b, c является равнобедренным.

8. Среди чисел a, b, c есть хотя бы одна пара взаимно противоположных.

9. Числа a и b выражают длины катетов одного прямоугольного треугольника, а c и d — другого. Эти треугольники являются подобными.

10. Даны три стороны одного и три стороны другого треугольника. Эти треугольники равновеликие, т.е. имеют равные площади.

11. Данная тройка натуральных чисел a, b, c является тройкой Пифагора, т.е. $c^2 = a^2 + b^2$.

12. Все цифры данного четырехзначного числа N различны.

13. Данные числа x, y являются координатами точки, лежащей в первой координатной четверти.

14. (x_1, y_1) и (x_2, y_2) — координаты левой верхней и правой нижней вершин прямоугольника. Точка $A(x, y)$ принадлежит прямоугольнику.

15. Число c является средним арифметическим чисел a и b.

16. Натуральное число N является точным квадратом.

17. Цифры данного четырехзначного числа N образуют строго возрастающую последовательность.

18. Цифры данного трехзначного числа N являются членами арифметической прогрессии.

19. Цифры данного трехзначного числа N являются членами геометрической прогрессии.

20. Данные числа c и d являются соответственно квадратом и кубом числа a.

21. Цифра M входит в десятичную запись четырехзначного числа N.

22. Данное четырехзначное число читается одинаково слева направо и справа налево.

23. Шахматный конь за один ход может переместиться с одного заданного поля на другое (каждое поле задано двумя координатами — целыми числами от 1 до 8).

24. В заданном натуральном трехзначном числе N имеется четная цифра.

25. Сумма каких-либо двух цифр заданного трехзначного натурального числа N равна третьей цифре.

26. Заданное число N является степенью числа a (показатель степени может находиться в диапазоне от 0 до 4).

27. Сумма цифр заданного четырехзначного числа N превосходит произведение цифр этого же числа на 1.

28. Сумма двух последних цифр заданного трехзначного числа N меньше заданного числа K, а первая цифра больше 5.

29. Заданное натуральное число N является двузначным и кратно K.

30. Сумма двух первых цифр заданного четырехзначного числа N равна произведению двух последних.

31. X — отрицательное целое число, делящееся на K нацело.

32. Среди заданных целых чисел A, B, C, D есть хотя бы два четных.

33. Прямоугольник с измерениями A, B подобен прямоугольнику с соответствующими измерениями C, D.

34. Дробь A/B является правильной.

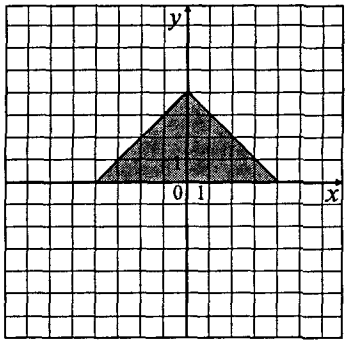
35. Шахматная ладья за один ход может переместиться с одного заданного поля на другое (каждое поле задано двумя координатами — целыми числами от 1 до 8).

36. График функции $y = ax^2 + bx + c$ проходит через заданную точку с координатами (m, n).

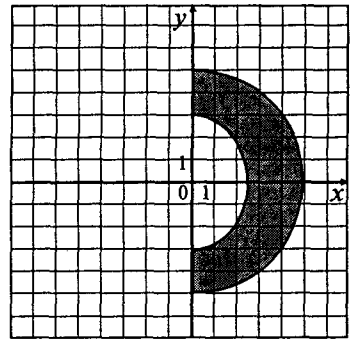
37. Величина d является корнем только одного из уравнений $ax^2 + bx + c = 0$ и $mx + n = 0$.

6.1.4. Области, описываемые логическими выражениями

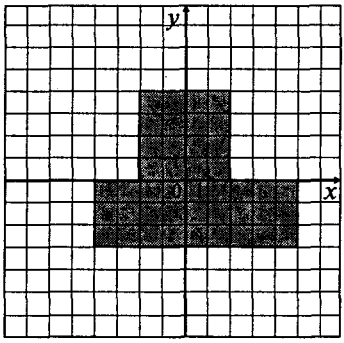
Для данных областей составить линейную программу, которая печатает `true`, если точка с координатами (x, y) принадлежит закрашенной области, и `false` — в противном случае:



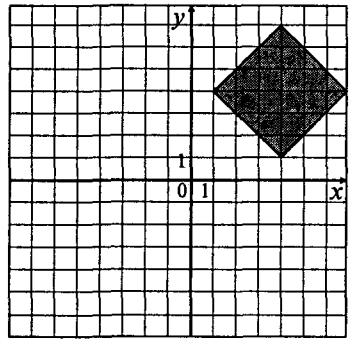
1.



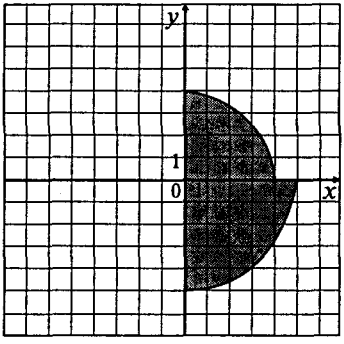
2.



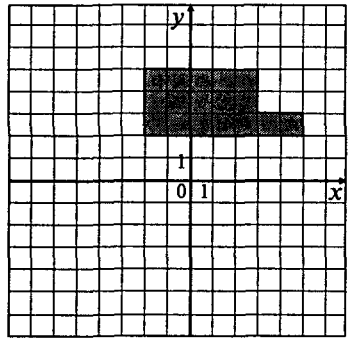
3.



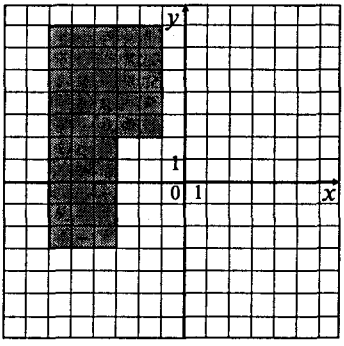
4.



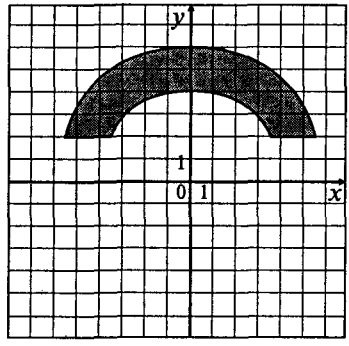
5.



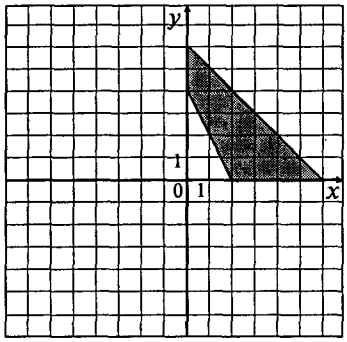
6.



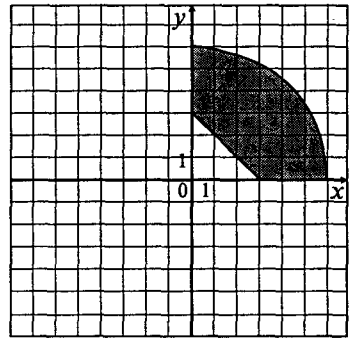
7.



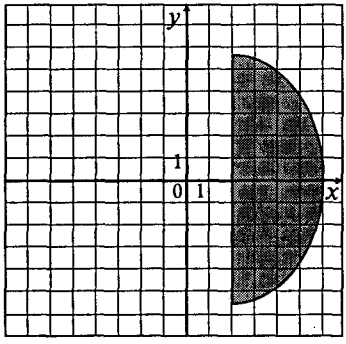
8.



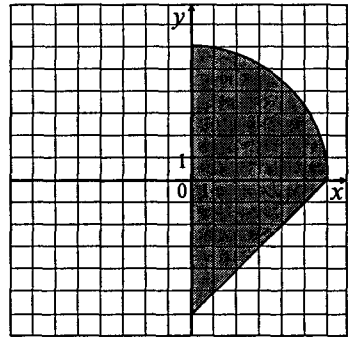
9.



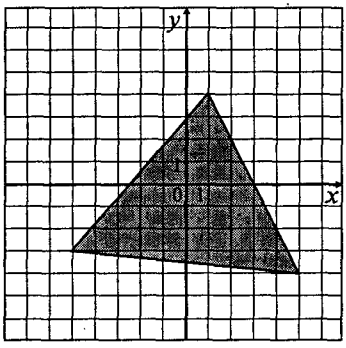
10.



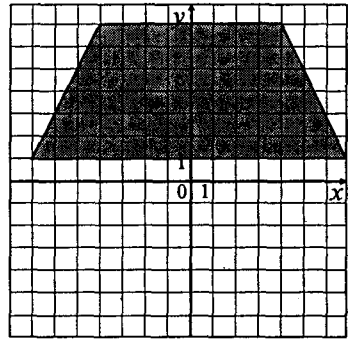
11.



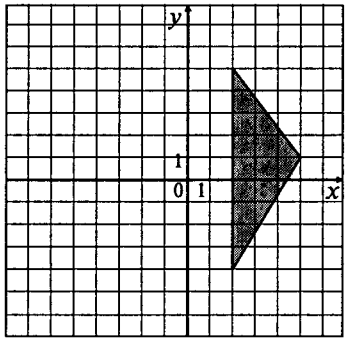
12.



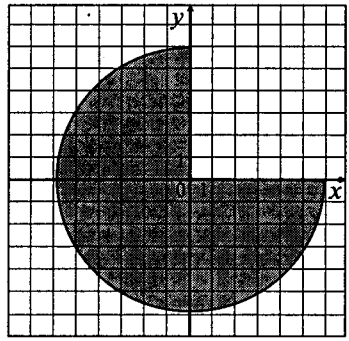
13.



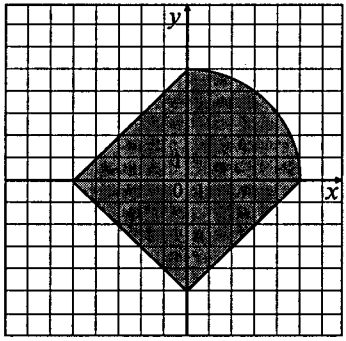
14.



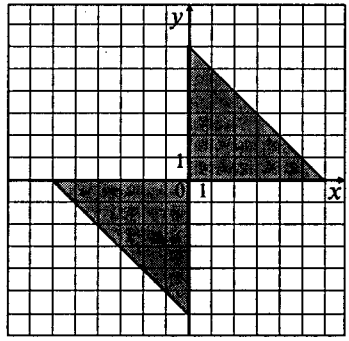
15.



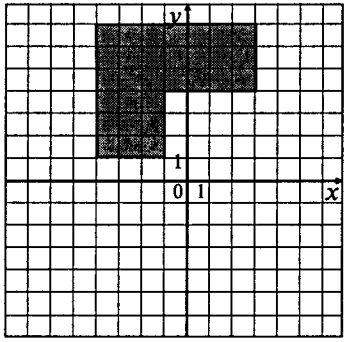
16.



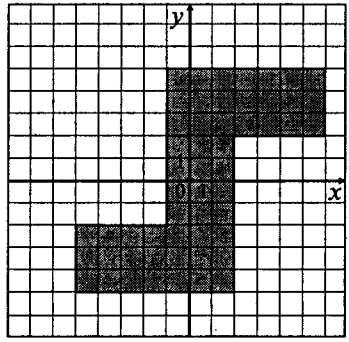
17.



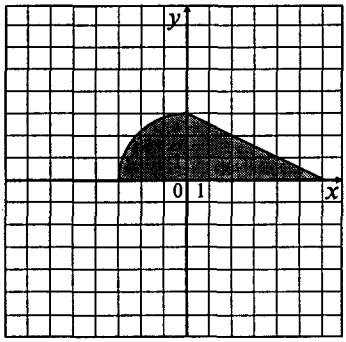
18.



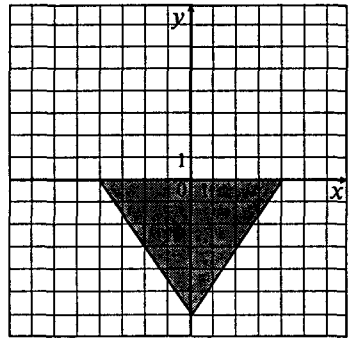
19.



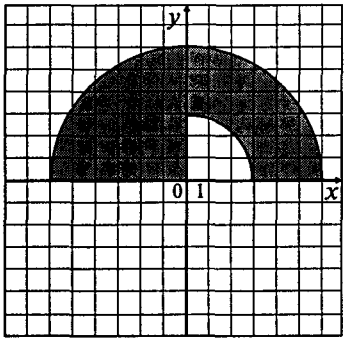
20.



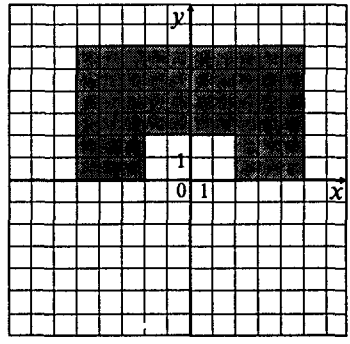
21.



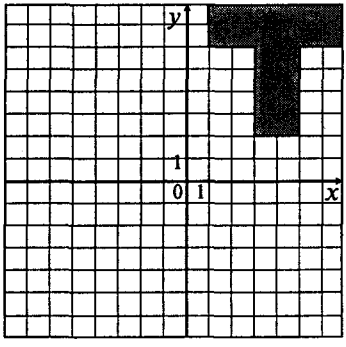
22.



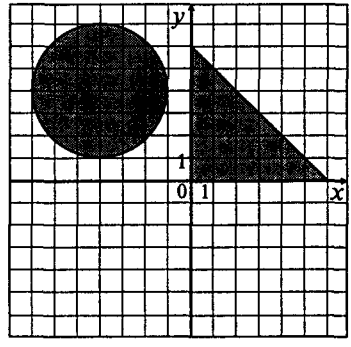
23.



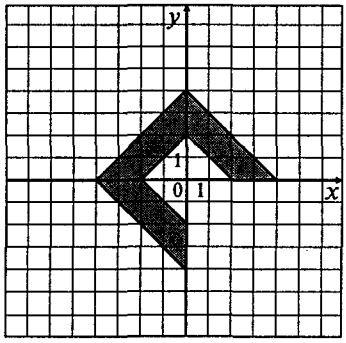
24.



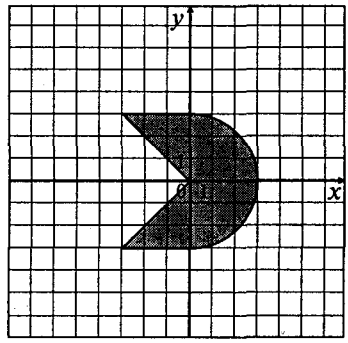
25.



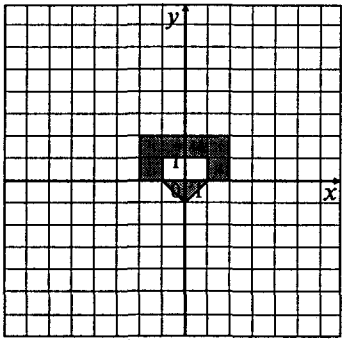
26.



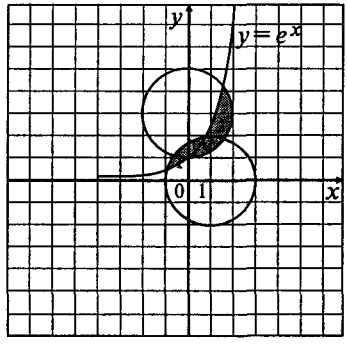
27.



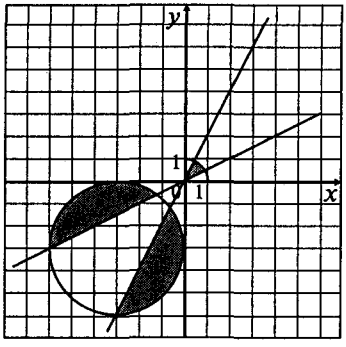
28.



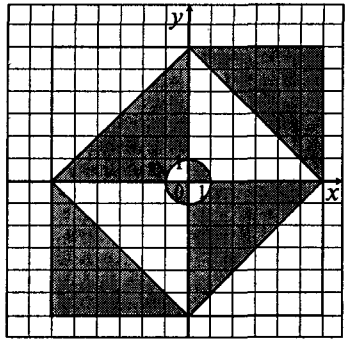
29.



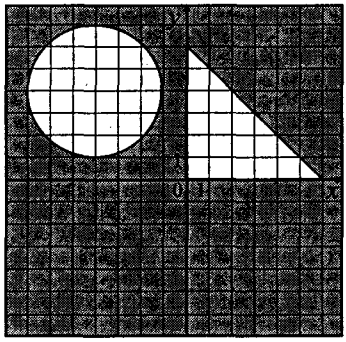
30.



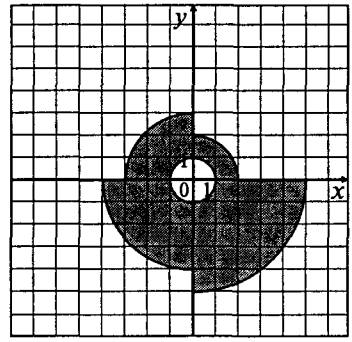
31.



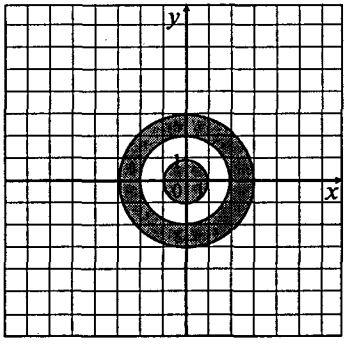
32.



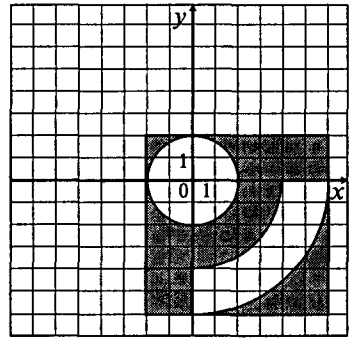
33.



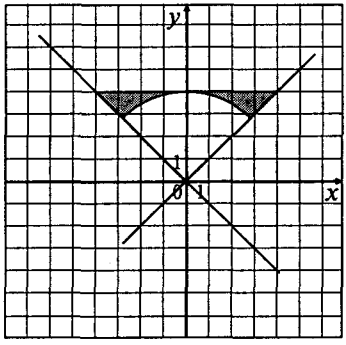
34.



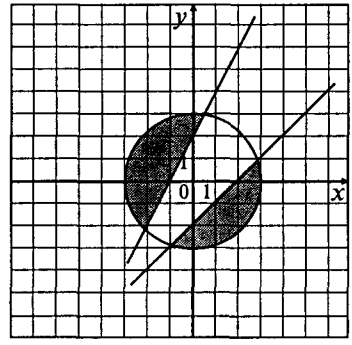
35.



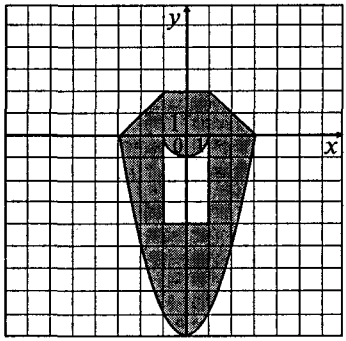
36.



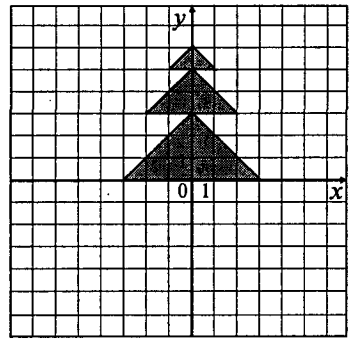
37.



38.



39.



40.

6.2. Задачи по теме «Развилка»

6.2.1. Текстовые задачи

А

1. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны, и в четвертую степень — отрицательные.

2. Даны две точки $A(x_1, y_1)$ и $B(x_2, y_2)$. Составить алгоритм, определяющий, которая из точек находится ближе к началу координат.

3. Даны два угла треугольника (в градусах). Определить, существует ли такой треугольник, и если да, то будет ли он прямоугольным.

4. Даны действительные числа x и y, не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее — их удвоенным произведением.

5. На плоскости XOY задана своими координатами точка A. Указать, где она расположена (на какой оси или в каком координатном угле).

6. Даны целые числа m, n. Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить числа нулями.

7. Подсчитать количество отрицательных среди чисел a, b, c.

8. Подсчитать количество положительных среди чисел a, b, c.

9. Подсчитать количество целых среди чисел a, b, c.

10. Определить, делителем каких чисел a, b, c является число k.

11. Услуги телефонной сети оплачиваются по следующему правилу: за разговоры до A минут в месяц — B руб., а разговоры сверх установленной нормы оплачиваются из расчета C руб. за минуту. Написать программу, вычисляющую плату за пользование телефоном для введенного времени разговоров за месяц.

12. Программа — льстец. На экране высвечивается вопрос «Кто ты: мальчик или девочка? Введи Д или М». В зависимости от ответа на экране должен появиться текст «Мне нравятся девочки!» или «Мне нравятся мальчики!».

13. Грузовой автомобиль выехал из одного города в другой со скоростью v_1 км/ч. Через t ч в этом же направлении выехал легковой автомобиль со скоростью v_2 км/ч. Составить программу, определяющую, догонит ли легковой автомобиль грузовой через t_1 ч после своего выезда.

14. Перераспределить значения переменных x и y так, чтобы в x оказалось большее из этих значений, а в y — меньшее.

15. Определить правильность даты, введенной с клавиатуры (число — от 1 до 31, месяц — от 1 до 12). Если введены некорректные данные, то сообщить об этом.

16. Составить программу, определяющую результат гадания на ромашке — «любит — не любит», взяв за исходное данное количество лепестков n.

17. Написать программу — модель анализа пожарного датчика в помещении, которая выводит сообщение «Пожароопасная ситуация», если температура в комнате превысила 60 °С.

18. Рис расфасован в два пакета. Масса первого — m кг, второго — n кг. Составить программу, определяющую:

а) какой пакет тяжелее — первый или второй;

б) массу более тяжелого пакета.

19. Написать программу, которая анализирует данные о возрасте и относит человека к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст вводится с клавиатуры.

20. Составить программу, определяющую, пройдет ли график функции $y = ax^2 + bx + c$ через заданную точку с координатами (m, n).

21. К финалу конкурса лучшего по профессии «Специалист электронного офиса» были допущены трое: Иванов, Петров, Сидоров. Соревнования проходили в три тура. Иванов в первом туре набрал m_1 баллов, во втором — n_1, в третьем — p_1. Петров — m_2, n_2, p_2 соответственно; Сидоров — m_3, n_3, p_3. Составить программу, определяющую, сколько баллов набрал победитель.

22. Написать программу-фильтр, которая при нажатии любых клавиш выводит на экран только буквы и цифры, при этом указывая, что выводится: буква или цифра.

В

1. Написать программу нахождения суммы большего и меньшего из трех чисел.

2. Написать программу, по длинам сторон распознающую среди всех треугольников ABC прямоугольные. Если таковых нет, то вычислить величину угла C.

3. Найти $\max\{\min(a, b), \min(c, d)\}$.

4. Даны три числа a, b, c. Определить, какое из них равно d. Если ни одно не равно d, то найти $\max(d-a, d-b, d-c)$.

5. Даны четыре точки $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$, $A_4(x_4, y_4)$. Определить, будут ли они вершинами параллелограмма.

6. Даны три точки $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$. Определить, будут ли они расположены на одной прямой. Если нет, то вычислить $\angle ABC$.

7. Даны действительные числа a, b, c. Удвоить эти числа, если $a \geq b \geq c$, и заменить их абсолютными значениями, если это не так.

8. На оси OX расположены три точки a, b, c. Определить, какая из точек b или c расположена ближе к a.

9. Даны три положительных числа a, b, c. Проверить, будут ли они сторонами треугольника. Если да, то вычислить площадь этого треугольника.

10. Написать программу решения уравнения $ax^3 + bx = 0$ для произвольных a, b.

11. Дан круг радиуса R. Определить, поместится ли правильный треугольник со стороной a в этом круге.

12. Даны числа x, y, z. Найти значение выражения:

$$u = \frac{\max^2(x, y, z) - 2^x \min(x, y, z)}{\sin 2x + \max(x, y, z) / \min(x, y, z)}.$$

13. Дано число x. Напечатать в порядке возрастания числа: $\sin x$, $\cos x$, $\ln x$. Если при каком-либо x некоторые из выражений не имеют смысла, вывести сообщение об этом и сравнивать значения только тех, которые имеют смысл.

14. Заданы размеры A, B прямоугольного отверстия и размеры x, y, z кирпича. Определить, пройдет ли кирпич через отверстие.

15. Составить программу, осуществляющую перевод величин из радианной меры в градусную и наоборот. Программа должна запрашивать, какой перевод нужно осуществить, и выполнять указанное действие.

16. Два прямоугольника, расположенные в первом квадранте, со сторонами, параллельными осям координат, заданы координатами своих левого верхнего и правого нижнего углов. Для первого прямоугольника это точки (x_1, y_1) и $(x_2, 0)$, для второго — (x_3, y_3), $(x_4, 0)$. Составить программу, определяющую, пересекаются ли данные прямоугольники, и вычисляющую площадь общей части, если она существует.

17. В небоскребе N этажей и всего один подъезд; на каждом этаже по 3 квартиры; лифт может останавливаться только на нечетных этажах. Человек садится в лифт и набирает номер нужной ему квартиры M. На какой этаж должен доставить лифт пассажира?

18. Написать программу, которая по заданным трем числам определяет, является ли сумма каких-либо двух из них положительной.

19. Известно, что из четырех чисел a_1, a_2, a_3 и a_4 одно отлично от трех других, равных между собой; присвоить номер этого числа переменной n.

20. Составить программу, которая проверяла бы, не приводит ли суммирование двух целых чисел A и B к переполнению (т.е. к результату большему чем 32 767). Если будет переполнение, то сообщить об этом, иначе вывести сумму этих чисел.

21. Написать программу, которая по паролю будет определять уровень доступа сотрудника к секретной информации в базе данных. Доступ к базе имеют только шесть человек, разбитых на три группы по степени доступа. Они имеют следующие пароли: 9583, 1747 — доступны модули баз A, B, C; 3331, 7922 — доступны модули баз B, C; 9455, 8997 — доступен модуль базы C.

22. Составить программу, реализующую эпизод применения компьютера в книжном магазине. Компьютер запрашивает стоимость книг, сумму денег, внесенную покупателем; если сдачи не требуется, печатает на экране «спасибо»; если денег внесено больше, чем необходимо, то печатает «возьмите сдачу» и указывает сумму сдачи; если денег недостаточно, то печатает сообщение об этом и указывает размер недостающей суммы.

23. В ЭВМ поступают результаты соревнований по плаванию для трех спортсменов. Составить программу, которая выбирает лучший результат и выводит его на экран с сообщением, что это результат победителя заплыва.

24. Определить взаимное расположение точки с координатами (x_0, y_0) и окружности радиуса R с центром в точке (x_1, y_1).

25. По координатам вершин четырехугольника определить, выпуклый он или нет.

26. Вычислить число и месяц в невисокосном году по номеру дня.

C

1. Даны действительные числа a, b, c ($a > 0$). Полностью исследовать биквадратное уравнение $ax^4 + bx^2 + c = 0$ (если действительных корней нет, то должно быть выдано сообщение об этом, иначе найти действительные корни, сообщив, сколько среди них различных).

2. Дана точка $A(x, y)$. Определить, принадлежит ли она треугольнику с вершинами в точках (x_1, y_1), (x_2, y_2), (x_3, y_3).

3. Написать программу, определяющую, будут ли прямые $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$ перпендикулярны. Если нет, то найти угол между ними.

4. Если сумма трех попарно различных действительных чисел X, Y, Z меньше единицы, то наименьшее из этих трех чисел за-

менить полусуммой двух других; в противном случае заменить меньшее из X, Y полусуммой двух оставшихся значений.

5. Написать программу для решения системы линейных уравнений

$$\begin{cases} a_1x + b_1y = c_1, \\ a_2x + b_2y = c_2. \end{cases}$$

6. Даны три положительных числа. Определить, можно ли построить треугольник со сторонами, длины которых равны этим числам. Если возможно, то ответить на вопрос, является ли он остроугольным.

7. Дано действительное число h. Выяснить, имеет ли уравнение $ax_2 + bx + c = 0$ действительные корни, если

$$a = \sqrt{\frac{|\sin 8h| + 17}{(1 - \sin 4h \cos(h^2 + 18))^2}},$$

$$b = 1 - \sqrt{\frac{3}{3 + |\operatorname{tg} ah^2 - \sin ah|}},$$

$$c = ah^2 \sin bh + bh^3 \cos ah.$$

Найти действительные корни или сообщить об их отсутствии.

8*. Заданы координаты вершин прямоугольника: (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Определить площадь части прямоугольника, расположенной в I координатной четверти.

9. Найти координаты точек пересечения прямой $y = kx + b$ и окружности радиуса R с центром в начале координат. В каких координатных четвертях находятся точки пересечения? Если точек пересечения нет или прямая касается окружности, выдать соответствующее сообщение.

6.2.2. Вычисление значений функций

Вычислить значение функции:

1. $F(x) = \begin{cases} x^2 - 3x + 9, & \text{если } x \le 3; \\ \dfrac{1}{x^3 + 6}, & \text{если } x > 3. \end{cases}$

2. $F(x) = \begin{cases} -x^2 + 3x + 9, & \text{если } x \ge 3; \\ \dfrac{1}{x^3 - 6}, & \text{если } x < 3. \end{cases}$

3. $F(x) = \begin{cases} 9, & \text{если } x \le -3; \\ \dfrac{1}{x^2+1}, & \text{если } x > -3. \end{cases}$

4. $F(x) = \begin{cases} 0, & \text{если } x \le 1; \\ \dfrac{1}{x+6}, & \text{если } x > 1. \end{cases}$

5. $F(x) = \begin{cases} -3x+9, & \text{если } x \le 7; \\ \dfrac{1}{x-7}, & \text{если } x > 7. \end{cases}$

6. $F(x) = \begin{cases} 3x-9, & \text{если } x \le 7; \\ \dfrac{1}{x^2-4}, & \text{если } x > 7. \end{cases}$

7. $F(x) = \begin{cases} x^2, & \text{если } 0 \le x \le 3; \\ 4, & \text{если } x > 3 \text{ или } x < 0. \end{cases}$

8. $F(x) = \begin{cases} x^2+4x+5, & \text{если } x \le 2; \\ \dfrac{1}{x^2+4x+5}, & \text{если } x > 2. \end{cases}$

9. $F(x) = \begin{cases} x^2-x, & \text{если } 0 \le x \le 1; \\ x^2-\sin \pi x^2, & \text{если } x > 1 \text{ или } x < 0. \end{cases}$

10. $F(x) = \begin{cases} -x^2+x-9, & \text{если } x \ge 8; \\ \dfrac{1}{x^4-6}, & \text{если } x < 8. \end{cases}$

11. $F(x) = \begin{cases} 4x^2+2x-19, & \text{если } x \ge -3,5; \\ -\dfrac{2x}{-4x+1}, & \text{если } x < 3,5. \end{cases}$

12. $F(x) = \begin{cases} -x^2+3x+9, & \text{если } x \le 3; \\ \dfrac{x}{x^2+1}, & \text{если } x > 3. \end{cases}$

13. $F(x) = \begin{cases} -3x + 9, & \text{если } x > 3; \\ \dfrac{x^3}{x^2 + 8}, & \text{если } x \le 3. \end{cases}$

14. $F(x) = \begin{cases} -x^3 + 9, & \text{если } x \le 13; \\ -\dfrac{3}{x + 1}, & \text{если } x > 13. \end{cases}$

15. $F(x) = \begin{cases} 45x^2 + 5, & \text{если } x > 3,6; \\ \dfrac{5x}{10x^2 + 1}, & \text{если } x \le 3,6. \end{cases}$

16. $F(x) = \begin{cases} x^4 + 9, & \text{если } x < 3,2; \\ \dfrac{54x^4}{-5x^2 + 7}, & \text{если } x \ge 3,2. \end{cases}$

17. $F(x) = \begin{cases} 1,2x^2 - 3x - 9, & \text{если } x > 3; \\ \dfrac{12,1}{2x^2 + 1}, & \text{если } x \le 3. \end{cases}$

18. $F(x) = \begin{cases} x^2 + 3x + 9, & \text{если } x \le 3; \\ \dfrac{\sin x}{x^2 - 9}, & \text{если } x > 3. \end{cases}$

19. $F(x) = \begin{cases} \cos 2x + 9, & \text{если } > -4; \\ -\dfrac{\cos x}{x - 9}, & \text{если } x \le -4. \end{cases}$

20. $F(x) = \begin{cases} \ln x + 9, & \text{если } x > 0; \\ -\dfrac{x}{x^2 - 7}, & \text{если } x \le 0. \end{cases}$

21. $F(x) = \begin{cases} -x^2 - 1,1x + 9, & \text{если } x \le -3; \\ \dfrac{\ln(x + 3)}{x^2 + 9}, & \text{если } x > -3. \end{cases}$

22. $F(x) = \begin{cases} 9 - x, & \text{если } x > 1,1; \\ \dfrac{\sin 3x}{x^4 + 1}, & \text{если } x < -1,1. \end{cases}$

23. $F(x) = \begin{cases} -x^2, & \text{если } x \geq 7; \\ \dfrac{2^{-x}}{x^2 - 9}, & \text{если } x \leq 7. \end{cases}$

24. $F(x) = \begin{cases} -x^2 - 9, & \text{если } x \geq 13; \\ -\dfrac{1}{x^2 + 9}, & \text{если } x \leq 13. \end{cases}$

25. $F(x) = \begin{cases} 0, & \text{если } x \leq 0; \\ x, & \text{если } 0 < x \leq 1; \\ x^4, & \text{если } x \geq 1. \end{cases}$

6.3. Задачи по теме «Оператор выбора»

1. Написать программу, которая по номеру дня недели (целому числу от 1 до 7) выдает в качестве результата количество уроков в вашем классе в этот день.

2. Написать программу, позволяющую по последней цифре числа определить последнюю цифру его квадрата.

3. Составить программу, которая по заданным году и номеру месяца *m* определяет количество дней в этом месяце.

4. Для каждой введенной цифры (0 — 9) вывести соответствующее ей название на английском языке (0 — *zero*, 1 — *one*, 2 — *two*, ...).

5. Составить программу, которая по данному числу (1—12) выводит название соответствующего ему месяца.

6. Составить программу, позволяющую получить словесное описание школьных отметок (1 — «плохо», 2 — «неудовлетворительно», 3 — «удовлетворительно», 4 — «хорошо», 5 — «отлично»).

7. Пусть элементами круга являются радиус (первый элемент), диаметр (второй элемент) и длина окружности (третий элемент). Составить программу, которая по номеру элемента запрашивала бы его соответствующее значение и вычисляла бы площадь круга.

8. Пусть элементами прямоугольного равнобедренного треугольника являются:

1) катет *a*;

2) гипотенуза b;

3) высота h, опущенная из вершины прямого угла на гипотенузу;

4) площадь S.

Составить программу, которая по заданному номеру и значению соответствующего элемента вычисляла бы значение всех остальных элементов треугольника.

9. Написать программу, которая по номеру месяца выдает название следующего за ним месяца (при $m = 1$ получаем февраль, 4 — май и т.д.).

10. Написать программу, которая бы по введенному номеру времени года (1 — зима, 2 — весна, 3 — лето, 4 — осень) выдавала соответствующие этому времени года месяцы, количество дней в каждом из месяцев.

11. В старояпонском календаре был принят 12-летний цикл. Годы внутри цикла носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Написать программу, которая вводит номер некоторого года и печатает его название по старояпонскому календарю.

(*Справка*: 1996 г. — год Крысы — начало очередного цикла.)

12. Для целого числа k от 1 до 99 напечатать фразу «Мне k лет», учитывая при этом, что при некоторых значениях k слово «лет» надо заменить на слово «год» или «года». Например, 11 лет, 22 года, 51 год.

13. Написать программу, которая бы по введенному номеру единицы измерения (1 — дециметр, 2 — километр, 3 — метр, 4 — миллиметр, 5 — сантиметр) и длине отрезка L выдавала бы соответствующее значение длины отрезка в метрах.

14. Написать программу, которая по вводимому числу от 1 до 11 (номеру класса) выдает соответствующее сообщение «Привет, k-классник». Например, если $k = 1$, «Привет, первоклассник»; если $k = 4$, «Привет, четвероклассник».

15. Написать программу, которая по введенному числу от 1 до 12 (номеру месяца) выдает все приходящиеся на этот месяц праздничные дни (например, если введено число 1, то должно получиться 1 января — Новый год, 7 января — Рождество).

16. Дано натуральное число N. Если оно делится на 4, вывести на экран ответ $N = 4k$ (где k — соответствующее частное); если остаток от деления на 4 равен 1 — $N = 4k + 1$; если остаток от деления на 4 равен 2 — $N = 4k + 2$; если остаток от деления на 4 равен 3 — $N = 4k + 3$. Например, $12 = 4 \cdot 3$, $22 = 4 \cdot 5 + 2$.

17. Имеется пронумерованный список деталей: 1) шуруп, 2) гайка, 3) винт, 4) гвоздь, 5) болт. Составить программу, которая по номеру детали выводит на экран ее название.

18. Составить программу, позволяющую по последней цифре данного числа определить последнюю цифру куба этого числа.

19. Составить программу, которая для любого натурального числа печатает количество цифр в записи этого числа.

20. Даны два действительных положительных числа x и y. Арифметические действия над числами пронумерованы (1 — сложение, 2 — вычитание, 3 — умножение, 4 — деление). Составить программу, которая по введенному номеру выполняет то или иное действие над числами.

21. Написать программу, которая бы по введенному номеру единицы измерения (1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна, 5 — центнер) и массе M выдавала бы соответствующее значение массы в килограммах.

22. Пусть элементами равностороннего треугольника являются:

1) сторона a;
2) площадь S;
3) высота h;
4) радиус вписанной окружности r;
5) радиус описанной окружности R.

Составить программу, которая по заданному номеру и значению соответствующего элемента вычисляла бы значение всех остальных элементов треугольника.

23. Составить программу для определения подходящего возраста кандидатуры для вступления в брак, используя следующее соображение: возраст девушки равен половине возраста мужчины плюс 7, возраст мужчины определяется соответственно как удвоенный возраст девушки минус 14.

24. Найти произведение цифр заданного k-значного числа.

25. Напишите программу, которая читает натуральное число в десятичном представлении, а на выходе выдает это же число в десятичном представлении и на естественном языке.

Например,

```
7 семь
204 двести четыре
52 пятьдесят два
```

26. Вычислить номер дня в невисокосном году по заданным числу и месяцу.

6.4. Задачи по теме «Циклы»

6.4.1. Цикл с параметром

1. Имеется серия измерений элементов треугольника. Группы элементов пронумерованы. В серии в произвольном порядке могут встречаться такие группы элементов треугольника:

1) основание и высота;
2) две стороны и угол между ними (угол задан в радианах);
3) три стороны.

Разработать программу, которая запрашивает номер группы элементов, вводит соответствующие элементы и вычисляет

площадь треугольника. Вычисления прекратить, если в качестве номера группы введен 0.

2. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?

3. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько амеб будет через 3, 6, 9, 12, ..., 24 часа.

4. Около стены наклонно стоит палка длиной x м. Один ее конец находится на расстоянии y м от стены. Определить значение угла α между палкой и полом для значений $x = k$ м и y, изменяющегося от 2 до 3 м с шагом h м.

5. У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей (указать все сочетания)?

6. Составить алгоритм решения задачи: сколько можно купить быков, коров и телят, платя за быка 10 руб., за корову — 5 руб., а за теленка — 0,5 руб., если на 100 руб. надо купить 100 голов скота?

7. Доказать (путем перебора возможных значений), что для любых величин A, B, C типа `Boolean` следующие пары логических выражений имеют одинаковые значения (эквивалентны):

а) `A OR B` и `B OR A;`
б) `A AND B` и `B AND A;`
в) `(A OR B) OR C` и `A OR C;`
г) `(A AND B) AND C` и `A AND (B AND C);`
д) `A AND (A OR B)` и `A;`
е) `A OR (A AND B)` и `A;`
ж) `A AND (B OR C)` и `(A AND B) OR (A AND C);`
з) `A OR (B AND C)` и `(A OR B) AND (A OR C).`

8. Составить программу для проверки утверждения: «Результатами вычислений по формуле $x^2 + x + 17$ при $0 \le x \le 15$ являются простые числа». Все результаты вывести на экран.

9. Составить программу для проверки утверждения: «Результатами вычислений по формуле $x^2 + x + 41$ при $0 \le x \le 40$ являются простые числа». Все результаты вывести на экран.

10. Составить программу-генератор простых чисел, в основу положить формулу $2x^2 + 29$ при $0 \le x \le 28$.

11. Составить программу-генератор простых чисел, в основу положить формулу $\dfrac{2^{2x+1}+1}{3}$ при $1 \le x \le 36$.

12. Составить программу-генератор чисел Пифагора a, b, c ($c^2 = a^2 + b^2$). В основу положить формулы: $a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$ (m, n — натуральные, $1 < m < k$, $1 < n < k$, k — данное число). Результат вывести на экран в виде таблицы из пяти столбцов: m, n, a, b, c.

13. Покупатель должен заплатить в кассу S руб. У него имеются купюры по 1, 5, 10, 50, 100, 500, 1000 и 10 000 руб. Сколько купюр разного достоинства отдаст покупатель, если он начинает платить с самых крупных купюр?

14. Ежемесячная стипендия студента составляет A руб., а расходы на проживание превышают стипендию и составляют B руб. в месяц. Рост цен ежемесячно увеличивает расходы на 3%. Составьте программу расчета суммы денег, которую необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию.

15. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в десятичной системе счисления.

16. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в шестнадцатеричной системе счисления.

17. Найти сумму всех n-значных чисел ($1 \le n \le 4$).

18. Найти сумму всех n-значных чисел, кратных k ($1 \le n \le 4$).

19. Показать, что для всех $n = 1, 2, 3, N$

$$(1^5 + 2^5 + \ldots + n^5) + (1^7 + 2^7 + \ldots + n^7) = 2(1 + 2 + \ldots + n)^4.$$

20. Заменить буквы цифрами так, чтобы соотношение оказалось верным (одинаковым буквам соответствуют одинаковые цифры, разным — разные):

$$\textbf{ХРУСТ} \cdot \textbf{ГРОХОТ} = \textbf{РРРРРРРРРРР}.$$

21. Составить программу, которая запрашивает пароль (например, четырехзначное число) до тех пор, пока он не будет правильно введен.

22. Составить программу, которая находит наибольшее значение отношения трехзначного числа к сумме его цифр.

23. Вычислить сумму кодов всех символов, которые в цикле вводятся с клавиатуры до нажатия на клавишу Esc.

24. Вычислить количество точек с целочисленными координатами, находящихся в круге радиуса R ($R>0$).

25. Напечатать в возрастающем порядке все трехзначные числа, в десятичной записи которых нет одинаковых цифр (операции деления и нахождения остатка от деления не использовать).

26. Вывести на дисплей календарь на текущий год.

27. Составить алгоритм решения ребуса **РАДАР** = **(Р + А + Д)**^{4} (различные буквы обозначают различные цифры, старшая — не 0).

28. Составить алгоритм решения ребуса **МУХА + МУХА + + МУХА = СЛОН** (различные буквы обозначают различные цифры, старшая — не 0).

29. Составить алгоритм решения ребуса **ДРУГ — ГУРД** = 2727 (различные буквы обозначают различные цифры, старшая — не 0).

30. Составить алгоритм решения ребуса **КОТ + КОТ = ТОК** (различные буквы обозначают различные цифры, старшая — не 0).

6.4.2. Ряды

1. Дано натуральное число N. Вычислить

$$S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \cdots + (-1)^n \cdot \frac{1}{2^n}.$$

2. Дано натуральное число N. Вычислить:

$$S = \frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \cdots + \frac{1}{\sin 1 + \sin 2 + \cdots + \sin N}.$$

3. Дано натуральное число N. Вычислить произведение первых N сомножителей

$$P = \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdots \frac{2N}{2N+1}.$$

4. Дано натуральное число N. Вычислить

$$\frac{\cos 1}{\sin 1} \cdot \frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} \cdots \frac{\cos 1 + \cos 2 + \ldots + \cos N}{\sin 1 + \sin 2 + \ldots + \sin N}.$$

5. Дано действительное число x. Вычислить

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}.$$

6. Даны натуральное число n и действительное число x. Вычислить

$$S = \sin x + \sin\sin x + \ldots + \underbrace{\sin\sin \ldots \sin x}_{n \text{ раз}}.$$

7. Даны действительное число a и натуральное число n. Вычислить

$$P = a(a+1) \ldots (a+n-1).$$

8. Даны действительное число a и натуральное число n. Вычислить

$$P = a(a-n)(a-2n) \ldots (a-n^2).$$

9. Даны действительное число a и натуральное число n. Вычислить

$$S = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + \ldots + \frac{1}{a^{2n-2}}.$$

10. Дано действительное число x. Вычислить

$$\frac{(x-1)(x-3)(x-7) \ldots (x-63)}{(x-2)(x-4)(x-8) \ldots (x-64)}.$$

11. Вычислить

$$(1+\sin 0,1)(1+\sin 0,2)\ldots(1+\sin 10).$$

12. Даны натуральное число n и действительное число x. Вычислить

$$\sin x+\sin x^2+\ldots+\sin x^n.$$

13. Дано натуральное число n. Вычислить

$$S=1\cdot 2+2\cdot 3\cdot 4+\ldots+n\cdot(n+1)\ldots 2n.$$

14. Дано натуральное число n. Вычислить

$$P=\left(1-\frac{1}{2^2}\right)\left(1-\frac{1}{3^2}\right)\ldots\left(1-\frac{1}{n^2}\right),\ \text{где } n>2.$$

15. Дано натуральное число n. Вычислить

$$P=\left(1-\frac{1}{2}\right)\left(1-\frac{1}{4}\right)\left(1-\frac{1}{6}\right)\ldots\left(1-\frac{1}{2n}\right).$$

16. Дано натуральное число n. Вычислить

$$S=1!+2!+3!+\ldots+n!\ \ (n>1).$$

17. Дано натуральное число n. Вычислить

$$S=\frac{1}{3^2}+\frac{1}{5^2}+\frac{1}{7^2}+\ldots+\frac{1}{(2n+1)^2}.$$

18. Для данного действительного числа x вычислить по схеме Горнера

$$y=x^{10}+2x^9+3x^8+\ldots+10x+11.$$

19. Числа Фибоначчи (f_n) определяются формулами

$$f_0=f_1=1,\ f_n=f_{n-1}+f_{n-2}\ \text{при } n=2,\ 3,\ \ldots$$

Определить f_{40}.

20. Дано натуральное число n. Вычислить $y=1\cdot 3\cdot 5\ldots(2n-1)$.

21. Дано натуральное число n. Вычислить $y=2\cdot 4\cdot 6\ldots(2n)$.

22. Вычислить $y=\cos x+\cos x^2+\cos x^3+\ldots+\cos x^n$.

23. Вычислить $y=\sin 1+\sin 1,1+\sin 1,2+\ldots+\sin 2$.

24. Даны натуральные числа n и k. Вычислить

$$\sqrt{k+\sqrt{2k+\ldots+\sqrt{k(n-1)+\sqrt{kn}}}}.$$

25. Дано натуральное число n. Вычислить

$$\frac{2}{1}+\frac{3}{2}+\frac{4}{3}+\ldots+\frac{n+1}{n}.$$

6.4.3. Вычисление последовательностей

Даны числовой ряд и некоторое число ε. Найти сумму тех членов ряда, модуль которых больше или равен заданному ε. Общий член ряда имеет вид:

1. $a_n = \frac{(-1)^{n-1}}{n^n}$;
2. $a_n = \frac{1}{2^n} + \frac{1}{3^n}$;
3. $a_n = \frac{2n-1}{2^n}$;
4. $a_n = \frac{1}{(3n-2)(3n+1)}$;
5. $a_n = \frac{10^n}{n!}$;
6. $a_n = \frac{n!}{(2n)!}$;
7. $a_n = \frac{n!}{n^n}$;
8. $a_n = \frac{2^n \cdot n!}{n^n}$;
9. $a_n = \frac{3^n \cdot n!}{(2n)!}$;
10. $a_n = \frac{n!}{3n^n}$;
11. $a_n = \frac{n!}{(2^n)!}$;
12. $a_n = \frac{2^n}{(n-1)!}$.

Найти наименьший номер члена последовательности, для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$. Вывести на экран этот номер и все элементы a_i, где $i = 1, 2, ..., n$.

13. $a_n = \operatorname{arctg} a_{n-1} + 1,\ a_1 = 0.$

14. $a_n = 2 + \frac{1}{a_{n-1}},\ a_1 = 2.$

15. $a_n = \frac{1}{2} \operatorname{tg} a_{n-1},\ a_1 = 0{,}5.$

16. $a_n = \frac{1}{(2n)^2}.$

17. $a_n = \frac{1}{2} \cos a_{n-1},\ a_1 = 0{,}5.$

18. $a_n = \frac{2 + a_{n-1}^2}{2a_{n-1}},\ a_1 = 2.$

19. $a_n = \dfrac{a_{n-1} + a_{n-2}}{2}$, $a_1 = 1$, $a_2 = 2$.

20. $a_n = \dfrac{n^{\ln n}}{(\ln n)^n}$.

21. $a_n = e^{-a_{n-1}}$, $a_1 = 0$.

22. $a_n = \dfrac{x}{2a_{n-1}^2}$, $a_1 = x$.

Найти наименьший номер элемента последовательности, для которого выполняется условие M. Вывести на экран этот номер и все элементы a_i, где $i = 1, 2, \ldots, n$.

23. $a_n = \dfrac{1}{2}\left(a_{n-1} + \dfrac{2}{a_{n-1}}\right)$, $a_1 = 1$, M: $|a_n^2 - 2| < \varepsilon$.

24. $a_n = \dfrac{(-1)^n n}{2^n}$, M: $|a_n| < \varepsilon$.

25. $a_n = \dfrac{(-1)^n 2^n}{n!}$, M: $|a_n| < \varepsilon$.

26. $a_n = \dfrac{1}{(n+1)^2}$, M: $a_n < \varepsilon$.

6.4.4. Табулирование функций

Составить программу для вычисления значений функции $F(x)$ на отрезке $[a, b]$ с шагом h. Результат представить в виде таблицы, первый столбец которой — значения аргумента, второй — соответствующие значения функции.

1. $F(x) = x - \sin x$.
2. $F(x) = \sin^2 x$.
3. $F(x) = 2\cos x - 1$.
4. $F(x) = \operatorname{tg} x$.
5. $F(x) = \operatorname{ctg} x + 1$.
6. $F(x) = \sin x - \cos x$.
7. $F(x) = x \sin x$.
8. $F(x) = \sin\left(\dfrac{1}{x}\right) + 2$.
9. $F(x) = x\cos\left(\dfrac{1}{x}\right) + 2$.
10. $2\sin^2 x + 1$.
11. $F(x) = \sqrt{x}\cos^2 x$.
12. $F(x) = \sin x + \operatorname{tg} x$.

13. $F(x) = \cos x + \operatorname{ctg} x$.

14. $F(x) = 2\operatorname{tg}\frac{x}{2} + 1$.

15. $F(x) = \operatorname{tg}\frac{x}{2} + 2\cos x$.

16. $F(x) = \operatorname{ctg}\frac{x}{3} + \frac{1}{2}\sin x$.

17. $F(x) = \frac{1}{2}\sin\frac{x}{4} + 1$.

18. $F(x) = 2\cos\sqrt{x} + 0{,}5$.

19. $F(x) = x^2 \sin^2 x + 1$.

20. $F(x) = \frac{1}{2}\operatorname{ctg}\frac{x}{4} + 4$.

21. $F(x) = \sin^2 x - \cos 2x$.

22. $F(x) = 7\sin^2 x - \frac{1}{2}\cos x$.

23. $F(x) = -\cos 2x$.

24. $F(x) = \operatorname{tg} 2x - 3$.

25. $F(x) = \sin x + 0{,}5\cos x$.

26. $F(x) = \frac{x}{\cos x}$.

6.5. Задачи по теме «Целочисленная арифметика»

A

1. Дано натуральное число n. Найти сумму первой и последней цифры этого числа.

2. Дано натуральное число n. Переставить местами первую и последнюю цифры этого числа.

3. Даны два натуральных числа m и n ($m \le 9999$, $n \le 9999$). Проверить, есть ли в записи числа m цифры, совпадающие с цифрами в записи числа n.

4. Дано натуральное число n. Проверить, есть ли в записи числа три одинаковых цифры ($n \le 9999$).

5. Дано натуральное число $n \le 99$. Дописать к нему цифру k в конец и в начало.

6. Даны натуральные числа n, k. Проверить, есть ли в записи числа n^k цифра m.

7. Среди всех n-значных чисел указать те, сумма цифр которых равна данному числу k.

8. Заданы три натуральных числа A, B, C, которые обозначают число, месяц и год. Найти порядковый номер даты, начиная отсчет с начала года.

9. Найти наибольшую и наименьшую цифры в записи данного натурального числа.

10. Произведение n первых нечетных чисел равно p. Сколько сомножителей взято? Если введенное число n не является указанным произведением, сообщить об этом.

11. Найти на отрезке $[n, m]$ натуральное число, имеющее наибольшее количество делителей.

12. Задумано некоторое число x ($x < 100$). Известны числа k, m, n — остатки от деления этого числа на 3, 5, 7. Найти x.

13. Игрок А объявляет двузначное число от 01 до 99. Игрок В меняет местами его цифры и прибавляет полученное число к сумме его цифр. Полученный результат он объявляет игроку А. Игрок А проделывает с этим числом ту же процедуру, и так они продолжают поступать поочередно, объявляя числа. От суммы чисел берется остаток от деления на 100, поэтому объявляются лишь двузначные числа. Какие числа может объявить игрок А на начальном шаге, чтобы игрок В в некоторый момент объявил число 00.

14. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до $N - 1$, у которых сумма всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». *Пример.* $N = 44$. Числа: 17, 26, 35.

15. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до $N - 1$, у которых произведение всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». *Пример.* $N = 44$. Числа: 18, 24.

16. Дано натуральное число N. Определить количество 8-значных чисел, у которых сумма цифр в цифровой записи числа меньше, чем N. Если таких чисел нет, то вывести слово «нет».

17. Дано натуральное число N. Определить количество 8-значных чисел, у которых сумма цифр в цифровой записи числа больше, чем N. Если таких чисел нет, то вывести слово «нет».

18. Дано натуральное число N. Найти наибольшее число M ($M > 1$), на которое сумма цифр в цифровой записи числа N делится без остатка. Если такого числа нет, то вывести слово «нет». *Пример.* $N = 12\,345$, $M = 5$. Сумма цифр числа N, равная 15, делится на 5.

19. Дано натуральное число N. Найти наименьшее число M ($N < M < 2N$), которое делится на сумму цифр числа N (без остатка). Если такого числа нет, то вывести слово «нет». *Пример.* $N = 12\,345$, $M = 12\,360$. Число 12 360 делится на число 15 — сумму цифр числа N.

20. Дано натуральное число N ($N > 9$). Определить количество нулей, идущих подряд в младших разрядах данного числа. *Пример.* $N = 1\,020\,000$. Количество нулей равно четырем.

21. Дано натуральное число N ($N > 9$). Определить количество нулей в цифровой записи числа, кроме нулей в младших разрядах. *Пример.* $N = 10\,025\,000$. Количество нулей равно двум.

22. Дано натуральное число N ($N > 9$). Определить сумму цифр в первой половине числа (старшие разряды). *Пример.* $N = 12\,345\,678$. Сумма составляет $1 + 2 + 3 + 4 = 10$.

23. Дано натуральное число N ($N > 9$). Определить сумму цифр во второй половине числа (младшие разряды). *Пример.* $N = 12\,345\,678$. Сумма составляет $5 + 6 + 7 + 8 = 26$.

24. Дано натуральное число N. Если число содержит 3 цифры, то получить новое число M, которое образуется путем перестановки первой и последней цифр данного числа. Если количество цифр не 3, то $M = N$. *Пример.* $N = 123$, $M = 321$.

25. Дано натуральное число N. Если число содержит 5 цифр, то получить новое число M, которое образуется путем исключения средней цифры исходного числа. Если количество цифр не 5, то $M = N$. *Пример.* $N = 12345$, $M = 1245$.

26. Женщина шла на базар продавать яйца. Ее случайно сбил с ног всадник, в результате чего все яйца разбились. Всадник предложил оплатить убытки и спросил, сколько у нее было яиц. Женщина сказала, что точного числа не помнит, но когда она брала яйца парами, то оставалось одно яйцо. Одно яйцо оставалось также, когда она брала по 3, 4, 5 и 6 яиц, но когда она брала по 7 штук, то в остатке ничего не было. Какое минимальное число яиц могло быть в корзине?

В

1. Дано натуральное число n. Проверить, будут ли все цифры числа различными.

2. Найти все целые корни уравнения $ax^3 + bx^2 + cx + d = 0$, где a, b, c и d — заданные целые числа, причем $a \neq 0$ и $d \neq 0$. *Замечание*: целыми корнями могут быть только положительные и отрицательные делители коэффициента d.

3. Дано натуральное число n. Поменять порядок следования цифр в этом числе на обратный или сообщить, что это невозможно в силу переполнения.

4. Найти все делители натурального числа n.

5. Натуральное число M называется совершенным, если оно равно сумме всех своих делителей, включая 1, но исключая себя. Напечатать все совершенные числа меньшие заданного числа N.

6. Натуральные числа a, b, c называются числами Пифагора, если выполняется условие $a^2 + b^2 = c^2$. Напечатать все числа Пифагора меньшие N.

7. Дано натуральное число n. Среди чисел 1, ..., n найти такие, запись которых совпадает с последними цифрами записи их квадратов (например, $6^2 = 36$, $25^2 = 625$).

8. Составить программу, которая по номеру дня в году выводит число и месяц в общепринятой форме (например, 33-й день года — 2 февраля).

9. Долгожитель (возраст не менее 100 лет) обнаружил однажды, что если к сумме квадратов цифр его возраста прибавить число дня его рождения, то как раз получится его возраст. Сколько лет долгожителю?

10. Дано целое $n > 2$. Напечатать все простые числа из диапазона $[2, n]$.

11. Найти наименьшее натуральное число n, представимое двумя различными способами в виде суммы кубов двух натуральных чисел.

12. Даны натуральные числа n, m. Найти все натуральные числа меньшие n, квадрат суммы цифр которых равен m.

13. На отрезке $[2, n]$ определить число с максимальной суммой делителей.

14. Даны натуральные числа p и q. Получить все делители числа q, взаимно простые с p.

15. Для заданных натуральных n и k определить, равно ли число n сумме k-х степеней своих цифр.

16. Найти все n-значные числа, сумма квадратов цифр которых кратна M.

17. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

18. Задано натуральное число n. Найти количество натуральных чисел, не превышающих n и не делящихся ни на одно из чисел 2, 3, 5.

19. Пусть f_n — n-й член последовательности, определяемой следующим образом:

$$f_n = -f_{n-1} - 2f_{n-2},\ f_1 = 1,\ f_2 = -1.$$

Покажите, что $2^{n+1} - 7f_{n-1}^2$ есть полный квадрат.

20. Последовательность Хэмминга образуют натуральные числа, не имеющие других простых делителей, кроме 2, 3 и 5. Найти:

а) первые N элементов этой последовательности;

б) сумму первых N элементов;

в) N-й элемент;

г) первый элемент больший данного числа M, а также номер этого элемента в последовательности;

д) сумму всех элементов с номера N по номер M.

21. На отрезке $[2, n]$ найти все натуральные числа, сумма цифр которых при умножении числа на a не изменится.

22. Составить программу удаления из десятичной записи числа N единиц, сохранив порядок следования оставшихся цифр. Сформировать и напечатать полученное число.

23. Школа и дом Петра находятся на одной стороне улицы. Однажды по дороге в школу он стал складывать номера домов, мимо которых проходил на своей стороне улицы, начиная с номера своего дома. Когда сумма номеров оказалась равной 99, Петр перешел через поперечную улицу. После этого он начал заново складывать номера домов, мимо которых проходил, и при сумме 117 перешел через еще одну поперечную улицу. Петр и в следующем

квартале складывал номера домов. Сумма номеров домов третьего квартала оказалась равной 235, включая номер дома школы. Каков номер дома Петра? Каков номер дома школы?

24. Дано натуральное число N. Определить количество цифр в цифровой записи данного числа, которые имеют наименьшее значение. *Пример.* $N = 4548$. Количество цифр с наименьшим значением равно двум (две цифры 4).

25. Дано натуральное число N. Определить количество цифр в цифровой записи данного числа, которые имеют наибольшее значение. *Пример.* $N = 1808$. Количество цифр с наибольшим значением равно двум (две цифры 8).

26. Дано натуральное число N. Получить новое число M, которое образуется из числа N путем замены последней цифры на значение наименьшей цифры в записи числа N. *Пример.* $N = 128\,452$, $M = 129\,451$.

27. Дано натуральное число N. Получить новое число M, которое образуется из числа N путем замены последней цифры на значение наибольшей цифры в записи числа N. *Пример.* $N = 128\,452$, $M = 128\,458$.

28. Определить количество M-значных натуральных чисел, у которых сумма цифр, стоящих в нечетных разрядах, равна N ($1 \le N \le 30$, $0 < M < 5$).

29. Вычислить сумму тех чисел из заданного отрезка $[a, b]$ (a, b — натуральные), в запись которых входит цифра k.

С

1. Дано натуральное k. Напечатать k-ю цифру последовательности 12345678910111213..., в которой выписаны подряд все натуральные числа.

2. Дано натуральное k. Напечатать k-ю цифру последовательности 149162536..., в которой выписаны подряд квадраты всех натуральных чисел.

3. Составить программу перевода натурального числа из десятичной системы счисления в двоичную.

4. Составить программу перевода данного натурального числа n в шестнадцатеричную систему счисления.

5. Дано натуральное число n. Переставить его цифры так, чтобы образовалось максимальное число, записанное теми же цифрами.

6. Дано натуральное число n. Переставить его цифры так, чтобы образовалось наименьшее число, записанное теми же цифрами.

7. Для записи римскими цифрами используются символы I, V, X, L, C, D, M, обозначающие соответственно числа 1, 5, 10, 50, 100, 500, 1000. Составить программу, которая запись любого дан-

ного числа n ($n \le 3999$) арабскими цифрами переводила бы в запись римскими цифрами.

8. Используя все цифры от 1 до 9 по одному разу в различных комбинациях и операции сложения и вычитания, получить в сумме 100.

9. Используя все цифры от 1 до 9 по одному разу и операции сложения и вычитания, получить в сумме 100, при условии, что цифры появляются в возрастающем или убывающем порядке.

Например,

$$123 + 4 - 5 + 67 - 89 = 100,$$
$$9 - 8 + 76 - 5 + 4 + 3 + 21 = 100.$$

10. Палиндромы.

Палиндром — это сочетание символов, которые читаются одинаково в прямом и обратном направлениях. Элементом палиндрома может быть буква (например, КОК, ПОП, А РОЗА УПАЛА НА ЛАПУ АЗОРА), цифра (4884, 121) или слово (STRAP ON — NO PARTS).

10.1. Составить программу, которая определяет, является ли заданное натуральное число палиндромом.

10.2. Найти целые числа, которые при возведении в квадрат дают палиндромы, например $26^2 = 676$.

10.3. Найти целые числа-палиндромы, которые при возведении в квадрат также дают палиндромы ($22^2 = 484$).

10.4. Найти целые числа, которые при возведении в 3, или 4, или 5 степень дают палиндромы, например $11^3 = 1331$.

10.5. Дано натуральное число n. Если это не палиндром, реверсируйте его цифры и сложите исходное число с числом, полученным в результате реверсирования. Если сумма не палиндром, то повторите те же действия и выполняйте их до тех пор, пока не получите палиндром. Ниже приведен пример для исходного числа 78:

$$78+87=165;\ 165+561=726;\ 726+627=1353;\ 1353+3531=4884.$$

11. Целое число можно представить как сумму его частей. Такое представление называется *разбиением*. Например, число 4 можно представить как

$$4;\ 3+1;\ 2+1+1;\ 2+2;\ 1+1+1+1.$$

Обозначим через $P(n)$ количество разбиений числа n; $P(4) = 5$. Напишите программу, которая для данного числа n печатает его разбиения и $P(n)$.

12. Дано натуральное k. Напечатать k-ю цифру последовательности 246810121416182022426283O..., в которой выписаны подряд все натуральные четные числа.

6.6. Задачи по теме «Подпрограммы»

6.6.1. Нерекурсивные процедуры и функции

А

1. Треугольник задан координатами своих вершин. Составить программу для вычисления его площади.

2. Составить программу для нахождения наибольшего общего делителя и наименьшего общего кратного двух натуральных чисел

$$(\text{НОК}(A, B) = \frac{A \cdot B}{\text{НОД}(A, B)}).$$

3. Составить программу для нахождения наибольшего общего делителя четырех натуральных чисел.

4. Составить программу для нахождения наименьшего общего кратного трех натуральных чисел.

5. Написать программу для нахождения суммы большего и меньшего из трех чисел.

6. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.

7. На плоскости заданы своими координатами n точек. Составить программу, определяющую, между какими из пар точек самое большое расстояние. *Указание.* Координаты точек занести в массив.

8. Составить программу, которая в массиве $A[N]$ находит второе по величине число (вывести на печать число, которое меньше максимального элемента массива, но больше всех других элементов).

9. Составить программу, проверяющую, являются ли данные три числа взаимно простыми.

10. Написать программу для вычисления суммы факториалов всех нечетных чисел от 1 до 9.

11. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D — натуральные числа). Составить программу для деления дроби на дробь. Результат должен быть несократимой дробью.

12. Задан массив D. Определить следующие суммы: $D[1]+D[2]+D[3]$; $D[3]+D[4]+D[5]$; $D[4]+D[5]+D[6]$.

Пояснение. Составить подпрограмму для вычисления суммы трех последовательно расположенных элементов массива с номерами от k до m.

13. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D — натуральные числа). Составить программу для умножения дроби на дробь. Результат должен быть несократимой дробью.

14. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D — натуральные числа). Составить программу для вычитания из первой дроби второй. Результат должен быть несократимой дробью.

15. Даны две дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D — натуральные числа). Составить программу для сложения этих дробей. Результат должен быть несократимой дробью.

16. На плоскости заданы своими координатами n точек. Создать массив размером $n(n-1)$, элементами которого являются расстояния от каждой из точек до $n-1$ других.

17. Даны числа X, Y, Z, T — длины сторон четырехугольника. Вычислить его площадь, если угол между сторонами длиной X и Y — прямой.

18. Сформировать массив $X(N)$, N-й член которого определяется формулой $X(N) = \frac{1}{N!}$.

19. Составить программу для вычисления суммы факториалов всех четных чисел от m до n.

20. Заменить отрицательные элементы линейного массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество произведенных замен.

21. Дан массив $A(N)$. Сформировать массив $B(M)$, элементами которого являются большие из двух рядом стоящих в массиве A чисел. (Например, массив A состоит из элементов 1; 3; 5; −2; 0; 4; 0. Элементами массива B будут 3; 5; 4).

22. Дан массив $A(N)$ (N — четное). Сформировать массив $B(M)$, элементами которого являются средние арифметические соседних пар рядом стоящих в массиве A чисел. (Например, массив A состоит из элементов 1; 3; 5; −2; 0; 4; 0; 3. Элементами массива B будут 2; 1,5; 2; 1,5).

23. Дано простое число. Составить функцию, которая будет находить следующее за ним простое число.

24. Составить функцию для нахождения наименьшего нечетного натурального делителя k ($k \neq 1$) любого заданного натурального числа n.

В

1. Дано натуральное число N. Составить программу для формирования массива, элементами которого являются цифры числа N.

2. Составить программу, определяющую, в каком из данных двух чисел больше цифр.

3. Заменить данное натуральное число на число, которое получается из исходного записью его цифр в обратном порядке (например, дано число 156, нужно получить 651).

4. Даны натуральные числа K и N. Составить программу формирования массива A, элементами которого являются числа, сумма цифр которых равна K и которые не больше N.

5. Даны три квадратных матрицы A, B, C n-го порядка. Вывести на печать ту из них, норма которой наименьшая. *Пояснение.* Нормой матрицы назовем максимум из абсолютных величин ее элементов.

6. Два натуральных числа называются «дружественными», если каждое из них равно сумме всех делителей (кроме его самого) другого числа (например, числа 220 и 284). Найти все пары «дружественных чисел», которые не больше данного числа N.

7. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка $[n, 2n]$, где n — заданное натуральное число больше 2.

8. Написать программу вычисления суммы

$$\frac{p}{q} = 1 - \frac{1}{2} + \frac{1}{3} - \cdots + \frac{(-1)^{n+1}}{n}$$

для заданного числа n. Дробь $\frac{p}{q}$ должна быть несократимой (p, q — натуральные).

9. Написать программу вычисления суммы

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

для заданного числа n. Результат представить в виде несократимой дроби $\frac{p}{q}$ (p, q — натуральные).

10. Натуральное число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n, равна самому числу. Найти все числа Армстронга от 1 до k.

11. Написать программу, которая находит и выводит на печать все четырехзначные числа вида $\overline{abcd}$, для которых выполняется:

а) a, b, c, d — разные цифры;

б) $\overline{ab} - \overline{cd} = a + b + c + d$.

12. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

13. Найти все натуральные n-значные числа, цифры в которых образуют строго возрастающую последовательность (например, 1234, 5789).

14. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

15. Составить программу для нахождения чисел из интервала $[M, N]$, имеющих наибольшее количество делителей.

16. Для последовательности $a_1 = 1$, $a_{n+1} = a_n + \frac{1}{1+a_n}$ составить программу печати k-го члена в виде обыкновенной несократимой дроби. Например, $a_2 = \frac{3}{2}$, $a_3 = \frac{19}{10}$.

17. Дано натуральное число n. Выяснить, можно ли представить его в виде произведения трех последовательных натуральных чисел.

18. Имеется часть катушки с автобусными билетами. Номер билета шестизначный. Составить программу, определяющую количество счастливых билетов на катушке, если меньший номер билета — N, больший — M (билет является счастливым, если сумма первых трех его цифр равна сумме последних трех).

19. Написать программу, определяющую сумму n-значных чисел, содержащих только нечетные цифры. Определить также, сколько четных цифр в найденной сумме.

20. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т.д. Сколько таких действий надо произвести, чтобы получился нуль?

21. Составить программу для разложения данного натурального числа на простые множители. Например, $200 = 2^3 \cdot 5^2$.

22. Дано натуральное число n. Найти все числа Мерсена меньшие n. (Простое число называется числом Мерсена, если оно может быть представлено в виде $2^p - 1$, где p — тоже простое число. Например, $31 = 2^5 - 1$ — число Мерсена.)

23. Дано четное число $n > 2$. Проверить для него гипотезу Гольдбаха: каждое четное n представляется в виде суммы двух простых чисел.

24. Даны натуральные числа n и k, $n > 1$. Напечатать k десятичных знаков числа $1/n$. Программа должна использовать только целые переменные.

25. Дано натуральное число $n > 1$. Определить длину периода десятичной записи дроби $1/n$.

26. Функция f с натуральными аргументами и значениями определена так: $f(0) = 0$, $f(1) = 1$, $f(2n) = f(n)$, $f(2n+1) = f(n) + f(n+1)$. Составить программу вычисления $f(n)$ по заданному n.

27. На отрезке $[100, N]$ $(2^{10} < N < 2^{31})$ найти количество чисел, составленных из цифр a, b, c.

6.6.2. Рекурсивные процедуры и функции

Решить следующие задачи, используя рекурсивную подпрограмму.

1. Найти сумму цифр заданного натурального числа.

2. Подсчитать количество цифр в заданном натуральном числе.

3. Описать функцию $C(m, n)$, где $0 \le m \le n$, для вычисления биномиального коэффициента C_n^m по следующей формуле:

$$C_n^0 = C_n^n = 1;\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1} \text{ при } 0 < m < n.$$

4. Описать рекурсивную функцию `Root(a, b, ε)`, которая методом деления отрезка пополам находит с точностью ε корень уравнения $f(x)=0$ на отрезке $[a, b]$ (считать, что $\varepsilon > 0$, $a < b$, $f(a) \cdot f(b) < 0$ и $f(x)$ — непрерывная и монотонная на отрезке $[a, b]$ функция).

5. Описать функцию `min(X)` для определения минимального элемента линейного массива X, введя вспомогательную рекурсивную функцию `min1(k)`, находящую минимум среди последних элементов массива X, начиная с k-го.

6. Описать рекурсивную логическую функцию `Simm(S,I,J)`, проверяющую, является ли симметричной часть строки S, начинающаяся i-м и заканчивающаяся j-м ее элементами.

7. Составить программу для вычисления наибольшего общего делителя двух натуральных чисел.

8. Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке. Например, для числа 1234 получаем результат 4321.

9. Составить программу для перевода данного натурального числа в p-ичную систему счисления ($2 \le p \le 9$).

10. Дана символьная строка, представляющая собой запись натурального числа в p-ичной системе счисления ($2 \le p \le 9$). Составить программу для перевода этого числа в десятичную систему счисления.

11. Составить программу для вычисления суммы: $1! + 2! + 3! + \ldots + n!$ ($n \le 15$).

Примечание. Тип результата значения функции — `LongInt`.

12. Составить программу для вычисления суммы: $2! + 4! + 6! + \ldots + n!$ ($n \le 16$, n — четное).

Примечание. Тип результата значения функции — `LongInt`.

13*. Дано n различных натуральных чисел. Напечатать все перестановки этих чисел.

14. Логическая функция возвращает `True`, если ее аргумент — простое число.

15. Описать функцию, которая удаляет из строки все лишние пробелы. Пробелы считаются лишними, если их подряд идет более двух, если они стоят в конце строки после последней точки, если стоят после открывающегося парного знака препинания.

6.7. Задачи по теме «Одномерные массивы»

А

1. В массив $A[N]$ занесены натуральные числа. Найти сумму тех элементов, которые кратны данному K.

2. В целочисленной последовательности есть нулевые элементы. Создать массив из номеров этих элементов.

3. Дана последовательность целых чисел a_1, a_2, ..., a_n. Выяснить, какое число встречается раньше — положительное или отрицательное.

4. Дана последовательность действительных чисел a_1, a_2, ..., a_n. Выяснить, будет ли она возрастающей.

5. Дана последовательность натуральных чисел a_1, a_2, ..., a_n. Создать массив из четных чисел этой последовательности. Если таких чисел нет, то вывести сообщение об этом факте.

6. Дана последовательность чисел a_1, a_2, ..., a_n. Указать наименьшую длину числовой оси, содержащую все эти числа.

7. Дана последовательность действительных чисел a_1, a_2, ..., a_n. Заменить все ее члены, большие данного Z, этим числом. Подсчитать количество замен.

8. Последовательность действительных чисел оканчивается нулем. Найти количество членов этой последовательности.

9. Дан массив действительных чисел, размерность которого N. Подсчитать, сколько в нем отрицательных, положительных и нулевых элементов.

10. Даны действительные числа a_1, a_2, ..., a_n. Поменять местами наибольший и наименьший элементы.

11. Даны целые числа a_1, a_2, ..., a_n. Вывести на печать только те числа, для которых $a_i \geq i$.

12. Даны натуральные числа a_1, a_2, ..., a_n. Указать те из них, у которых остаток от деления на M равен L $(0 \leq L \leq M - 1)$.

13. В заданном одномерном массиве поменять местами соседние элементы, стоящие на четных местах, с элементами, стоящими на нечетных местах.

14. При поступлении в вуз абитуриенты, получившие двойку на первом экзамене, ко второму не допускаются. В массиве $A[n]$ записаны оценки экзаменующихся, полученные на первом экзамене. Подсчитать, сколько человек не допущено ко второму экзамену.

15. Дана последовательность чисел, среди которых имеется один нуль. Вывести на печать все числа до нуля включительно.

16. В одномерном массиве размещены: в первых элементах — значения аргумента, в следующих — соответствующие им значения функции. Напечатать элементы этого массива в виде двух параллельных столбцов (аргумент и значения функции).

17. Пригодность детали оценивается по размеру B, который должен соответствовать интервалу $(A - \delta, A + \delta)$. Определить, имеются ли в партии из N деталей бракованные. Если да, то подсчитать их количество, в противном случае выдать отрицательный ответ.

18. У вас есть доллары. Вы хотите обменять их на рубли. Есть информация о стоимости купли-продажи в банках города. В городе N банков. Составьте программу, определяющую, какой банк выбрать, чтобы выгодно обменять доллары на рубли.

19. Дан целочисленный массив с количеством элементов n. Напечатать те его элементы, индексы которых являются степенями двойки (1, 2, 4, 8, 16, ...).

20. Задана последовательность из N вещественных чисел. Определить, сколько среди них чисел меньших K, равных K и больших K.

21. Задана последовательность N вещественных чисел. Вычислить

$$S_i = \sqrt{\frac{(X_i - M)^2}{N - 1}},$$

где M — среднее арифметическое данной последовательности.

22. Задан массив (`VAR A: ARRAY [1..N] OF '0'..'9';`). Определить, входит ли в него последовательность символов 123, если да, то сколько раз и с каких позиций ($N > 3$).

23. Задан массив действительных чисел. Определить, сколько раз меняется знак в данной последовательности чисел, запомнить номера позиций, в которых происходит смена знака.

24. Задана последовательность N вещественных чисел. Вычислить сумму чисел, порядковые номера которых являются простыми числами.

25. Задана последовательность N вещественных чисел. Вычислить сумму чисел, порядковые номера которых являются числами Фибоначчи.

26. Задана последовательность N вещественных чисел. Вычислить значение выражения $\sqrt[N]{|x_1 \cdot x_2 \ldots x_N|}$.

27. Задана последовательность N целых чисел. Вычислить сумму элементов массива, порядковые номера которых совпадают со значением этого элемента.

28. Заполнить массив из N элементов с начальным значением заданным $A[0] \neq 0$, по принципу `A[I]=A[I DIV 2]+A[I-1]`.

29. Определить количество элементов последовательности натуральных чисел, кратных числу M и заключенных в промежутке от L до N.

30. Определить, сколько процентов от всего количества элементов последовательности целых чисел составляют нечетные элементы.

31. Сформировать массив простых чисел не больших заданного натурального числа N.

32. Сформировать массив простых множителей заданного числа.

В

1. Дан одномерный массив $A[N]$. Найти

$$\max(a_2,\ a_4,\ \ldots,\ a_{2k}) + \min(a_1,\ a_3,\ \ldots,\ a_{2k+1}).$$

2. Дана последовательность действительных чисел $a_1, a_2, \ldots, a_n$. Указать те ее элементы, которые принадлежат отрезку $[c, d]$.

3. Дана последовательность целых положительных чисел. Найти произведение только тех из них, которые больше заданного числа M. Если таких чисел нет, то выдать сообщение об этом.

4. Последовательность $a_1, a_2, \ldots, a_n$ состоит из нулей и единиц. Поставить в начало этой последовательности нули, а затем единицы.

5. Даны действительные числа $a_1, a_2, \ldots, a_n$. Среди них есть положительные и отрицательные. Заменить нулями те числа, величина которых по модулю больше максимального числа

$$\left(|a_i| > \max\{a_1,\ a_2,\ \ldots,\ a_n\}\right).$$

6. Даны действительные числа $a_1, a_2, \ldots, a_n$. Найти

$$\max(a_1 + a_{2n},\ a_2 + a_{2n-1},\ \ldots,\ a_n + a_{n+1}).$$

7. В последовательности действительных чисел $a_1, a_2, \ldots, a_n$ есть только положительные и отрицательные элементы. Вычислить произведение отрицательных элементов P_1 и произведение положительных элементов P_2. Сравнить модуль P_2 с модулем P_1, указать, какое из произведений по модулю больше.

8. Дан массив действительных чисел. Среди них есть равные. Найти его первый максимальный элемент и заменить его нулем.

9. Дана последовательность действительных чисел $a_1 \le a_2 \le \ldots \le a_n$. Вставить в нее действительное число b так, чтобы последовательность осталась неубывающей.

10. Даны целые положительные числа $a_1, a_2, \ldots, a_n$. Найти среди них те, которые являются квадратами некоторого числа m.

11. Дана последовательность целых чисел $a_1, a_2, \ldots, a_n$. Образовать новую последовательность, выбросив из исходной те члены, которые равны $\min(a_1, a_2, \ldots, a_n)$.

12. У прилавка магазина выстроилась очередь из n покупателей. Время обслуживания i-го покупателя равно t_i ($i = 1, \ldots, n$). Определить время C_i пребывания i-го покупателя в очереди.

13. «Суперзамок». Секретный замок для сейфа состоит из 10 расположенных в ряд ячеек, в которые надо вставить игральные кубики. Но дверь открывается только в том случае, когда в любых

трех соседних ячейках сумма точек на передних гранях кубиков равна 10. (Игральный кубик имеет на каждой грани от 1 до 6 точек.) Напишите программу, которая разгадывает код замка при условии, что два кубика уже вставлены в ячейки.

14. В массиве целых чисел с количеством элементов n найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.

15. Каждый солнечный день улитка, сидящая на дереве, поднимается вверх на 2 см, а каждый пасмурный день опускается вниз на 1 см. В начале наблюдения улитка находилась в A см от земли на B-метровом дереве. Имеется 30-элементный массив, содержащий сведения о том, был ли соответствующий день наблюдения пасмурным или солнечным. Написать программу, определяющую местоположение улитки к концу 30-го дня наблюдения.

16. Дан целочисленный массив с количеством элементов n. Сжать массив, выбросив из него каждый второй элемент.

Примечание. Дополнительный массив не использовать.

17. Задан массив, содержащий несколько нулевых элементов. Сжать его, выбросив эти элементы.

18. Задан массив с количеством элементов N. Сформировать два массива: в первый включить элементы исходного массива с четными номерами, а во второй — с нечетными.

19. Дана последовательность целых чисел $a_1, a_2, \ldots, a_n$. Указать пары чисел a_i, a_j, таких, что $a_i + Ea_j = m$.

20. Дана последовательность целых чисел $a_1, a_2, \ldots, a_n$. Наименьший член этой последовательности заменить целой частью среднего арифметического всех членов, остальные члены оставить без изменения. Если в последовательности несколько наименьших членов, то заменить последний по порядку.

21. Даны две последовательности целых чисел $a_1, a_2, \ldots, a_n$ и $b_1, b_2, \ldots, b_n$. Преобразовать последовательность $b_1, b_2, \ldots, b_n$ по следующему правилу: если $a_i \le 0$, то b_i увеличить в 10 раз, в противном случае b_i заменить нулем ($i = 1, 2, \ldots, n$).

22. Дана последовательность действительных чисел $a_1, a_2, \ldots, a_n$. Требуется домножить все члены последовательности $a_1, a_2, \ldots, a_n$ на квадрат ее наименьшего члена, если $a_k \ge 0$, и на квадрат ее наибольшего члена, если $a_k < 0$ ($1 \le k \le n$).

23. Даны координаты n точек на плоскости: $(X_1, Y_1), \ldots, (X_n, Y_n)$ ($n \le 30$). Найти номера пары точек, расстояние между которыми наибольшее (считать, что такая пара единственная).

24. Дана последовательность n различных целых чисел. Найти сумму ее членов, расположенных между максимальным и минимальным значениями (в сумму включить и оба этих числа).

25. Японская радиокомпания провела опрос N радиослушателей по вопросу: «Какое животное вы связываете с Японией и

японцами?». Составить программу получения k наиболее часто встречающихся ответов и их долей (в процентах).

26. Дан массив, состоящий из n натуральных чисел. Образовать новый массив, элементами которого будут элементы исходного, оканчивающиеся на цифру k.

27. Дан массив целых чисел. Найти в этом массиве минимальный элемент m и максимальный элемент M. Получить в порядке возрастания все целые числа из интервала $(m; M)$, которые не входят в данный массив.

28. Даны действительное число x и массив $A[n]$. В массиве найти два члена, среднее арифметическое которых ближе всего к x.

29. Даны две последовательности $a_1, a_2, \ldots, a_n$ и $b_1, b_2, \ldots, b_m$ $(m < n)$. В каждой из них члены различны. Верно ли, что все члены второй последовательности входят в первую последовательность?

30. Напишите программу, входными данными которой является возраст n человек. Программа подсчитывает количество людей, возраст которых находится в интервале 10 лет, а именно:

0—9 лет;
10—19 лет;
20—29 лет и т.д.

Напечатать результаты расчетов в удобочитаемой форме.

31. Дан массив $X[N]$ целых чисел. Не используя других массивов, переставить его элементы в обратном порядке.

32. Коэффициенты многочлена хранятся в массиве $A[N]$ (N — натуральное число, степень многочлена). Вычислить значение этого многочлена в точке x (т.е. $a[N]\cdot x^N + \ldots + a[1]\cdot x + a[0]$). Вычислить значение его производной в той же точке.

33. В массивах $A[K]$ и $B[L]$ хранятся коэффициенты двух многочленов степеней K и L. Поместить в массив $C[M]$ коэффициенты их произведения. (Числа K, L, M — натуральные, $M = K + L$; элемент массива с индексом I содержит коэффициент при x в степени I.)

34. Вывести информацию о наибольшем, наименьшем и наименее удаленном от среднего арифметического членах последовательности вещественных чисел.

35. Задан массив A. Определить значение k, при котором сумма $|A[1]+A[2]+\ldots+A[k] - (A[k+1]+\ldots+A[N])|$ минимальна (т.е. минимален модуль разности сумм элементов в правой и левой части, на которые массив делится этим k).

С

1. В одномерном массиве все отрицательные элементы переместить в начало массива, а остальные — в конец с сохранением порядка следования. Дополнительный массив заводить не разрешается.

2. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д.

Определить минимальный радиус окружности с центром в начале координат, которая содержит все точки.

3. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д.

Определить кольцо с центром в начале координат, которое содержит все точки.

4. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д. (x_i, y_i — целые).

Определить номера точек, которые могут являться вершинами квадрата.

5. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д.

Определить номера точек, которые могут являться вершинами равнобедренного треугольника.

6. Задан целочисленный массив размерности N. Есть ли среди элементов массива простые числа? Если да, то вывести номера этих элементов.

7. Дана последовательность целых чисел. Найти количество различных чисел в этой последовательности.

8. Дан массив из n четырехзначных натуральных чисел. Вывести на экран только те, у которых сумма первых двух цифр равна сумме двух последних.

9. Даны две последовательности целых чисел a_1, a_2, ..., a_n и b_1, b_2, ..., b_n. Все члены последовательностей — различные числа. Найти, сколько членов первой последовательности совпадает с членами второй последовательности.

10. Дан целочисленный массив $A[n]$, среди элементов есть одинаковые. Создать массив из различных элементов $A[n]$.

11. На плоскости n точек заданы своими координатами, и также дана окружность радиуса R с центром в начале координат. Указать множество всех треугольников с вершинами в заданных точках, пересекающихся с окружностью; множество всех треугольников, содержащихся внутри окружности.

12. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д. Найти номера наиболее и наименее удаленных друг от друга точек.

13. В одномерном массиве с четным количеством элементов ($2N$) находятся координаты N точек плоскости. Они располагаются в следующем порядке: x_1, y_1, x_2, y_2, x_3, y_3 и т.д.

Определить три точки, являющиеся вершинами треугольника, для которого разность точек вне его и внутри является минимальной.

14. Некоторое число содержится в каждом из трех целочисленных неубывающих массивов $x[1] \le ... \le x[p]$, $y[1] \le ... \le y[q]$, $z[1] \le ... \le z[r]$. Найти одно из таких чисел. Число действий должно быть порядка $p+q+r$.

15. Выяснить, есть ли одинаковые числа в каждом из трех целочисленных неубывающих массивов $x[1] \le ... \le x[p]$, $y[1] \le ... \le y[q]$, $z[1] \le ... \le z[r]$. Найти одно из таких чисел или сообщить о его отсутствии.

16. Дана целочисленная таблица $A[n]$. Найти наименьшее число K элементов, которые можно выкинуть из данной последовательности, так чтобы осталась возрастающая подпоследовательность.

17. Разделить массив на две части, поместив в первую элементы, большие среднего арифметического их суммы, а во вторую — меньшие (части не сортировать).

Сортировка массивов

1. Заданы два одномерных массива с различным количеством элементов и натуральное число k. Объединить их в один массив, включив второй массив между k-м и $(k+1)$-м элементами первого, при этом не используя дополнительный массив.

2. Даны две последовательности $a_1 \le a_2 \le ... \le a_n$ и $b_1 \le b_2 \le ... \le b_m$. Образовать из них новую последовательность чисел так, чтобы она тоже была неубывающей. *Примечание.* Дополнительный массив не использовать.

3. **Сортировка выбором.** Дана последовательность чисел $a_1, a_2, ..., a_n$. Требуется переставить элементы так, чтобы они были расположены по убыванию. Для этого в массиве, начиная с первого, выбирается наибольший элемент и ставится на первое место, а первый — на место наибольшего. Затем, начиная со второго, эта процедура повторяется. Написать алгоритм сортировки выбором.

4. **Сортировка обменами.** Дана последовательность чисел $a_1, a_2, ..., a_n$. Требуется переставить числа в порядке возрастания. Для этого сравниваются два соседних числа a_i и a_{i+1}. Если $a_i > a_{i+1}$, то делается перестановка. Так продолжается до тех пор, пока все элементы не станут расположены в порядке возрастания. Составить алгоритм сортировки, подсчитывая при этом количества перестановок.

5. **Сортировка вставками.** Дана последовательность чисел $a_1, a_2, ..., a_n$. Требуется переставить числа в порядке возрастания. Делается это следующим образом. Пусть $a_1, a_2, ..., a_i$ — упорядоченная последовательность, т.е. $a_1 \le a_2 \le ... \le a_i$. Берется следующее число a_{i+1} и вставляется в последовательность так, чтобы новая

последовательность была тоже возрастающей. Процесс производится до тех пор, пока все элементы от $i+1$ до n не будут перебраны. *Примечание.* Место помещения очередного элемента в отсортированную часть производить с помощью двоичного поиска. Двоичный поиск оформить в виде отдельной функции.

6. **Сортировка Шелла.** Дан массив n действительных чисел. Требуется упорядочить его по возрастанию. Делается это следующим образом: сравниваются два соседних элемента a_i и a_{i+1}. Если $a_i \le a_{i+1}$, то продвигаются на один элемент вперед. Если $a_i > a_{i+1}$, то производится перестановка и сдвигаются на один элемент назад. Составить алгоритм этой сортировки.

7. Пусть даны две неубывающие последовательности действительных чисел $a_1 \le a_2 \le \ldots \le a_n$ и $b_1 \le b_2 \le \ldots \le b_m$. Требуется указать те места, на которые нужно вставлять элементы последовательности b_1, b_2, ..., b_m в первую последовательность так, чтобы новая последовательность оставалась возрастающей.

8. Даны дроби $\frac{p_1}{q_1}, \frac{p_2}{q_2}, \ldots, \frac{p_n}{q_n}$ (p_i, q_i — натуральные). Составить программу, которая приводит эти дроби к общему знаменателю и упорядочивает их в порядке возрастания.

9. **Алгоритм фон Неймана.** Упорядочить массив a_1, a_2, ..., a_n по неубыванию с помощью алгоритма сортировки слияниями:

1) каждая пара соседних элементов сливается в одну группу из двух элементов (последняя группа может состоять из одного элемента);

2) каждая пара соседних двухэлементных групп сливается в одну четырехэлементную группу и т.д.

При каждом слиянии новая укрупненная группа упорядочивается.

10. **Сортировка подсчетом.** Выходной массив заполняется значениями -1. Затем для каждого элемента определяется его место в выходном массиве путем подсчета количества элементов строго меньших данного. Естественно, что все одинаковые элементы попадают на одну позицию, за которой следует ряд значений -1. После этого оставшиеся в выходном массиве позиции со значением -1 заполняются копией предыдущего значения.

11. **«Хитрая» сортировка.** Из массива путем однократного просмотра выбирается последовательность элементов, расположенных в порядке возрастания, переносится в выходной массив и заменяется во входном на -1. Затем оставшиеся элементы включаются в полученную упорядоченную последовательность методом «погружения», когда очередной элемент путем ряда обменов «погружается» до требуемой позиции в уже упорядоченную часть массива.

6.8. Задачи по теме «Двумерные массивы»

6.8.1. Задачи на формирование массивов

В задачах 1—12 сформировать квадратную матрицу порядка n по заданному образцу:

1. $\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ n & n-1 & n-2 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ n & n-1 & n-2 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & n-1 & n-2 & \dots & 1 \end{pmatrix}$

2. $\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 2 & 0 \\ 0 & 0 & 0 & \cdots & 3 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & n-1 & 0 & \cdots & 0 & 0 & 0 \\ n & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$

(n — четное).

3. $\begin{pmatrix} n & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & n-1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & n-2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 2 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$

4. $\begin{pmatrix} 1\cdot 2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 2\cdot 3 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 3\cdot 4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & (n-1)n & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & n(n+1) \end{pmatrix}$

5. $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 1 \end{pmatrix}$

6. $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 2 & 2 & 2 & \cdots & 2 & 2 & 0 \\ 3 & 3 & 3 & \cdots & 3 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ n-1 & n-1 & 0 & \cdots & 0 & 0 & 0 \\ n & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$

7. $\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 0 & 0 & 1 & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 1 \end{pmatrix}$

8. $\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 1 & 0 & \cdots & 0 & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 1 & 0 & \cdots & 0 & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$

9. $\begin{pmatrix} n & 0 & 0 & \dots & 0 & 0 & 0 \\ n-1 & n & 0 & \dots & 0 & 0 & 0 \\ n-2 & n-1 & n & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2 & 3 & 4 & \dots & n-1 & n & 0 \\ 1 & 2 & 3 & \dots & n-2 & n-1 & n \end{pmatrix}$

10. $\begin{pmatrix} 1 & 2 & 3 & \dots & n-2 & n-1 & n \\ 2 & 3 & 4 & \dots & n-1 & n & 0 \\ 3 & 4 & 5 & \dots & n & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ n-1 & n & 0 & \dots & 0 & 0 & 0 \\ n & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix}$

11. $\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & n \\ 0 & 2 & 0 & \dots & 0 & n-1 & 0 \\ 0 & 0 & 3 & \dots & n-2 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 2 & 0 & \dots & 0 & n-1 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 & n \end{pmatrix}$

12. $\begin{pmatrix} 1 & 2 & 3 & \dots & n-2 & n-1 & n \\ 2 & 1 & 2 & \dots & n-2 & n-2 & n-1 \\ 3 & 2 & 1 & \dots & n-4 & n-3 & n-2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ n-1 & n-2 & n-3 & \dots & 2 & 1 & 2 \\ n & n-1 & n-2 & \dots & 3 & 2 & 1 \end{pmatrix}$

13. Построить квадратную матрицу порядка $2n$:

$$\begin{pmatrix} \overbrace{1 & 1 & \dots & 1}^{n} & \overbrace{2 & 2 & \dots & 2}^{n} \\ 1 & 1 & \dots & 1 & 2 & 2 & \dots & 2 \\ \vdots & \vdots & \ddots & \dots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 & 2 & 2 & 2 & 2 \\ 3 & 3 & \dots & 3 & 4 & 4 & \dots & 4 \\ 3 & 3 & \dots & 3 & 4 & 4 & \dots & 4 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \underbrace{3 & 3 & \dots & 3}_{n} & \underbrace{4 & 4 & \dots & 4}_{n} \end{pmatrix}$$

14. Дано действительное число x. Получить квадратную матрицу порядка $n + 1$:

$$\begin{pmatrix} 1 & x & x^2 & \dots & x^{n-2} & x^{n-1} & x^n \\ x & 0 & 0 & \dots & 0 & 0 & x^{n-1} \\ x^2 & 0 & 0 & \dots & 0 & 0 & x^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x^{n-1} & 0 & 0 & \dots & 0 & 0 & x \\ x^n & x^{n-1} & x^{n-2} & \dots & x^2 & x & 1 \end{pmatrix}$$

15. Даны действительные числа $a_1, a_2, \dots, a_n$. Получить квадратную матрицу порядка n:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_{n-2} & a_{n-1} & a_n \\ a_2 & a_3 & a_4 & \dots & a_{n-1} & a_{n-2} & a_1 \\ a_3 & a_4 & a_5 & \dots & a_n & a_1 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-1} & a_n & a_1 & \dots & a_{n-4} & a_{n-3} & a_{n-2} \\ a_n & a_1 & a_2 & \dots & a_{n-3} & a_{n-2} & a_{n-1} \end{pmatrix}$$

16. Получить матрицу:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & 9 & 10 \\ 0 & 1 & 2 & \dots & 8 & 9 \\ 0 & 0 & 1 & \dots & 7 & 8 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

17. Получить матрицу:

$$\begin{pmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \dots & 1 & 0 \\ 1 & 0 & \dots & 0 & 1 \end{pmatrix}$$

18. Составить программу, которая заполняет квадратную матрицу порядка n натуральными числами 1, 2, 3, ..., n^2, записывая их в нее «по спирали».

Например, для $n = 5$ получаем следующую матрицу:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 16 & 17 & 18 & 19 & 6 \\ 15 & 24 & 25 & 20 & 7 \\ 14 & 23 & 22 & 21 & 8 \\ 13 & 12 & 11 & 10 & 9 \end{pmatrix}$$

19. Дана действительная квадратная матрица порядка $2n$. Получить новую матрицу, переставляя ее блоки размера $n \times n$ по часовой стрелке, начиная с блока в левом верхнем углу.

20. Дана действительная квадратная матрица порядка $2n$. Получить новую матрицу, переставляя ее блоки размера $n \times n$ крест-накрест.

21. Дан линейный массив $x_1, x_2, \ldots, x_{n-1}, x_n$. Получить действительную квадратную матрицу порядка n:

$$\begin{pmatrix} x_1 & x_2 & \ldots & x_{n-1} & x_n \\ x_1^2 & x_2^2 & \ldots & x_{n-1}^2 & x_n^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^n & x_2^n & \ldots & x_{n-1}^n & x_n^n \end{pmatrix}$$

22. Дан линейный массив $x_1, x_2, \ldots, x_{n-1}, x_n$. Получить действительную квадратную матрицу порядка n:

$$\begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ x_1 & x_2 & \ldots & x_{n-1} & x_n \\ x_1^2 & x_2^2 & \ldots & x_{n-1}^2 & x_n^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \ldots & x_{n-1}^{n-1} & x_n^{n-1} \end{pmatrix}$$

23. Получить квадратную матрицу порядка n:

$$\begin{pmatrix} 1 & 2 & \ldots & n-1 & n \\ n+1 & n+2 & \ldots & 2n-1 & 2n \\ 2n+1 & 2n+2 & \ldots & 3n-1 & 3n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (n-1)n+1 & (n-1)n+2 & \ldots & n^2-1 & n^2 \end{pmatrix}$$

24. Получить квадратную матрицу порядка n:

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & n-1 \end{pmatrix}$$

25. Магическим квадратом порядка n называется квадратная матрица размера $n \times n$, составленная из чисел 1, 2, ..., n^2 так, что суммы по каждому столбцу, каждой строке и каждой из двух больших диагоналей равны между собой. Построить такой квадрат.

Пример магического квадрата порядка 3:

$$\begin{matrix} 6 & 1 & 8 \\ 7 & 5 & 3 \\ 2 & 9 & 4 \end{matrix}$$

26. Сформировать квадратную матрицу порядка N по правилу $A[I,J] = \sin\left(\frac{I^2 - J^2}{N}\right)$ и подсчитать количество положительных элементов в ней.

6.8.2. Операции с элементами массивов

1. Вычислить сумму и число положительных элементов матрицы $A[N, N]$, находящихся над главной диагональю.

2. Дана матрица A размером $n \times m$. Определить k — количество особых элементов массива A, считая его элемент особым, если он больше суммы остальных элементов его столбца.

3. Задана квадратная матрица. Поменять местами строку с максимальным элементом на главной диагонали со строкой с заданным номером m.

4. Дана матрица $B[N, M]$. Найти в каждой строке матрицы максимальный и минимальный элементы и поменять их местами с первым и последним элементом строки соответственно.

5. Дана целая квадратная матрица n-го порядка. Определить, является ли она магическим квадратом, т. е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

6. Элемент матрицы назовем *седловой точкой*, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот, является наибольшим в своей строке и наименьшим в своем столбце. Для заданной целой матрицы размером $n \times m$ напечатать индексы всех ее седловых точек.

7. Дана матрица размером $n \times m$. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (или один из них) оказался в верхнем левом углу.

8. Определить, является ли заданная целая квадратная матрица n-го порядка симметричной (относительно главной диагонали).

9. Дана целочисленная квадратная матрица. Найти в каждой строке наибольший элемент и поменять его местами с элементом главной диагонали.

10. Упорядочить по возрастанию элементы каждой строки матрицы размером $n \times m$.

11. Задана матрица размером $n \times m$. Найти максимальный по модулю элемент матрицы. Переставить строки и столбцы матрицы таким образом, чтобы максимальный по модулю элемент был расположен на пересечении k-й строки и k-го столбца.

12. Дана квадратная матрица $A[N, N]$. Записать на место отрицательных элементов матрицы нули, а на место положительных — единицы. Вывести на печать нижнюю треугольную матрицу в общепринятом виде.

13. Дана действительная матрица размером $n \times m$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.

14. Дана действительная квадратная матрица порядка N (N — нечетное), все элементы которой различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

15. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, суммируя элементы одномерного массива. Преобразовать исходную матрицу по правилу: четные строки разделить на полученное значение, нечетные оставить без изменения.

16. Задана квадратная матрица. Получить транспонированную матрицу.

17. Квадратная матрица, симметричная относительно главной диагонали, задана верхним треугольником в виде одномерного массива. Восстановить исходную матрицу и напечатать по строкам.

18. Заданы матрица порядка n и число k. Разделить элементы k-й строки на диагональный элемент, расположенный в этой строке.

19. Для целочисленной квадратной матрицы найти число элементов, кратных k, и наибольший из них.

20. Найти наибольший и наименьший элементы прямоугольной матрицы и поменять их местами.

21. Дана прямоугольная матрица. Найти строку с наибольшей и наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.

22. В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единствен.

23. В данной действительной квадратной матрице порядка n найти наибольший по модулю элемент. Получить квадратную матрицу порядка $n - 1$ путем отбрасывания в исходной матрице строки и столбца, на пересечении которых расположен элемент с найденным значением.

24. Дана действительная квадратная матрица порядка n. Преобразовать матрицу по следующему правилу: строку с номером n сделать столбцом с номером n, а столбец с номером n — строкой с номером n.

25. Пусть дана действительная матрица размером $n \times m$. Требуется преобразовать матрицу следующим образом: поэлементно вычесть последнюю строку из всех строк, кроме последней.

26. Определить номера тех строк целочисленной матрицы $A[N, K]$, которые совпадают с массивом $D[K]$. Если таких строк нет, выдать соответствующее сообщение.

27. Определить наименьший элемент каждой четной строки матрицы $A[M, N]$.

28. Расположить столбцы матрицы $D[M, N]$ в порядке возрастания элементов k-й строки ($1 \le k \le M$).

29. Определить номера строк матрицы $R[M, N]$, хотя бы один элемент которых равен c, и элементы этих строк умножить на d.

30. Матрица $A[N, M]$ (M кратно 4) разделена по вертикали на две половины. Определить сумму элементов каждого столбца левой половины и сумму элементов каждого четного столбца правой половины матрицы A.

31. Дана квадратная целочисленная матрица порядка n. Сформировать результирующий одномерный массив, элементами которого являются строчные суммы тех строк, которые начинаются с k идущих подряд положительных чисел.

32. «Тестирование коллектива». Пусть целочисленная матрица размером $n \times m$ содержит информацию об учениках некоторого класса из n человек. В первом столбце проставлена масса (кг), во втором — рост (см), в третьем — успеваемость (средний балл) и т.д. (используйте свои дополнительные показатели). Ученик называется *среднестатистическим* по k-му параметру (*уникальным* по k-му параметру), если на нем достигается минимум (максимум) модуля разности среднего арифметического чисел из k-го столбца и значения k-го параметра этого ученика. Ученик называется *самым уникальным* (*самым средним*), если он уникален (является среднестатистическим) по самому большому количеству параметров. По данной матрице определить самых уникальных учеников и самых средних.

33. Прямоугольное поле разбито на $m \times n$ квадратных клеток. Некоторые клетки покрашены в черный цвет. Известно, что все черные клетки могут быть разбиты на несколько непересекающихся и не имеющих общих вершин черных прямоугольников. Считая, что цвета клеток даны в виде массива типа `array [1..m, 1..n] of boolean`, подсчитать число черных прямоугольников, о которых шла речь. Число действий должно быть порядка $m \times n$.

Указание. Число прямоугольников равно числу их левых верхних углов. Является ли клетка верхним углом, можно узнать, посмотрев на ее цвет, а также цвет верхнего и левого соседей. (Не забудьте, что их может не быть, если клетка с краю.)

34. Даны квадратная таблица $A[N, N]$ и число $M \le N$. Для каждого квадрата размером $M \times M$ в этой таблице вычислить сумму стоящих в нем чисел. Общее число действий должно быть порядка n^2.

Указание. Сначала для каждого горизонтального прямоугольника размером $M \times 1$ вычислить сумму стоящих в нем чисел. (При сдвиге такого прямоугольника по горизонтали на 1 нужно добавить одно число и одно вычесть.) Затем, используя эти суммы, вычислить суммы в квадратах. (При сдвиге квадрата по вертикали добавляется полоска, а другая полоска убавляется.)

35. Среди тех строк целочисленной матрицы, которые содержат только нечетные элементы, найти строку с максимальной суммой модулей элементов.

36. Подсчитать количество строк заданной целочисленной матрицы $N \times N$, являющихся перестановкой чисел 1, 2, ..., N (т.е. содержащих каждое из чисел 1, 2, ..., N ровно один раз).

37. Среди столбцов заданной целочисленной матрицы, содержащих только такие элементы, которые по модулю не больше 10, найти столбец с минимальным произведением элементов.

38. Массивом `chars[M][N]` кодируется поле, на котором расположено несколько прямоугольников. Каждый состоит из целого числа клеток, прямоугольники не накладываются друг на друга и не соприкасаются. Разные прямоугольники могут состоять из разных символов. Один и тот же прямоугольник не может состоять из различных символов. Пустые квадраты поля кодируются символом «точка». Подсчитать число прямоугольников разных типов. Пример:

```
# # # . . . ? ? . . + .
# # # . = . ? ? . . + .
# # # . . . . . . . + .
. . . . . ? ? ? . . . .
? ? ? . . . . . . . = =
? ? ? . . . # # # # . .
```

Для этого поля программа должна выдать ответ:

```
#-прямоугольников: 2
?-прямоугольников: 3
+-прямоугольников: 1
=-прямоугольников: 2
```

39. *Характеристикой столбца целочисленной матрицы* назовем сумму модулей его отрицательных нечетных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик.

40. Для заданной квадратной матрицы найти такие k, что k-я строка матрицы совпадает с k-м столбцом.

41. Найти максимальный элемент среди всех элементов тех строк заданной матрицы, которые упорядочены (либо по возрастанию, либо по убыванию).

42. *Расстояние* между k-й и l-й строками квадратной матрицы A ($N \times N$) определяется как

$$r = \sum_{j=1}^{N} \left(|a_{kj}| \cdot |a_{lj}| \right).$$

Указать номер строки, максимально удаленной от первой строки заданной матрицы.

43. Определить, является ли заданная матрица ортонормированной, т.е. равно ли скалярное произведение каждой пары различных строк (столбцов) нулю.

44. Определить среднее арифметическое элементов, лежащих на пересечении строк, номера которых кратны R, и столбцов, номера которых кратны S.

45. Определить номера строк матрицы, в которых знаки элементов чередуются.

6.9. Задачи по теме «Работа со строками»

1. Дана строка, заканчивающаяся точкой. Подсчитать, сколько слов в строке.

2. Дана строка, содержащая английский текст. Найти количество слов, начинающихся с буквы *b*.

3. Дана строка. Подсчитать, сколько в ней букв *r*, *k*, *t*.

4. Дана строка. Определить, сколько в ней символов *, ;, :.

5. Дана строка, содержащая текст. Найти длину самого короткого слова и самого длинного слова.

6. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов ему предшествует.

7. Дана строка, содержащая текст, заканчивающийся точкой. Вывести на экран слова, содержащие три буквы.

8. Дана строка. Преобразовать ее, удалив каждый символ * и повторив каждый символ, отличный от *.

9. Дана строка. Определить, сколько раз входит в нее группа букв *abc*.

10. Дана строка. Подсчитать количество букв *k* в последнем ее слове.

11. Дана строка. Подсчитать, сколько различных символов встречается в ней. Вывести их на экран.

12. Дана строка. Подсчитать самую длинную последовательность подряд идущих букв *a*.

13. Дана строка символов, среди которых есть одна открывающаяся и одна закрывающаяся скобка. Вывести на экран все символы, расположенные внутри этих скобок.

14. Имеется строка, содержащая буквы латинского алфавита и цифры. Вывести на экран длину наибольшей последовательности цифр, идущих подряд.

15. Дан набор слов, разделенных точкой с запятой (;). Набор заканчивается двоеточием (:). Определить, сколько в нем слов, заканчивающихся буквой *a*.

16. Дана строка. Указать те слова, которые содержат хотя бы одну букву *k*.

17. Дана строка. Найти в ней те слова, которые начинаются и оканчиваются одной и той же буквой.

18. В строке заменить все двоеточия (:) точкой с запятой (;). Подсчитать количество замен.

19. В строке удалить символ «двоеточие» (:) и подсчитать количество удаленных символов.

20. В строке между словами вставить вместо пробела запятую и пробел.

21. Удалить часть символьной строки, заключенной в скобки (вместе со скобками).

22. Определить, сколько раз в строке встречается заданное слово.

23. В строке имеется одна точка с запятой (;). Подсчитать количество символов до точки с запятой и после нее.

24. Дана строка. Преобразовать ее, заменив точками все двоеточия (:), встречающиеся среди первых $n/2$ символов, и заменив точками все восклицательные знаки, встречающиеся среди символов, стоящих после $n/2$ символов.

25. Строка содержит одно слово. Проверить, будет ли оно читаться одинаково справа налево и слева направо (т.е. является ли оно палиндромом).

26. В записке слова зашифрованы — каждое из них записано наоборот. Расшифровать сообщение.

27. Проверить, одинаковое ли число открывающихся и закрывающихся скобок в данной строке.

28. Строка, содержащая произвольный русский текст, состоит не более чем из 200 символов. Написать, какие буквы и сколько

раз встречаются в этом тексте. Ответ должен приводиться в грамматически правильной форме, например *а* — 25 раз, *к* — 3 раза и т.д.

29. Упорядочить данный массив английских слов по алфавиту.

30. Даны две строки *A* и *B*. Составьте программу, проверяющую, можно ли из букв, входящих в *A*, составить *B* (буквы можно использовать не более одного раза и можно переставлять).

Например, *A*: **ИНТЕГРАЛ**; *B*: **АГЕНТ** — составить можно; *B*: **ГРАФ** — составить нельзя.

31. Строка содержит произвольный русский текст. Проверить, каких букв в нем больше: гласных или согласных.

32. Двумерный массив $n \times m$ содержит некоторые буквы русского алфавита, расположенные в произвольном порядке. Написать программу, проверяющую, можно ли из этих букв составить данное слово *S*. Каждая буква массива используется не более одного раза.

33. Результаты вступительных экзаменов представлены в виде списка из *N* строк, в каждой строке которого записаны фамилия студента и отметки по каждому из *M* экзаменов. Определить количество абитуриентов, сдавших вступительные экзамены только на «отлично».

34. Составить программу преобразования натуральных чисел, записанных в римской нумерации, в десятичную систему счисления.

35. Из заданной символьной строки выбрать те символы, которые встречаются в ней только один раз, в том порядке, в котором они встречаются в тексте.

36. В символьном массиве хранятся фамилии и инициалы учеников класса. Требуется напечатать список класса с указанием для каждого ученика количества его однофамильцев.

37. Дано число в двоичной системе счисления. Проверить правильность ввода этого числа (в его записи должны быть только символы 0 и 1). Если число введено неверно, повторить ввод. При правильном вводе перевести число в десятичную систему счисления.

38. В заданной строке удалить все лишние пробелы.

39. Для заданного текста определить длину содержащейся в нем максимальной серии символов, отличных от букв.

40. *Расстояние между двумя словами равной длины* — это количество позиций, в которых различаются эти слова. В заданном предложении найти пару слов заданной длины с максимальным расстоянием.

41. Отредактировать заданное предложение, удаляя из него те слова, которые встречаются в предложении заданное число раз.

42. Напечатать те слова, которые встречаются в каждом из двух заданных предложений.

43. Отредактировать заданное предложение, удаляя из него все слова с нечетными номерами и переворачивая слова с четными номерами.

44. *Шифрация.* Один из методов шифрации называется наложением гаммы. Делается это следующим образом: берется некоторое случайное число в диапазоне от 127 до 255 — гамма, и код каждого символа строки заменяется кодом, получающимся в результате операции: `новый код=старый код XOR гамма`.

Написать программу, реализующую:

а) данный метод шифрации;

б) дешифрацию строки при заданной гамме.

Входные данные: шифруемая строка.

Выходные данные:

- гамма;
- зашифрованная строка.

45. *Кодировщик.* Написать программу, перекодирующую строку в кодировке KOI в строку в кодировке Windows-1251 и обратно.

46. *Тэг курсива.* Дан текст, в котором встречаются структуры `<i>` и `</i>`. Заменить каждое вхождение `<i>` на `<курсив>`, а каждое вхождение `</i>` на `<конец курсива>`. *Замечание.* В программе следует учесть, что буква `i` может быть как строчной, так и прописной.

47. *Форматирование текста.* Дан текст, состоящий из предложений, разделяемых точками. Напишите программу, производящую следующее форматирование: после каждой точки в конце предложения должен стоять хотя бы один пробел; первое слово в предложении должно начинаться с прописной буквы. *Замечание.* Текст может быть как на русском, так и на английском языке.

48. *Статистика.* Дан текст. Напишите программу, определяющую процентное отношение строчных и прописных букв к общему числу символов в нем.

49. *Статистика-2.* Дан текст. Определите, каких букв (строчных или прописных) в нем больше, и преобразуйте следующим образом: если больше прописных букв, чем строчных, то все буквы преобразуются в прописные; если больше строчных, то все буквы преобразуются в строчные; если поровну и тех и других — текст остается без изменения.

50. *Частота появления букв в словах.* Дан текст, содержащий слова на латинице, разделенные пробелами. Определить, какие буквы в словах совпадают чаще: первые, последние или средние. Позиция средней буквы в слове определяется по формуле:

```
поз_средн_буквы=длина_слова div 2+1
```

где `div` — операция целочисленного деления.

51. *Лишние пробелы.* Дана строка, состоящая из слов, разделенных пробелами. Напишите программу, удаляющую лишние пробелы. Пробел считается лишним, если он:

• стоит в начале строки;
• стоит в конце строки;
• следует за пробелом.

52. *Форматированный вывод числа.* С клавиатуры вводится целое число в десятичной системе счисления. Написать программу, реализующую вывод его представления с разделением на триады цифр.

Пример.
Число: 100000
Форматированный вывод: 100 000
Число: 1000000
Форматированный вывод: 1 000 000

53. *Оптимизатор исходного кода.* Для увеличения/уменьшения значения целочисленной переменной на единицу в языках программирования можно использовать операции сложения/вычитания, а можно — операции инкремент/декремент. Известно, что операции инкремент/декремент выполняются гораздо быстрее, чем сложение/вычитание, поэтому их использование часто предпочтительнее. Дан массив строк, представляющий фрагмент текста программы на языке Паскаль. Известно, что данный фрагмент оперирует только с целочисленными переменными. В каждой строке — одна команда.

Преобразовать данный текст, заменив каждую строку вида `переменная:=переменная+1;` на строку вида `Inc(переменная);` а каждую строку вида `переменная:=переменная-1;` на строку вида `Dec(переменная);`

Пример.
Исходный текст:

```
Begin
ReadLn(a,b);
a:=a+1;
c:=b+1;
b:=b-1;
WriteLn('a=',a);
WriteLn('b=',b);
End.
```

Преобразованный текст:

```
Begin
ReadLn(a,b);
Inc(a);
c:=b+1;
Dec(b);
WriteLn('a=',a);
WriteLn('b=',b);
End.
```

6.10. Задачи на «длинную арифметику»

1. Составить программу для сравнения двух n-значных чисел ($n > 20$).

2. Составить программу, суммирующую два натуральных n-значных числа, где $n > 20$.

3. Составить программу для вычисления степеней чисел вида a^n, если $a >$ `MaxInt`, $n > 10$.

4. Составить программу для вычисления числа $2^{64} - 1$, в результате сохранить все цифры.

5. Составить программу для вычисления 100!.

6. Составить программу для извлечения точного квадратного корня из n-разрядного числа ($n > 40$).

7. Составить программу для умножения многозначных чисел.

8. Составить программу для вычисления точного значения $n!$, где $n > 12$.

9. Составить программу для вычисления точного значения n^n, где $n > 10$.

10. Составить программу для деления числа a на число b, если a, b — многозначные числа.

11. Вычислить $100! + 2^{100}$.

12. Вычислить $100! - 2^{100}$.

13. Вычислить 7^{123}.

14. Встречаются ли среди цифр числа $2^{11213} - 1$ две идущие подряд девятки?

15. Вычислить 2^{-200}.

16. Составить программу для нахождения частного и остатка от деления m-значного числа на n-значное (m, $n > 20$).

17. Выяснить, какое из чисел a^m, b^n больше и на сколько (a, $b \le 40000$; m, $n \le 10$).

18. Найти n знаков в десятичной записи числа $\sqrt{m}$ ($n > 50$).

19. Найти количество делителей n-значного натурального числа ($n > 20$).

20. Вычислить точное значение $(n!)!$ ($n \ge 4$).

21. Составить программу для вычисления точного значения суммы $1! + 2! + 3! + \ldots + n!$ при $n > 10$.

22. Составить программу для вычисления точного значения суммы дробей $\frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$ при $n > 10$. Ответ должен быть представлен в виде несократимой дроби $\frac{p}{q}$, где p, q — натуральные числа.

23. Вычислить точное значение $(n^n)!$ при $n \ge 3$.

24. Составить программу для вычисления точного значения суммы первых n членов последовательности $1, k, k^2, k^3, \ldots, k^n$ ($n >$ `MaxInt`).

25. Составить программу для вычисления точного значения суммы первых n членов последовательности чисел, кратных данному натуральному числу k ($n >$ `MaxInt`).

26. Вычислить точное значение суммы $1^2 + 2^2 + 3^2 + \ldots + n^2$ ($n \geq 20000$).

27. Вычислить точное значение суммы $1^n + 2^n + 3^n + \ldots + n^n$ ($n \geq 10$).

28. Найти первое простое число, которое больше 10^{11}.

29. Составить программу для вычисления точного значения многочлена $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, где a_i и x — целые числа большие 10^{11}.

30. Найти наибольший общий делитель и наименьшее общее кратное чисел m и n (m, $n \geq 10^{11}$).

31. Проверить, являются ли числа m и n (m, $n \geq 10^{11}$) взаимно простыми.

32. Доказать, что число $2^{19936} \cdot (2^{19937} - 1)$ является совершенным, т.е. равно сумме всех своих делителей, кроме самого себя.

33. Вычислить функцию $Y = F(X)$, разложенную в степенной ряд, с заданной степенью точности ε ($10^{-1} \leq \varepsilon \leq 10^{-1000}$).

6.11. Задачи по теме «Множества»

1. Известны сорта роз, выращиваемых тремя цветоводами: «Анжелика», «Виктория», «Гагарин», «Ave Maria», «Катарина», «Юбилейная». Определить те сорта, которые имеются у каждого из цветоводов, которые есть хотя бы у одного из цветоводов, которых нет ни у одного из цветоводов.

2. Заданы имена девочек. Определить, какие из этих имен встречаются во всех классах данной параллели, которые есть только в некоторых классах и какие из этих имен не встречаются ни в одном классе.

3. Задан некоторый набор товаров. Определить для каждого из товаров, какие из них имеются в каждом из n магазинов, какие товары есть хотя бы в одном магазине и каких товаров нет ни в одном магазине.

4. Имеется список класса (все имена различны). Определить, есть ли в классе человек, который побывал в гостях у всех. (Для каждого ученика составить множество побывавших у него в гостях друзей, сам ученик в это множество не входит.)

5. Имеется множество, содержащее натуральные числа из некоторого диапазона. Сформировать два множества, первое из которых содержит все простые числа из данного множества, а второе — все составные.

6. На трех участках возделывают сельскохозяйственные культуры. Известны виды культур, выращиваемых на каждом из участ-

ков. Определить виды тех культур, которые возделывают на каждом из участков; возделывают хотя бы на одном из участков; не возделывают ни на одном участке. (Культуры: картофель, укроп, морковь, горох, капуста, редис.)

7. Известны марки машин, изготовляемых в данной стране и импортируемых за рубеж. Даны некоторые N стран. Определить для каждой из марок, какие из них были:

- доставлены во все страны;
- доставлены в некоторые из стран;
- не доставлены ни в одну страну.

8. В озере водится несколько видов рыб. Три рыбака поймали рыб, представляющих некоторые из имеющихся видов. Определить:

- какие виды рыб есть у каждого рыбака;
- какие рыбы есть в озере, но нет ни у одного из рыбаков.

9. В N колхозах выращивают некоторые сельскохозяйственные культуры из имеющегося перечня. Определить культуры:

- возделываемые во всех колхозах;
- возделываемые только в некоторых колхозах.

10. Есть список игрушек, некоторые из которых имеются в N детских садах. Определить игрушки из списка:

- которых нет ни в одном из детсадов;
- которые есть в каждом из детсадов.

11. Составить программу, которая вычисляет сумму тех элементов двумерного массива, номера строк и столбцов которых принадлежат соответственно непустым множествам S_1 и S_2.

12. Задано некоторое множество M и множество T того же типа. Подсчитать, сколько элементов из множеств T и M совпадает.

13. Из диапазона целых чисел $m \ldots n$ выделить:

1) множество чисел, делящихся без остатка или на k, или на l (k, l — простые);

2) множество чисел, делящихся на $k \cdot l$ без остатка.

14. Дан текст из цифр и строчных латинских букв, за которыми следует точка. Определить, каких букв — гласных (*a*, *e*, *i*, *o*, *u*) или согласных — больше в этом тексте.

15. Подсчитать количество различных цифр в десятичной записи натурального числа.

16. Напечатать в возрастающем порядке все цифры, не входящие в запись данного натурального числа.

17. Дан текст из строчных латинских букв, за которыми следует точка. Напечатать все буквы, входящие в текст не менее двух раз.

18. Дан текст из строчных латинских букв, за которыми следует точка. Напечатать все буквы, входящие в текст по одному разу.

19. Дан текст, за которым следует точка. В алфавитном порядке напечатать все строчные русские гласные буквы (*а*, *е*, *и*, *о*, *у*, *ы*, *э*, *ю*, *я*), входящие в этот текст.

20. Дан текст на русском языке. Напечатать в алфавитном порядке все гласные буквы, которые входят в каждое слово.

21. Дан текст на русском языке. Напечатать в алфавитном порядке все согласные буквы, которые не входят ни в одно слово.

22. Дан текст на русском языке. Напечатать в алфавитном порядке все звонкие согласные буквы, которые входят в каждое нечетное слово и не входят ни в одно четное слово.

23. Дан текст на русском языке. Напечатать в алфавитном порядке все звонкие согласные буквы, которые входят хотя бы в одно слово.

24. Дан текст на русском языке. Напечатать в алфавитном порядке все глухие согласные буквы, которые не входят хотя бы в одно слово.

25. Дан текст на русском языке. Напечатать в алфавитном порядке все согласные буквы, которые входят только в одно слово.

26. Дан текст на русском языке. Напечатать в алфавитном порядке все глухие согласные буквы, которые не входят только в одно слово.

27. Дан текст на русском языке. Напечатать в алфавитном порядке все звонкие согласные буквы, которые входят более чем в одно слово.

28. Дан текст на русском языке. Напечатать в алфавитном порядке все гласные буквы, которые не входят более чем в одно слово.

29. Дан текст на русском языке. Напечатать в алфавитном порядке все глухие согласные буквы, которые входят в каждое нечетное слово и не входят хотя бы в одно четное слово.

6.12. Задачи по теме «Записи (структуры)»

1. Распечатать список учеников, фамилии которых начинаются на букву *В*, с указанием даты их рождения.

2. Из данного списка спортсменов распечатать сведения о тех из них, кто занимается плаванием. Указать возраст, сколько лет они занимаются спортом.

3. Вычислить средний балл учеников класса, если известны оценки каждого ученика по математике, русскому языку и физике. Распечатать список учеников, имеющих средний балл выше среднего в классе.

4. Распечатать фамилии рабочих бригады, начинающиеся с букв *А* и *С*, с указанием их месячной зарплаты.

5. Из ассортимента конфет, выпускаемых пермской кондитерской фабрикой, выбрать те, стоимость которых от 30 до 55 руб. за 1 кг. Указать срок их годности и номера магазинов, в которых они имеются в продаже.

6. Распечатать список учеников музыкальной школы, которые учатся играть на скрипке. Указать также, сколько лет они занимаются музыкой и принимали ли участие в каких-либо конкурсах.

7. Среди работников данного предприятия найти тех, чья заработная плата за месяц ниже средней по предприятию, а также распечатать список тех, кто проработал на предприятии более 10 лет, с указанием их фамилии, зарплаты, стажа работы и должности.

8. Распечатать фамилии тех учеников, которые не получили ни одной тройки за последнюю четверть. В каких классах учатся эти ученики? Каков их средний балл?

9. Распечатать фамилии детей данного детского сада, которые родились в определенном месяце; указать их возраст и группу.

10. Распечатать список тех учителей школы, которые преподают математику и информатику, указать стаж их работы и недельную нагрузку.

11. Распечатать анкетные данные учеников, участвовавших в олимпиаде по информатике и заработавших не менее 30 баллов.

12. Распечатать фамилии тех учеников класса, которые являются хорошистами и отличниками по итогам года. Также указать, насколько их средний балл отличается от среднего балла класса.

13. По данным сведениям об учениках класса определить среднюю массу мальчиков и средний рост девочек. Кто из учеников класса самый высокий?

14. Даны результаты переписи населения, которые хранятся в памяти ЭВМ. Напечатать фамилии, имена и подсчитать общее число жителей, родившихся после 1990 г.

15. При поступлении в университет лица, получившие оценку «неудовлетворительно» на первом экзамене, ко второму экзамену не допускаются. Считая фамилии абитуриентов и их оценки после первого экзамена исходными данными, составить список абитуриентов, допущенных ко второму экзамену.

16. Составить программу назначения стипендии студентам по результатам сессии, используя следующие правила:

1) если все оценки 5, назначается повышенная стипендия;

2) если все оценки 4 и 5, назначается обычная стипендия;

3) если есть оценка 3, стипендия не назначается.

В результате работы программы должен быть напечатан список группы с оценками и средним баллом каждого студента и два списка фамилий (назначенных на повышенную и обычную стипендию).

17. В таблице хранятся следующие данные об учениках: фамилия, имя, отчество, рост, масса. Вычислить средний рост учеников, рост самого высокого и самого низкого ученика. Сколько учеников могут заниматься в баскетбольной секции, если рост баскетболиста должен быть больше 170 см?

18. На аптечном складе хранятся лекарства. Сведения о лекарствах содержатся в специальной ведомости: наименование лекарственного препарата; количество; цена; срок хранения (в месяцах). Выяснить, сколько стоит самый дорогой и самый дешевый препарат; сколько препаратов хранится на складе; какие препара-

ты имеют срок хранения более 3 месяцев; сколько стоят все препараты, хранящиеся на складе.

19. В столовой предлагается N комплексных обедов, состоящих из Q блюд. Известна стоимость и калорийность каждого блюда. Сколько стоит самый дешевый и самый дорогой обед? Сколько калорий включает в себя самое калорийное блюдо?

20. Торговый склад производит уценку хранящейся продукции. Если продукция хранится на складе дольше n месяцев, то она уценивается в 2 раза, а если срок хранения превысил m ($m < n$) месяцев, но не достиг n, то — в 1,5 раза. Ведомость уценки товаров должна содержать следующую информацию: наименование товара, количество товара, цена товара до уценки, срок хранения товара, цена товара после уценки, общая стоимость товара до уценки, общая стоимость товаров после уценки. Выяснить максимальный и минимальный сроки хранения товаров на складе; максимальную и минимальную цену товаров до уценки и после уценки.

21. N спортсменов-многоборцев принимают участие в соревнованиях по M видам спорта. По каждому виду спорта спортсмен набирает определенное количество очков. Вычислить, сколько очков в сумме набрал каждый спортсмен после окончания соревнований. Вычислить разницу в очках для спортсменов, занявших первое и последнее места.

22. N учеников проходили тестирование, выполнив M тестов по какому-либо предмету. Сколько очков набрал каждый ученик по всем темам? Вычислить средний балл, полученный учениками, и разницу между лучшим результатом и средним баллом.

23. Описать переменную `служащий`, состоящую из имени, фамилии, отчества служащего, даты рождения, образования, домашнего адреса, профессии. Определить имена людей с высшим образованием. Выдать данные о служащем, который имеет ту или иную профессию.

24. Описать переменную `круг`, в которой содержатся все данные для построения круга в декартовой системе координат.

а) Определить координаты центра, радиус, площадь и длину окружности круга минимального радиуса, который будет содержать внутри себя все заданные круги.

б) Рассматривая окружности попарно, определить координаты точек пересечения или точки касания для каждой пары, либо вывести сообщение об отсутствии общих точек.

25. Описать переменную `экзаменационная ведомость` (предмет, номер группы, номер зачетной книжки, фамилия, имя, отчество студента, его оценки по итогам текущей сессии). Определить отличников, хорошистов, троечников и двоечников.

26. Создать записи, определяющие положение точки в декартовой и в полярной системах координат. Считая, что задан массив координат точек в декартовой системе координат, получить соот-

ветствующий массив координат заданных точек в полярной системе координат.

27. Задан массив квадратных трехчленов, где каждый трехчлен имеет комплексные коэффициенты. Сформировать массив корней трехчленов. Сформировать массив значений трехчленов в точке $x = c + di$.

28. Дан массив записей, содержащих дату (число, месяц, год) и время (час, минута, секунда). Упорядочить этот массив в порядке «возрастания», т.е. от более ранних значений к более поздним.

29. Описать переменную `расписание`, содержащую:

- день недели;
- количество пар в этот день;
- время начала и конца пары;
- название предмета;
- фамилию преподавателя.

Вывести полную информацию о занятиях, относящихся к предметной области «Информатика».

30. В библиотеке имеются книги, газеты, журналы. Для каждого печатного издания указать:

- название;
- год выпуска (для книги), дату выпуска (для газет и журналов);
- автора (для книги), редактора (для газеты), состав редколлегии (для журнала);
- объем.

Вывести информацию об изданиях, вышедших в заданном году.

6.13. Задачи по теме «Файлы»

6.13.1. Типизированные файлы целых чисел

1. Заполнить файл последовательного доступа `f` целыми числами, полученными с помощью генератора случайных чисел. Получить в файле `g` те компоненты файла `f`, которые являются четными.

2. Записать в файл последовательного доступа N действительных чисел. Вычислить произведение компонентов файла и вывести на печать.

3. Заполнить файл последовательного доступа `f` целыми числами, полученными с помощью генератора случайных чисел. Получить в файле `g` все компоненты файла `f`, которые делятся на m и не делятся на n.

4. Записать в файл последовательного доступа N целых чисел, полученных с помощью генератора случайных чисел. Подсчитать количество пар противоположных чисел среди компонентов этого файла.

5. Заполнить файл последовательного доступа `f` целыми числами, полученными с помощью генератора случайных чисел. Из

файла f получить файл g, исключив повторные вхождения чисел. Вывести файл g на печать.

6. Записать в файл последовательного доступа N произвольных натуральных чисел. Переписать в другой файл последовательного доступа те элементы, которые кратны K. Вывести полученный файл на печать.

7. Заполнить файл последовательного доступа N действительными числами, полученными с помощью датчика случайных чисел. Найти сумму минимального и максимального элементов этого файла.

8. Записать в файл последовательного доступа N натуральных чисел: a_1, a_2, ..., a_n (числа получить с помощью датчика случайных чисел). Сформировать новый файл последовательного доступа, элементами которого являются числа a_1, $a_1 \cdot a_2$, $a_1 \cdot a_2 \cdot a_3$, ..., $a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n$.

9. Записать в файл f последовательного доступа N натуральных чисел. Получить в другом файле последовательного доступа все компоненты файла f, кроме тех, которые кратны K. Вывести полученный файл на печать.

10. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Найти количество удвоенных нечетных чисел среди компонентов файла.

11. Заполнить файл f натуральными числами, полученными с помощью генератора случайных чисел. Найти количество квадратов нечетных чисел среди компонентов.

12. Записать в файл прямого доступа N действительных чисел. Найти наибольшее из значений модулей компонентов с нечетными номерами.

13. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Из файла f получить файл g, исключив повторные вхождения чисел. Порядок следования чисел сохранить.

14. Записать в файл последовательного доступа N действительных чисел. Найти разность первого и последнего компонентов файла.

15. Записать в файл f N целых чисел, полученных с помощью генератора случайных чисел. Заполнить файл g числами, которые являются произведениями соседних компонентов файла f.

16. Записать в файл последовательного доступа n элементов последовательности $b_n = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + \cdots + (-1)^{n-1} \cdot \frac{1}{n!}$. Вывести на печать те компоненты файла, для которых выполняется $|b_n| > \varepsilon$, где ε — заданное число.

17. Записать в файл последовательного доступа N действительных чисел a_1, a_2, ..., a_n. Организовать новый файл последователь-

ного доступа, элементы которого вычисляются по формуле $b_k = \frac{\sum_{k=1}^{i} a_k}{i}$. Вывести полученный файл на печать.

18. Багаж пассажира характеризуется количеством вещей и их общим весом. Дан файл `Bagazh`, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно — действительного (вес в килограммах).

Найти багаж, средний вес одной вещи в котором отличается не более чем на m кг от общего среднего веса одной вещи.

19. В условиях предыдущей задачи найти число пассажиров, имеющих более двух вещей, и число пассажиров, количество вещей которых превосходит среднее число вещей.

20. В условиях задачи 18 выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом менее m кг.

21. Дан файл `Bibl`, содержащий сведения о книгах. Сведения о каждой из книг — это фамилия автора, название и год издания.

Найти названия книг данного автора, изданных начиная с 1960 г.

22. В условиях предыдущей задачи определить, имеется ли книга с названием «Информатика». Если да, то напечатать фамилию автора и год издания. Если таких книг несколько, то напечатать имеющиеся сведения обо всех этих книгах.

23. Дан файл `T`, который содержит номера телефонов сотрудников учреждения: указываются фамилия, инициалы и номер телефона. Найти номер телефона сотрудника по его фамилии и инициалам.

24. Дан файл, содержащий различные даты. Каждая дата — это число, месяц и год. Найти год с наименьшим номером.

25. Дан файл, содержащий различные даты. Каждая дата — это число, месяц и год. Найти все весенние даты.

26. В условиях предыдущей задачи найти самую позднюю дату.

27. Дан файл `Tovar`, содержащий сведения об экспортируемых товарах: указываются наименование товара, страна, импортирующая товар, и объем поставляемой партии в штуках. Составить список стран, в которые экспортируется данный товар, и указать общий объем его экспорта.

28. Дан файл `Assort`, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет).

а) Получить название игрушек, цена которых не превышает 140 руб. и которые подходят детям 5 лет;

б) определить стоимость самого дорогого конструктора;

в) напечатать название наиболее дорогих игрушек (цена которых отличается от цены самой дорогой игрушки не более чем на 50 руб.);

г) получить названия игрушек, которые подходят детям как четырех, так и десяти лет;

д) получить сведения о том, можно ли подобрать игрушку, любую, кроме мяча, подходящую ребенку трех лет;

е) получить название самой дешевой игрушки;

ж) получить название самой дорогой игрушки для детей до четырех лет;

з) получить названия игрушек для детей четырех-пяти лет;

и) получить название самой дорогой игрушки, подходящей детям двух-трех лет;

к) определить стоимость самой дорогой куклы;

л) определить стоимость кукол для детей шести лет;

м) для детей какого возраста предназначается конструктор?

н) для детей какого возраста предназначены кубики? Указать их среднюю стоимость.

6.13.2. Текстовые файлы

В заданиях 1—25 исходные текстовые файлы создаются с помощью какого-либо текстового процессора.

1. Дан файл, содержащий текст, записанный строчными русскими буквами. Получить в другом файле тот же текст, записанный заглавными буквами.

2. Дан файл, содержащий произвольный текст. Выяснить, чего в нем больше: русских букв или цифр.

3. Дан файл, содержащий текст на русском языке. Выяснить, входит ли данное слово в указанный текст, и если да, то сколько раз.

4. Дан файл, содержащий текст на русском языке. В предложениях некоторые из слов записаны подряд несколько раз (предложение заканчивается точкой или восклицательным знаком). Получить в новом файле отредактированный текст, в котором удалены повторные вхождения слов в предложение.

5. Дан файл, содержащий текст, набранный заглавными русскими буквами. Провести частотный анализ текста, т.е. указать (в процентах), сколько раз встречается та или иная буква.

6. Дан файл, содержащий текст на русском языке. Определить, сколько раз встречается в нем самое длинное слово.

7. Дан файл, содержащий произвольный текст. Проверить, правильно ли в нем расставлены круглые скобки (т.е. находится ли правее каждой открывающейся скобки закрывающаяся и левее закрывающейся — открывающаяся).

8. Дан файл, содержащий текст на русском языке. Составить в алфавитном порядке список всех слов, встречающихся в этом тексте.

9. Дан файл, содержащий текст на русском языке. Определить, сколько раз встречается в нем самое короткое слово.

10. Дан файл, содержащий текст на русском языке и некоторые два слова. Определить, сколько раз они встречаются в тексте и сколько из них — непосредственно друг за другом.

11. Дан файл, содержащий текст на русском языке. Выбрать из него те символы, которые встречаются в нем только один раз, в том порядке, в котором они встречаются в тексте.

12. Дан файл, содержащий текст и арифметические выражения вида a⊗b, где ⊗ — один из знаков +, –, *, /. Выписать все арифметические выражения и вычислить их значения.

13. Даны файл, содержащий текст на русском языке, и некоторые буквы. Найти слово, содержащее наибольшее количество указанных букв.

14. Даны файл, содержащий текст на русском языке, и некоторая буква. Подсчитать, сколько слов начинается с указанной буквы.

15. Дан файл, содержащий текст на русском языке. Найти слово, встречающееся в каждом предложении, или сообщить, что такого слова нет.

16. Дан файл, содержащий текст, включающий русские и английские слова. Подсчитать, каких букв в тексте больше — русских или латинских.

17. Дан файл, содержащий текст. Сколько слов в тексте? Сколько цифр в тексте?

18. Дан файл, содержащий текст, включающий в себя русские и английские слова. Получить новый файл, заменив в исходном все заглавные буквы строчными и наоборот.

19. Дан файл, содержащий зашифрованный русский текст. Каждая буква заменяется на следующую за ней (буква *я* заменяется на *а*). Получить в новом файле расшифровку данного текста.

20. Даны два текстовых файла f_1 и f_2. Файл f_1 содержит произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Файл f_2 содержит не более 30 слов, которые разделены запятыми. Эти слова образуют пары: каждое второе является синонимом первого. Заменить в файле f_1 те слова, которые можно, их синонимами. Результат поместить в новый файл.

21. Дан текстовый файл. Удалить из него все лишние пробелы, оставив между словами не более одного пробела. Результат поместить в новый файл.

22. Даны текстовый файл и некоторое слово. Напечатать те строки файла, которые содержат данное слово.

23. Дан текстовый файл. Напечатать в алфавитном порядке все слова из данного файла, имеющие заданную длину *n*.

24. Текстовый файл содержит запись многочлена некоторой степени с одной переменной x, коэффициенты многочлена — целые. Например, `5x^4-3x^3+15x^2-4`. Указать степень многочлена, его коэффициенты. Дописать в указанный файл таблицу значений этого многочлена на данном отрезке $[a, b]$.

25. Дан файл, содержащий текст на русском языке. Подсчитать количество слов, начинающихся и заканчивающихся на одну и ту же букву.

6.14. Задачи по теме «Модули»

I. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над комплексными числами:

1) сложения;
2) вычитания;
3) умножения;
4) деления;
5) модуля комплексного числа;
6) возведения комплексного числа в степень n (n — натуральное).

Комплексное число представить следующим типом:

```
Type Complex=Record
          R: Real;
          M: Real
End;
```

Используя этот модуль, решить следующие задачи.

1. Дан массив A — массив комплексных чисел. Получить массив C, элементами которого будут модули сумм рядом стоящих комплексных чисел.

2. Дан массив $A[M]$ — массив комплексных чисел. Получить матрицу $B[N, M]$, каждая строка которой получается возведением в степень, равную номеру этой строки, данного массива A.

II. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над обыкновенными дробями вида $\frac{P}{Q}$ (P — целое, Q — натуральное):

1) сложения;
2) вычитания;
3) умножения;
4) деления;
5) сокращения дроби;
6) возведения дроби в степень n (n — натуральное);
7) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Дробь представить следующим типом:

```
Type Frac=Record
          P: Integer;
          Q: 1..32767
End;
```

Используя этот модуль, решить задачи 1, 2.

1. Дан массив A — массив обыкновенных дробей. Найти сумму всех дробей, результат представить в виде несократимой дроби. Вычислить среднее арифметическое всех дробей, результат представить в виде несократимой дроби.

2. Дан массив A — массив обыкновенных дробей. Отсортировать его в порядке возрастания.

III. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций с квадратными матрицами:

1) сложения двух матриц;
2) умножения одной матрицы на другую;
3) нахождения транспонированной матрицы;
4) вычисления определителя матрицы.

Матрицу описать следующим образом:

```
Const NMax=10;
Type Matrica=Array[1..NMax,1..Nmax] Of Real;
```

Используя этот модуль, решить следующие задачи.

1. Решить систему линейных уравнений N-го порядка ($2 \le N \le 10$) методом Крамера.

2. Задан массив величин типа `Matrica`. Отсортировать этот массив в порядке возрастания значений определителей матриц.

IV. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над векторами:

1) сложения;
2) вычитания;
3) скалярнго умножения векторов;
4) умножения вектора на число;
5) нахождения длины вектора.

Вектор представить следующим типом:

```
Type Vector=Record
X,Y: Real
End;
```

Используя этот модуль, решить задачи 1, 2.

1. Дан массив A — массив векторов. Отсортировать его в порядке убывания длин векторов.

2. С помощью датчика случайных чисел сгенерировать $2N$ целых чисел. N пар этих чисел задают N точек координатной плоскости. Вывести номера тройки точек, которые являются координатами вершин треугольника с наибольшим углом.

V. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в P-ичной системе счисления ($2 \le P \le 9$):

1) сложения;
2) вычитания;
3) умножения;
4) деления;
5) перевода из десятичной системы счисления в P-ичную;
6) перевода из P-ичной системы счисления в десятичную;
7) функции проверки правильности записи числа в P-ичной системе счисления;
8) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

P-ичное число представить следующим типом:

```
Type Chislo=Array[1..16] Of 0..9;
```

Используя этот модуль, решить задачи 1, 2.

1. Возвести число в степень (основание и показатель степени записаны в P-ичной системе счисления). Результат выдать в P-ичной и десятичной системах счисления.

2. Дан массив A — массив чисел, записанных в P-ичной системе счисления. Отсортировать его в порядке убывания. Результат выдать в P-ичной и десятичной системах счисления.

VI. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в шестнадцатеричной системе счисления:

1) сложения;
2) вычитания;
3) умножения;
4) деления;
5) перевода из двоичной системы счисления в шестнадцатеричную;
6) перевода из шестнадцатеричной системы счисления в десятичную;
7) функции проверки правильности записи числа в шестнадцатеричной системе счисления;
8) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Используя этот модуль, решить следующие задачи.

1. Возвести число в степень (основание и показатель степени записаны в шестнадцатеричной системе счисления). Результат выдать в шестнадцатеричной и десятичной системах счисления.

2. Дан массив A — массив чисел, записанных в шестнадцатеричной системе счисления. Отсортировать его в порядке убывания. Результат выдать в шестнадцатеричной и десятичной системах счисления.

VII. Определим граф как набор точек, некоторые из которых соединены отрезками, подграф — как граф, являющийся подмножеством данного графа.

Реализовать в виде модуля набор подпрограмм, определяющих:

1) число точек в графе;

2) число отрезков в графе;

3) число изолированных подграфов в графе (подграфов, не соединенных отрезками);

4) диаметр графа — длину максимальной незамкнутой линии в графе (длина каждого звена — 1);

5) граф — объединение двух графов;

6) подграф — пересечение двух графов;

7) подграф — дополнение данного графа до полного (графа с тем же количеством вершин, что и в данном, и с линиями между любыми двумя вершинами);

8) число отрезков, выходящих из каждой вершины графа.

При запуске должны инициализироваться переменные: `Full_Graph` — полный граф с числом вершин `NumberOfVertix`, `Null_Graph` — граф без отрезков с числом вершин `NumberOfVertix`.

Граф представить как объект

```
Const NumberOfVertix=50;
Type Graph=Array[1..NumberOfVertix,
                 1..NumberOfVertix] Of Boolean;
```

Используя модуль, решить следующую задачу.

Найти все правильные графы из N вершин (граф правилен, если из всех вершин выходит равное количество отрезков).

VIII. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над длинными числами:

1) сложения;

2) вычитания;

3) умножения;

4) нахождения частного и остатка от деления одного числа на другое;

5) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Длинное число представить следующим типом:

```
Type Tsifra=0..9;
Chislo=Array[1..1000] Of Tsifra;
```

Используя этот модуль, решить задачи 1, 2.

1. Возвести число в степень (основание и показатель степени — длинные числа).

2. Дан массив длинных чисел. Упорядочить этот массив в порядке убывания.

IX. Реализовать в виде модуля набор подпрограмм для выполнения операций с многочленами от одной переменной (первый многочлен степени m, второй — степени n):

1) сложения;
2) вычитания;
3) умножения;
4) деления с остатком;
5) операций отношения (равно, не равно);
6) возведения в натуральную степень k;
7) вычисления производной от многочлена;
8) вычисления значения в точке x_0.

Многочлен представить следующим типом:

```
Type Mnogochlen=Array[1..500] Of Integer;
```

Используя этот модуль, решить задачи 1, 2.

1. Найти наибольший общий делитель многочленов $P(x)$ и $Q(x)$.
2. Вычислить $P^s(x) - Q^r(x)$.

X. Разработать способ представления множеств, содержащих более 255 элементов (до 2000). Создать модуль, позволяющий выполнять следующие операции над элементами таких множеств:

а) объединение;
б) пересечение;
в) разность;
г) функция проверки принадлежности элемента множеству;
д) функция проверки, является ли данное множество подмножеством (надмножеством) другого.

Используя созданный модуль, решить следующие задачи.

1. Дан массив множеств. Упорядочить элементы массива в порядке возрастания количества компонент соответствующих множеств.
2. Разработать программу, которая вводит несколько множеств, выражение, операндами которого являются эти множества, с операциями объединения, пересечения и вычитания, вычисляет значение этого выражения и выводит результат.

6.15. Задачи по теме «Динамические структуры данных»

1. Составить программу, которая вставляет в список L новый элемент F за каждым вхождением элемента E.
2. Составить программу, которая вставляет в список L новый элемент F перед первым вхождением элемента E, если E входит в L.
3. Составить программу, которая вставляет в непустой список L, элементы которого упорядочены по неубыванию, новый элемент E так, чтобы сохранилась упорядоченность.
4. Составить программу, которая удаляет из списка L все элементы E, если таковые имеются.

5. Составить программу, которая удаляет из списка L за каждым вхождением элемента E один элемент, если таковой имеется и он отличен от E.

6. Составить программу, которая удаляет из списка L все отрицательные элементы.

7. Составить программу, которая проверяет, есть ли в списке L хотя бы два одинаковых элемента.

8. Составить программу, которая переносит в конец непустого списка L его первый элемент.

9. Составить программу, которая вставляет в список L за первым вхождением элемента E все элементы списка L, если E входит в L.

10. Составить программу, которая переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке.

11. Составить программу, которая в списке L из каждой группы подряд идущих одинаковых элементов оставляет только один.

12. Составить программу, которая формирует список L, включив в него по одному разу элементы, которые входят одновременно в оба списка L_1 и L_2.

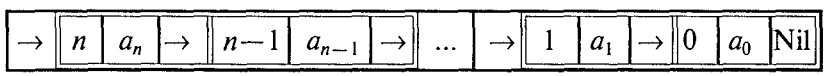
13. Составить программу, которая формирует список L, включив в него по одному разу элементы, которые входят в список L_1, но не входят в список L_2.

14. Составить программу, которая формирует список L, включив в него по одному разу элементы, которые входят в один из списков L_1 и L_2, но в то же время не входят в другой.

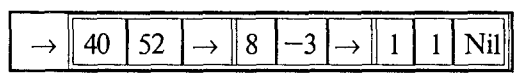
15. Многочлен

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включается в список.



Ниже показано представление многочлена $S(x) = 52x^{40} - 3x^8 + x$.



Описать на языке Паскаль тип данных, соответствующий такому представлению многочленов, и определить следующие процедуры и функции для работы с этими списками-многочленами:

15.1. Логическую функцию `Ravno(P,Q)`, проверяющую на равенство многочлены P и Q.

15.2. Функцию `Znach(P,x)`, вычисляющую значение многочлена P в целочисленной точке x.

15.3. Процедуру `Diff(P,Q)`, которая строит многочлен Q — производную многочлена P.

15.4. Процедуру `Slozh(P,Q,R)`, которая строит многочлен R — сумму многочленов Q и P.

15.5. Процедуру `Print(P,v)`, которая печатает многочлен P как многочлен от переменной, однобуквенное имя которой является значением литерного параметра v; например, для указанного выше многочлена S процедура `Print(S,'y')` должна напечатать `52y↑40-3y↑8+y`.

15.6. Процедуру `Vvod(P)`, которая считывает из входного файла безошибочную запись многочлена (за ней — пробел) и формирует соответствующий список-многочлен P.

15.7. Дан многочлен $P(x)$ степени n. Получить многочлен $P^2(x)$.

15.8. Дан многочлен $P(x)$ степени n. Получить многочлен $P(x+1)-P(x)$. Какова степень этого многочлена?

16. Составить программу для упорядочения в порядке возрастания элементов однонаправленного списка.

17. Составить программу, заполняющую список последовательностью случайных различных целых чисел и суммирующую те его элементы, которые расположены между минимальным и максимальным элементом (если минимальный элемент предшествует максимальному).

18. Дан список, содержащий целые числа. Сформировать другой список из элементов данного, абсолютные величины которых являются простыми числами.

19. Дан список, содержащий натуральные числа. Удалить те его элементы, которые кратны заданному числу k.

20. Дан список, элементами которого являются векторы

```
(Const NMax=200;
Type Vector=Array[1..NMax] Of Real;).
```

Сформировать список из длин этих векторов.

21. Элементами списка являются слова — имена существительные, записанные в именительном падеже (строки длиной не более 15 символов). Составить программу, которая добавляет за каждым словом все его падежные формы.

22. Дан список, содержащий целые числа. Определить количество различных элементов этого списка.

23. Даны упорядоченные списки L_1 и L_2. Вставить элементы списка L_2 в список L_1, не нарушая его упорядоченности.

24. Дан список, содержащий запись неотрицательных целых чисел в двоичной системе счисления. Заменить каждый элемент списка на его запись в шестнадцатеричной системе счисления.

25. Дан список, содержащий обыкновенные дроби вида $\frac{P}{Q}$, (P — целое, Q — натуральное). Составить программу для суммирования модулей этих дробей. Ответ представить в виде обыкновенной несократимой дроби.

26. Программа должна находить среднее арифметическое элементов непустого однонаправленного списка вещественных чисел, заменять все вхождения числа x на число y, менять местами первый и последний элементы, проверять, упорядочены ли числа в списке по возрастанию.

27. Дан список вещественных чисел. Написать следующие функции:

а) проверки наличия в нем двух одинаковых элементов;

б) переноса в начало его последнего элемента;

в) переноса в конец его первого элемента;

г) вставки списка самого в себя вслед за первым вхождением числа x.

28. Дан список строк. Написать следующие подпрограммы:

а) обращение списка (изменить ссылки в списке так, чтобы элементы оказались расположены в противоположном порядке);

б) из каждой группы подряд идущих элементов оставить только один;

в) оставить в списке только первые вхождения одинаковых элементов.

29. Даны два списка L_1 и L_2 пар вещественных чисел. Написать подпрограммы, возвращающие новый список L, включающий в себя:

а) пары списка L_1, первая координата которых встречается как вторая координата у пар списка L_2;

б) пары (x, y) списка L_1, встречающиеся в виде (y, x) в списке L_2;

в) пары (x, y), где $x < y$ списка L_1.

30. Даны два списка L_1 и L_2 вещественных чисел. Написать подпрограммы, возвращающие новый список L, включающий по одному разу числа, которые:

а) входят одновременно в оба списка;

б) входят хотя бы в один из списков;

в) входят в один из списков L_1 и L_2, но в то же время не входят в другой из них;

г) входят в список L_1, но не входят в список L_2.

31. Целое длинное число представляется строкой цифр. Написать программу, упорядочивающую числа по неубыванию.

32. Дан список слов, среди которых есть пустые. Написать подпрограмму, выполняющую следующее действие:

а) перестановку первого и последнего непустых слов;

б) печать текста из первых букв непустых слов;

в) удаление из непустых слов первых букв;

г) определение количества слов в непустом списке, отличных от последнего.

6.16. Задачи по теме «Графика»

Построить чертежи к следующим задачам.

1. В треугольной пирамиде построить сечение, параллельное основанию.

2. В треугольной пирамиде построить сечение, проходящее через боковое ребро и медиану основания.

3. В треугольной пирамиде построить сечение, проходящее через одну из сторон основания и середину противоположного ребра.

4. В треугольной пирамиде построить сечение, проходящее через среднюю линию боковой грани и противоположную вершину основания.

5. В треугольной пирамиде провести сечение, проходящее через сторону основания и наклоненное к основанию под углом 30°.

6. В правильной четырехугольной пирамиде провести сечение, проходящее через диагональ основания и вершину пирамиды.

7. В правильной четырехугольной пирамиде провести сечение, проходящее через диагональ основания и середину бокового ребра.

8. В правильной четырехугольной пирамиде провести сечение, проходящее через диагональ основания и наклоненное к плоскости основания под углом 30°.

9. В правильной четырехугольной пирамиде провести сечение, параллельное основанию и проходящее через середину бокового ребра.

10. В правильной четырехугольной пирамиде провести сечение, проходящее через вершину пирамиды и перпендикулярное плоскости основания.

11. В правильной четырехугольной пирамиде провести сечение, проходящее через одну из сторон основания и середину высоты.

12. Основание четырехугольной пирамиды — ромб. Вершина пирамиды проектируется в центр симметрии ромба. Провести сечение, проходящее через высоту основания, опущенную из тупого угла ромба, и боковое ребро, которое проходит через эту же вершину.

13. Основание пирамиды — ромб. Вершина пирамиды проектируется в вершину острого угла ромба. Провести сечение, проходящее через вершину пирамиды и высоту ромба, опущенную из тупого угла.

14. В прямоугольном параллелепипеде провести диагональное сечение.

15. В прямоугольном параллелепипеде провести сечение, проходящее через сторону нижнего основания и противоположную сторону верхнего основания.

16. В прямой четырехугольной призме провести сечение, проходящее через диагональ нижнего основания и одну из вершин верхнего основания.

17. В прямой четырехугольной призме провести сечение, проходящее через сторону нижнего основания под углом 30° к основанию.

18. В правильной шестиугольной призме провести сечение, проходящее через одну из сторон нижнего основания и противоположную ей сторону верхнего основания.

19. В прямоугольном параллелепипеде построить сечение, проходящее через одну из сторон нижнего основания и одну из вершин верхнего.

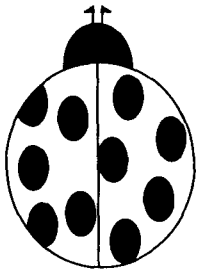
20. В прямоугольном параллелепипеде построить сечение, проходящее через одно из его ребер и точку пересечения диагоналей противолежащей этому ребру грани.

21. В правильной шестиугольной пирамиде построить сечение, проходящее через вершину и большую диагональ основания.

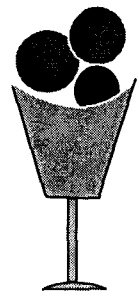
22. В прямом цилиндре построить осевое сечение.

23. В правильной шестиугольной призме построить сечение, проходящее через большую диагональ нижнего основания и одну из сторон верхнего.

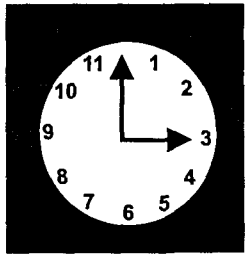
24. Изобразить на экране ЭВМ следующий рисунок:



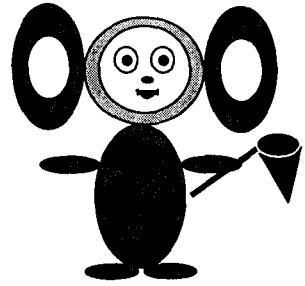
1)



2)

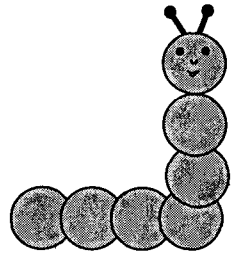


3)

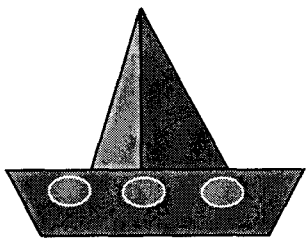


4)

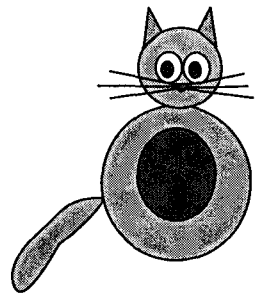
5)



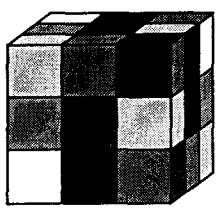
6)



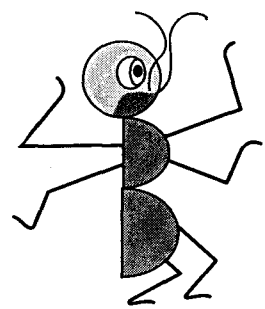
7)



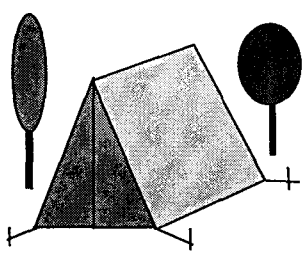
8)



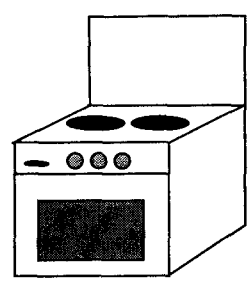
9)



10)

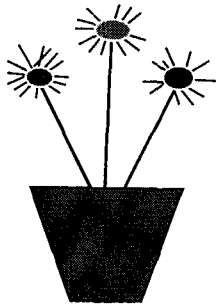


11)

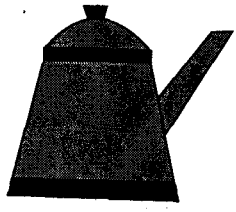


12)

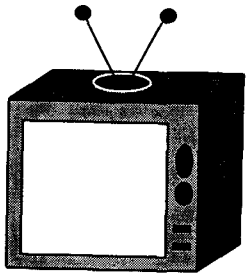
13)



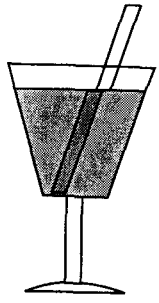
14)



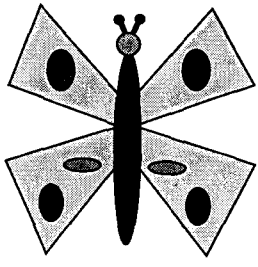
15)



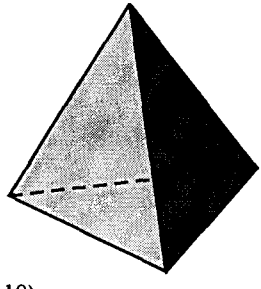
16)



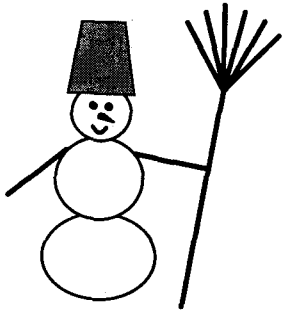
17)



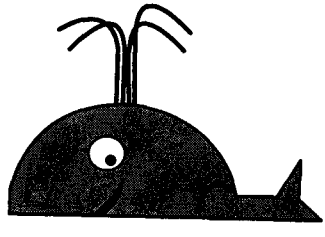
18)

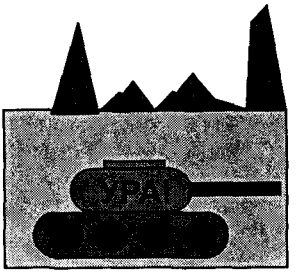


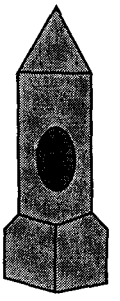
19)

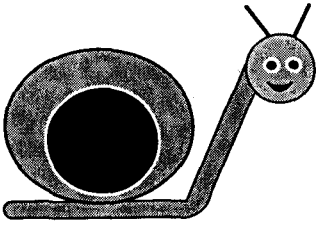


20)

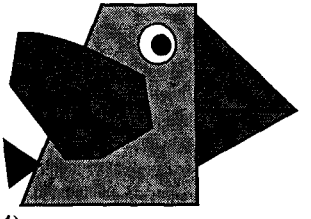


21) 

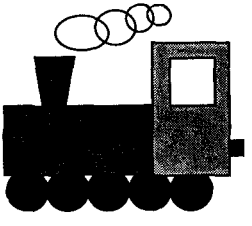
22) 



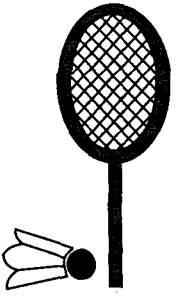
23)



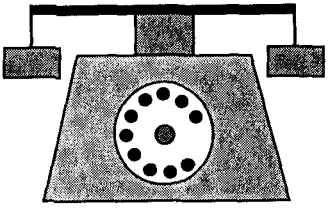
24)



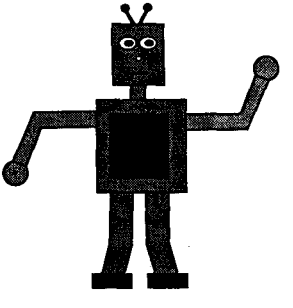
25)



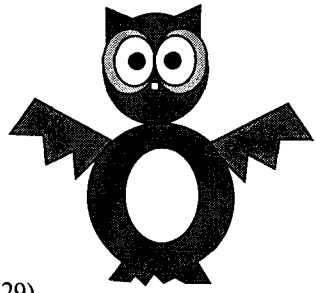
26)



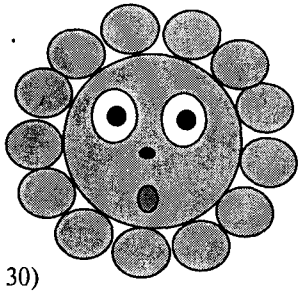
27)



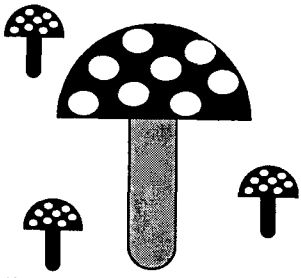
28)



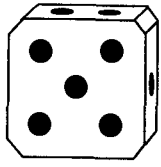
29)



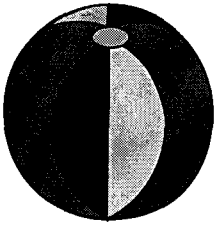
30)



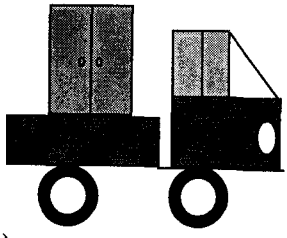
31)



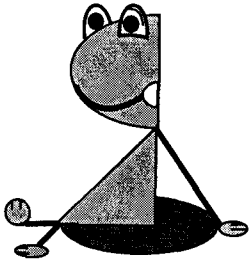
32)



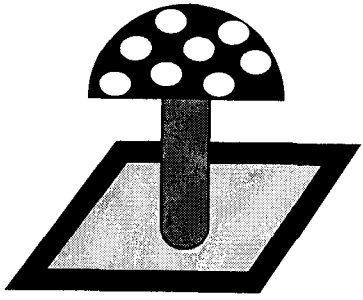
33)



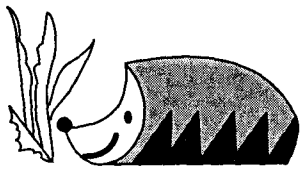
34)



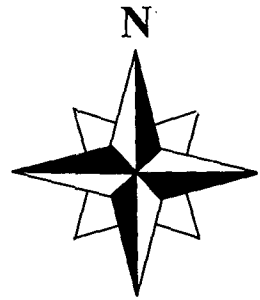
35)



36)

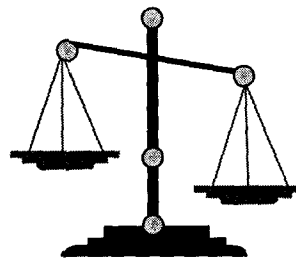


37)



38)

39)



40)

25. Построить графики функций разд. 6.2.2 и 6.4.

6.17. Задачи по теме «Объектно-ориентированное программирование»

1. Построить систему классов для описания плоских геометрических фигур: круга, квадрата, прямоугольника. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и поворота на заданный угол.

2. Построить описание класса, содержащего информацию о почтовом адресе организации. Предусмотреть возможность раздельного изменения составных частей адреса, создания и уничтожения объектов этого класса.

3. Составить описание класса для представления комплексных чисел с возможностью задания вещественной и мнимой частей как числами типов `double`, так и целыми числами. Обеспечить выполнение операций сложения, вычитания и умножения комплексных чисел.

4. Составить описание класса для работы с цепными списками строк (строки произвольной длины) с операциями включения в

список, удаления из списка элемента с заданным значением данного, удаления всего списка или конца списка, начиная с заданного элемента.

5. Составить описание класса для объектов-векторов, задаваемых координатами концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.

6. Составить описание класса прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменения размеров, построения наименьшего прямоугольника, содержащего два заданных прямоугольника, и прямоугольника, являющегося общей частью (пересечением) двух прямоугольников.

7. Составить описание класса для определения одномерных массивов целых чисел (векторов). Предусмотреть возможность обращения к отдельному элементу массива с контролем выхода за пределы индексов, возможность задания произвольных границ индексов при создании объекта и выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов, умножения и деления всех элементов массива на скаляр, печати (вывода на экран) элементов массива по индексам и всего массива.

8. Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы индексов, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, печать (вывод на экран) элементов массива и всего массива.

9. Составить описание класса многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, печать (вывод на экран) описания многочлена.

10. Составить описание класса одномерных массивов строк, каждая строка которых задается длиной и указателем на выделенную для нее память. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы индексов, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, печать (вывод на экран) элементов массива и всего массива.

11. Составить описание объектного типа `TMatr`, обеспечивающего размещение матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы.

12. Простые и иерархические меню.

а) Спроектировать простое меню в одной строке экрана. Меню обеспечивает перебор пунктов в результате нажатия на клавишу **Пробел**, позволяет зафиксировать выбор нажатием на клавишу **Enter** или отказаться от выбора нажатием на клавишу **Esc**. После выбора одного из пунктов в программу возвращается какое-то значение, связанное с выбранным пунктом, например символ. При отказе от выбора в программу возвращается `#27`.

Перед началом работы меню ему надо передать названия пунктов и возвращаемые символы (ими могут быть первые буквы пунктов или какие-то специальные символы). Все это можно сделать в форме строки вида

«Первое Второе Третье»

или

«Первое (a) Второе (b) Третье (c)» (здесь за названием пункта следует в скобках возвращаемый символ).

Состояние меню характеризуется координатами меню на экране, номером отмеченного пункта, общим количеством пунктов, перечнем названий пунктов и возвращаемых символов (во втором варианте представления).

Методами объекта являются:

`Init` — заполняет поле названий пунктов, подсчитывает количество пунктов, делает выбранным первый пункт;

`Select` — позволяет выбрать пункт меню и возвращает символ выбранного пункта, при отказе от выбора возвращает `#27`;

`Draw` — рисует меню, выделяя выбранный пункт цветом;

`LeftBoard` — возвращает начало названия данного пункта;

`Len` — возвращает длину названия пункта;

`WhatSel` — возвращает символ выбранного пункта.

б) Создать новый объект `TNeatMenu`, наследующий `TMenu`, который, в отличие от своего предка, будет восстанавливать вид экрана. Для этого нужно добавить новое поле `Store`, где будет храниться прежний экран во время действия меню, перекрыть метод `Init` и добавить метод `Done`, который восстанавливает состояние экрана.

в) Создать меню, которое изображает себя в форме столбца. Для этого рационально воспользоваться виртуальными методами. Достаточно изменить метод `Draw` объекта `TNeatMenu` и объявить одноименные методы виртуальными.

г) Разместить объекты в динамической памяти, для этого достаточно описать указатели на них.

д) Построить сложное иерархическое меню: пробел будет открывать главное меню, последовательное нажатие на клавиши

Enter и **Пробел** будет разворачивать подсвеченный пункт в подменю или, если пункт находится на нижнем уровне, клавиша **Enter** будет сворачивать подменю. Нажатие на клавишу **Esc** заканчивает работу программы.

е) Построить иерархическое меню: пробел будет открывать главное меню, нажатие на клавишу **Enter** будет разворачивать подсвеченный пункт в меню или, если пункт находится на самом нижнем уровне, клавиша **Enter** сворачивает подменю. Нажатие на клавишу **Esc** заканчивает работу программы. Нижний уровень — вертикальный.

13. Составить программу, работающую со связанными списками. Мы будем рассматривать связанный список как объект, содержащий связанный список данных и операций (методов), которые вы можете с ними выполнять. Связанный список данных состоит из указателей на начало («голову») и конец («хвост») связанного списка (в нашем примере из-за его гибкости используется двунаправленный связанный список). Каждый элемент связанного списка представляет собой реализацию отдельного объекта. Возможности, необходимые для использования связанного списка, предоставляют следующие операции:

- создание связанного списка (выделение для него памяти);
- уничтожение связанного списка (освобождение используемой памяти);
- инициализация связанного списка;
- деинициализация связанного списка;
- вставка элемента в середину списка перед существующим элементом;
- присоединение элемента к концу связанного списка;
- удаление элемента из связанного списка;
- возвращение первого элемента связанного списка;
- возвращение последнего элемента связанного списка.

Необходимо иметь в виду, что создание и инициализация, а также уничтожение и деинициализация методов — это не синонимы. При создании и уничтожении методы `create` и `destroy` выделяют и освобождают память для объекта (связанного списка), а методы инициализации и деинициализации `initialize` и `deinitialize` только инициализируют и деинициализируют ранее выделенные экземпляры объекта. Вы можете видеть, как объект связанного списка наследуется объектами стека или очереди, поскольку очередь и стек можно реализовать как связанный список с ограниченным числом операций. Например, можно реализовать очередь в виде связанного списка, в котором элементы могут добавляться к концу и извлекаться из начала. Если вы таким образом реализуете очередь, то нужно запретить наследуемые методы связанного списка, которые для очереди недопустимы (например, вставку в середину списка).

14. Определить объект TFish — аквариумная рыбка. Рыбка имеет координаты, скорость, размер, цвет, направление движения. Методами объекта являются:

• Init — устанавливает значения полей объекта и рисует рыбу на экране методом Draw.

• Draw — рисует рыбу в виде уголка с острием в точке Coord и направленного острием по ходу движения рыбы.

• Look — проверяет несколько точек на линии движения рыбы. Если хоть одна из них отличается по цвету от воды, возвращаются ее цвет и расстояние до рыбы.

• Run — перемещает рыбу в текущем направлении на расстояние, зависящее от текущей скорости рыбы. Иногда случайным образом меняет направление движения рыбы. Если рыба видит препятствие, направление движения меняется, пока препятствие не исчезнет из поля зрения рыбы.

15. Определить объект TAquarium, который является местом обитания рыб (см. задачу 14 данного раздела). Он представляет собой область экрана, наполненную водой. Рыбы живут в аквариуме, поэтому экземпляры объекта TFish должны быть полями объекта TAquarium.

Методы:

• Init — включает графический режим, заполняет аквариум водой, камнями и рыбами.

• Run — организует бесконечный цикл, в котором выполняется метод Run всех обитателей аквариума.

• Done — выключает графический режим.

16. Определить два объекта TPike и TKarp, которые наследуют объект Tfish (см. задачу 14). Оба они отличаются от TFish тем, что по-разному изображают себя на экране: TPike — в виде зеленой стрелки, а TKarp — в виде красного треугольника. Воспользуйтесь виртуальными методами. Для этого вернитесь к определению TFish и откорректируйте его, сделав Draw пустым и виртуальным.

17. Объединить карпов и щук (см. задачу 16) в две стаи. Стая — это связанный список рыб в динамической памяти. Для связи добавьте в объекты TPike и TKarp поле Next — указатель на следующую рыбу в стае. Сделайте аквариум владельцем не отдельных рыб, а двух стай и позвольте пользователю пополнять стаи, вводя рыб с клавиатуры.

18. Позволить щукам (см. задачу 16) проявить свой дурной характер и поедать карпов, как только они их увидят. Здесь возникнет проблема — установить, какого именно карпа видит щука. Она решается путем просмотра всей стаи карпов и поиска того, чьи координаты близки к координатам данной щуки. Найденный карп удаляется из стаи.

19. Составить программу для игры в шашки. Шашка каждого нового цвета выступает в качестве отдельного объекта. Характери-

стики шашки — цвет и позиция на доске. Методы — перемещение. Не забудьте о таких объектах, как «дамки».

20. Составить программу для игры в домино. В качестве объектов выступают кости домино. Методы — способы выставления той или иной кости.

21. Составить программу для игры в шахматы. Каждая уникальная шахматная фигура выступает в качестве отдельного объекта. Она характеризуется цветом, положением на доске, способом перемещения. Предусмотреть возможность превращения пешки в ферзя.

22—25. Задачи 18—21 в графическом режиме.

26—35. Задачи I—X из раздела «Задачи по теме "Модули"».

6.18. Большие проектные задания

1. Волчий остров (Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. — М.: Мир, 1981)

Волчий остров размером 20×20 заселен дикими кроликами, волками и волчицами. Имеется по нескольку представителей каждого вида. Кролики довольно глупы: в каждый момент времени они с одинаковой вероятностью 1/9 передвигаются в один из восьми соседних квадратов (за исключением участков, ограниченных береговой линией) или просто сидят неподвижно. Каждый кролик с вероятностью 0,2 превращается в двух кроликов. Каждая волчица передвигается случайным образом, пока в одном из соседних восьми квадратов не окажется кролик, за которым она охотится. Если волчица и кролик оказываются в одном квадрате, волчица съедает кролика и получает одно очко. В противном случае она теряет 0,1 очка. Волки и волчицы с нулевым количеством очков умирают.

В начальный момент времени все волки и волчицы имеют 1 очко. Волк ведет себя подобно волчице до тех пор, пока в соседних квадратах не исчезнут все кролики; тогда, если волчица находится в одном из восьми близлежащих квадратов, волк гонится за ней. Если волк и волчица окажутся в одном квадрате и там нет кролика, которого нужно съесть, они производят потомство случайного пола.

Запрограммировать предполагаемую экологическую модель и понаблюдать за изменением популяции в течение некоторого периода времени.

2. Задача об инфекции стригущего лишая (Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. — М.: Мир, 1981)

Промоделировать процесс распространения инфекции — стригущего лишая по участку кожи размером $n \times n$ (n — нечетное) клеток. Предполагается, что исходной зараженной клеткой кожи

является центральная. В каждый интервал времени пораженная инфекцией клетка может с вероятностью 0,5 заразить любую из соседних здоровых клеток. По прошествии шести единиц времени зараженная клетка становится невосприимчивой к инфекции, возникший иммунитет действует в течение последующих четырех единиц времени, а затем клетка оказывается здоровой. В ходе моделирования описанного процесса выдавать текущее состояние моделируемого участка кожи в каждом интервале времени, отмечая зараженные, невосприимчивые к инфекции и здоровые клетки.

3. Построение с помощью циркуля и линейки

Составить программу, автоматизирующую процесс построения фигур на плоскости с помощью циркуля и линейки. Программа должна уметь выполнять следующие команды:

- отметить произвольную точку и обозначить ее;
- построить прямую, проходящую через две точки;
- построить произвольную прямую;
- построить окружность с заданным центром данного радиуса;
- построить и обозначить точку пересечения двух линий.

Программа должна содержать 10 — 15 стандартных задач на построение школьного курса геометрии, предлагать их для решения и контролировать процесс построения и полученное решение.

4. «Морской бой»

Составить программу для игры в морской бой игрока с компьютером. Программа должна позволять расставлять корабли на поле 10×10, контролировать правильность их расстановки, давать противникам возможность поочередно делать ходы и выдавать соответствующие информационные сообщения. Когда в качестве одного из игроков выступает компьютер, программа должна анализировать предыдущие ходы и следующий делать на основе проведенного анализа.

5. Обучающе-контролирующая программа «Сложение и вычитание отрицательных чисел»

Составить программу, обучающую учащихся 6 класса сложению и вычитанию отрицательных чисел (см. учебник «Математика 6» Нурка или Виленкина), а также предлагающую серию заданий различной сложности для закрепления навыков действий над такими числами.

6. «Математико» (итальянская игра) (Б. А. Кордемский. Математическая смекалка. — СПб.: Манускрипт, 1994)

Имеется набор из 52 карточек, на которых записаны числа от 1 до 13, причем карточки с каждым из этих чисел встречаются четырежды. Разработать программу, которая позволяет имитировать игру человека с компьютером. Имеется квадратное поле с 25 клетками. Программа случайным образом извлекает какую-либо из имеющихся карточек и выдает записанное на ней чис-

ло. Каждый игрок заносит это число в одну из клеток квадрата. Так продолжается до тех пор, пока не будут заполнены все клетки квадрата (рис. 55).

По окончании игры заполнение соответствующего квадрата оценивается определенным количеством очков. Цель игры — разместить числа в клетках так, чтобы набрать наибольшее количество очков в соответствии с данной таблицей:

1	1	7	1	7	(80)
2	10	2	13	2	(40)
5	12	13	5	7	(10)
3	3	3	11	3	(160)
4	12	4	13	12	(20)
(20)	(50)	(10)	(10)	(10)	(160)

Рис. 55

Комбинации чисел	В ряду или столбце	По диагонали
За 2 одинаковых числа	10 очков	20 очков
За 2 пары одинаковых чисел	20 очков	30 очков
За 3 одинаковых числа	40 очков	50 очков
За 3 одинаковых числа и два других одинаковых числа	80 очков	90 очков
За 4 одинаковых числа	160 очков	170 очков
За 5 последовательных чисел, но необязательно по порядку расположенных	50 очков	60 очков
За три раза по 1 и два раза по 13	100 очков	110 очков
За числа 1, 13, 12, 11 и 10, но необязательно по порядку расположенных	150 очков	160 очков
За 4 единицы	200 очков	210 очков

Разработать для компьютера оптимальную стратегию заполнения квадрата.

7. Заполнение готовых форм с помощью информации из базы данных

Имеется база данных, содержащая сведения о некоторой группе людей (каждая запись содержит до 10 полей). Составить программу, которая, используя сведения из базы данных, позволяет заполнять некоторые документы (стандартные письма, приглашения, визитки, отчеты и т.д.), вписывая эти сведения в нужные места в указанных документах в соответствующих падежах, лицах, временах и т.д.

8. Карточные игры

Составить программу, которая раздает игральные карты заданному количеству игроков (одним из игроков является человек, за остальных играет компьютер) и моделирует игру в «дурака». Компьютерная программа играет случайным образом, без анализа уже вышедших карт. Количество игроков не превышает шести.

9. «Крестики-нолики»

Составить программу, позволяющую играть на бесконечном поле в «крестики-нолики»:

а) игроку с компьютером;

б) двум игрокам.

Если в качестве игрока выступает компьютер, программа делает первый ход. Делая очередной ход, программа анализирует ситуацию, рассчитывая возможные ходы противника вперед на 1—2 хода, и в результате проведенного анализа поступает оптимальным образом.

10. «Быки и коровы»

Составить программу, позволяющую играть в «Быки и коровы»:

а) игроку с компьютером;

б) двум игрокам.

Каждый из противников задумывает четырехзначное число, все цифры которого различны (первая цифра числа отлична от нуля). Необходимо разгадать задуманное число. Выигрывает тот, кто отгадает первый. Противники по очереди называют друг другу числа и сообщают о количестве «быков» и «коров» в названном числе («бык» — цифра есть в записи задуманного числа и стоит в той же позиции, что и в задуманном числе; «корова» — цифра есть в записи задуманного числа, но не стоит в той же позиции, что и в задуманном числе).

Например, если задумано число 3275 и названо число 1234, получаем в названном числе одного «быка» и одну «корову». Очевидно, что число отгадано в том случае, если имеем 4 «быка».

11. «Числовые головоломки»

Составить программу, которая:

1) предлагает игроку числовые головоломки типа **ОДИН + ОДИН= = МНОГО** из некоторого набора таких головоломок (до 30);

2) позволяет решить головоломку;

3) контролирует правильность решения.

12. «Графики»

Составить программу, которая предлагает пользователю некоторый список функций для построения графиков (например, $y = ax^2 + bx + c$, $y = a\sin x + b$ и т.д. — до 25 наименований). После выбора соответствующей функции, задания коэффициентов и отрезка, на котором выполняется построение, программа строит указанный график. Затем значение коэффициентов и положение графика можно менять (например, с помощью клавиш управле-

ния курсором), после чего график перестраивается и записывается обновленное уравнение соответствующей кривой.

13. Игра «Две лисы и 20 кур»

На поле указанной формы (рис. 56) находятся две лисы и 20 кур. Куры могут перемещаться на один шаг вверх, влево или вправо, но не назад и не по диагонали. Лисы также могут перемещаться только на один шаг (вверх, вниз, влево и вправо).

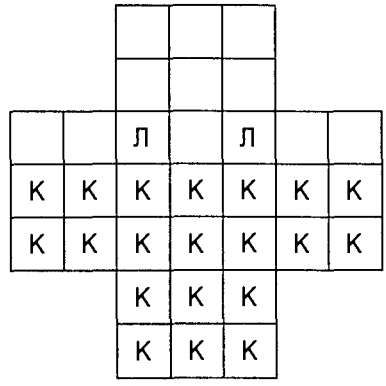


Рис. 56

Лиса может съесть курицу, как в игре в шашки: если в горизонтальном или вертикальном направлении за курицей на один шаг следует свободное поле, то лиса перепрыгивает через курицу и берет ее. Лисы всегда обязаны есть, и, когда у них есть выбор, они обязаны осуществлять «наиболее длинное поедание». Если два приема пищи имеют одинаковую длину, осуществляется один из них — по выбору лисы.

Составить программу, которая играет за лис. Игрок перемещает кур. Партнеры играют по очереди, причем куры начинают. Они выигрывают партию, если девяти из них удается занять 9 полей, образующих верхний квадрат поля.

Начальное положение кур и лис изображено на рис. 55.

Лисы выигрывают, если им удается съесть 12 кур, так как тогда оставшихся кур недостаточно, чтобы занять 9 верхних полей.

14. Программа «Игры со спичками»

Составить программу, которая:

1) предлагает игроку головоломки со спичками из некоторого набора таких головоломок (до 30 штук);

2) позволяет решить головоломку, передвигая спички;

3) контролирует правильность решения.

15. Графика в Турбо Паскале

Составить программу, демонстрирующую все графические возможности Турбо Паскаля и обучающую работе с основными графическими процедурами и функциями. Программа должна проконтролировать усвоение изученного материала (в виде теста или в какой-либо другой форме).

16. Игра в слова

Составить программу, позволяющую компьютеру и человеку играть в слова. Предварительно программа объясняет правила игры и позволяет уточнить их в любой момент.

Тематикой игры могут быть по выбору города, животные, растения и т.д. Тематику из предложенных компьютером (не менее 5)

выбирает человек. Для игры компьютер использует собственную базу данных (для каждой тематики свою), хранящуюся в виде текстового файла. Если названное человеком слово отсутствует в базе, уточняется, правильно ли оно названо, и в случае правильности заносится в базу. Правила игры: первый игрок называет слово, а второй должен предложить другое, начинающееся с той буквы, на которую оканчивается названное.

17. Ребусы

Выбрав какой-либо школьный предмет (информатика, математика и т.д.), подобрать ребусы по нему и предложить их для решения. Программа должна позволять выбрать тот или иной ребус, проконтролировать его решение и подвести итоги при завершении работы.

ПРИЛОЖЕНИЯ

Приложение 1

Турбо Паскаль. Модуль CRT

Таблица П1.1. Константы режимов работы

Имя константы	Номер режима	Режим
BW40	0	Черно-белый, 40 символов, 25 строк
CO40	1	Цветной, 40×25
BW80	2	Черно-белый, 80×25
CO80	3	Цветной, 80×25
Mono	7	Монохромный, 80×25, для монохромных дисплеев

Таблица П1.2. Константы цветов

Имя константы	Номер цвета	Цвет
Black	0	Черный
Blue	1	Темно-синий
Green	2	Темно-зеленый
Cyan	3	Бирюзовый
Red	4	Красный
Magenta	5	Фиолетовый
Brown	6	Коричневый
LightGray	7	Светло-серый
DarkGray	8	Темно-серый
LightBlue	9	Синий
LightGreen	10	Светло-зеленый
LightCyan	11	Светло-бирюзовый
LightRed	12	Розовый
LightMagenta	13	Малиновый
Yellow	14	Желтый
White	15	Белый
Blink	128	Мерцание символа

Таблица П1.3. Процедуры и функции

Интерфейс	Назначение
Установка режимов и окон	
`Procedure` **`AssignCrt`**`(File: text);`	Связывает окно дисплея с текстовым файлом, что позволяет ускорить вывод на экран
`Procedure` **`ClrScr`**`;`	Очищает экран и помещает курсор в верхний левый угол
`Procedure` **`TextMode`**`(Mode: integer);` `Mode` — номер текстового режима или соответствующая константа	Выбор текстового режима
`Procedure` **`Window`**`(x1,y1,x2,y2: byte);` `(x1,y1)` и `(x2,y2)` — координаты верхнего левого и нижнего правого углов окна	Определяет окно вывода в текстовом режиме
Управление цветом текста и фона	
`Procedure` **`HighVideo`**`;`	Устанавливает высокую яркость выводимых символов
`Procedure` **`LowVideo`**`;`	Устанавливает низкую яркость выводимых символов
`Procedure` **`NormVideo`**`;`	Возвращает цвет символов и фона, свойственный данному графическому режиму по умолчанию
`Procedure` **`TextBackground`** `(Color: byte);` `Color` — код цвета или соответствующая константа	Выбор цвета символов
Управление выводом текста	
`Procedure` **`ClrEol`**`;`	Стирает все символы от текущей позиции курсора до конца строки
`Procedure` **`DelLine`**`;`	Удаляет линию, в которой находится курсор
`Procedure` **`InsLine`**`;`	Вставляет новую строку текста перед строкой, где находится курсор

Интерфейс	Назначение
Работа с клавиатурой	
`Function` **`KeyPressed`**`: boolean;` Значение `True`, если нажата любая клавиша, и `False` — если нет	Определяет, была ли нажата клавиша на клавиатуре
`Function` **`ReadKey`**`: char;` Значение функции — код символа клавиши, нажатой на клавиатуре	Считывает символы из буфера клавиатуры
Управление курсором	
`Procedure` **`GotoXY`** `(X,Y: integer);` `X,Y` — координаты курсора	Перемещает курсор в указанные координаты окна вывода
`Function` **`WhereX`**`: integer;` Значение функции — координата `X` курсора	Возвращает текущую координату *X* курсора
`Function` **`WhereY`**`: integer;` Значение функции — координата `Y` курсора	Возвращает текущую координату *Y* курсора
Управление звуком	
`Procedure` **`NoSound`**`;`	Выключает динамик
`Procedure` **`Sound`**`(Hz: word);` `Hz` — частота звука в герцах	Включает звук динамика с заданной тональной частотой
Управление временем	
`Procedure` **`Delay`**`(Ms: word);` `Ms` — значение задержки в миллисекундах	Задержка исполнения программы на заданное число миллисекунд

Приложение 2

Турбо Паскаль. Модуль GRAPH

Таблица П2.1. Коды драйверов графических устройств

Имя	Значение	Назначение
Detect	0	Автоматический выбор драйвера
CGA	1	
MCGA	2	
EGA	3	
EGA64	4	
EGAMono	5	
IBM8514	6	
HercMono	7	
ATT400	8	
VGA	9	
PC3270	10	
CurrentDriver	−128	Текущий драйвер

Таблица П2.2. Константы графических режимов

Имя	Значение	Размер поля	Палитра	Число страниц
ATT400C0	0	320×200	C0	1
ATT400C1	1	320×200	C1	1
ATT400C2	2	320×200	C2	1
ATT400C3	3	320×200	C3	1
ATT400Med	4	640×200	2 цвета	1
ATT400Hi	5	640×400	2 цвета	1
CGAC0	0	320×200	C0	1
CGAC1	1	320×200	C1	1
CGAC2	2	320×200	C2	1
CGAC3	3	320×200	C3	1
CGACHi	4	640×200	2 цвета	1
EGALo	0	640×200	16 цветов	4
EGAHi	1	640×350	16 цветов	2
EGA64Lo	0	640×200	16 цветов	1
EGA64Hi	1	640×350	4 цвета	1
EGAMonoHi	0	640×350	2 цвета	1 или 2

Имя	Значение	Размер поля	Палитра	Число страниц
HercMonoHi	0	720×348	2 цвета	2
IBM8514Lo	0	640×480	256 цветов	1
IBM8514Hi	0	1024×768	256 цветов	1
MCGAC0	0	320×200	C0	1
MCGAC1	1	320×200	C1	1
MCGAC2	2	320×200	C2	1
MCGAC3	3	320×200	C3	1
MCGAMed	4	320×200	2 цвета	1
MCGAHi	5	640×480	2 цвета	1
PC3270Hi	0	720×350	2 цвета	1
VGALo	0	640×200	16 цветов	4
VGAMed	1	640×350	16 цветов	2
VGAHi	2	640×480	16 цветов	1

Примечание. Палитра C0 включает в себя следующие цвета: светло-зеленый, розовый и желтый; палитра C1 — светло-голубой, светло-фиолетовый и белый; палитра C2 — зеленый, красный и коричневый; палитра C3 — голубой, фиолетовый и светло-серый.

Таблица П2.3. Коды цветов

Имя константы	Номер цвета	Цвет
Black	0	Черный
Blue	1	Темно-синий
Green	2	Темно-зеленый
Cyan	3	Бирюзовый
Red	4	Красный
Magenta	5	Фиолетовый
Brown	6	Коричневый
LightGray	7	Светло-серый
DarkGray	8	Темно-серый
LightBlue	9	Синий
LightGreen	10	Светло-зеленый
LightCyan	11	Светло-бирюзовый
LightRed	12	Розовый
LightMagenta	13	Малиновый
Yellow	14	Желтый
White	15	Белый

Таблица П2.4. Коды линий

Имя	Значение	Назначение
Коды типов линий (для процедуры `SetLineStyle`*)*		
`SolidLn`	0	Сплошная
`DottedLn`	1	Пунктирная
`CenterLn`	2	Штрихпунктирная
`DashedLn`	3	Штриховая
`UserBitLn`	4	Заданная пользователем
Коды толщины линии		
`NormWidth`	1	Нормальная
`ThickWidth`	3	Толстая

Таблица П2.5. Константы орнамента заполнения (для процедуры `SetFillStyle`)

Имя	Значение	Назначение
`EmptyFill`	0	Заполнение цветом фона
`SolidFill`	1	Однородное заполнение цветом
`LineFill`	2	Заполнение —
`LtSlashFill`	3	Заполнение ///
`SlashFill`	4	Заполнение /// толстыми линиями
`BkSlashFill`	5	Заполнение \\\толстыми линиями
`LtBkSlashFill`	6	Заполнение \\\
`HatchFill`	7	Заполнение клеткой
`XHatchFill`	8	Заполнение косой клеткой
`InterleaveFill`	9	Заполнение частой клеткой
`WideDotFill`	10	Заполнение редкими точками
`CloseDotFill`	11	Заполнение частыми точками
`UserFill`	12	Определяется пользователем

Таблица П2.6. Процедуры и функции

Интерфейс	Назначение
Управление графическим режимом	
`Procedure` **`CloseGraph`**`;`	Закрывает графический режим
`Procedure` **`DetectGraph`**`(var grDriver, grMode: integer);` `GrDriver` — код драйвера, `grMode` — код графического режима	Определяет рекомендуемые к применению для данного компьютера графические драйвер и режим
`Function` **`GetDriverName`**`: string;` Значение функции — имя используемого драйвера	Определяет имя файла с используемым графическим драйвером
`Function` **`GetGraphMode`**`: integer;` Значение функции — код графического режима	Определяет код используемого графического режима
`Function` **`GetMaxName`**`: integer;` Значение функции — имя используемого графического режима	Определяет имя используемого графического режима
`Function` **`GetMaxMode`**`: integer;` Значение функции — максимальное значение кода режима	Определяет максимальное значение кода графического режима для используемого драйвера
`Procedure` **`GetModeRange`**`(grDriver: integer: var LoMode, HiMode: integer);` `GrDriver` — код графического режима; `LoMode, HiMode` — наименьшее и наибольшее значение кода графического режима для данного драйвера	Определяет минимальное и максимальное значение кода графического режима для указанного при обращении драйвера
`Procedure` **`GraphDefaults`**`;`	Устанавливает графический указатель в начало координат и переустанавливает графическую систему
`Function` **`GraphErrorMsg`**`(ErrorCode: integer): string;` `ErrorCode` — код графической ошибки. Значение функции — текстовое сообщение о характере ошибки	Выдает строку — сообщение об ошибке графического режима по коду ошибки
`Function` **`GraphResult`**`: integer;` Значение функции — код ошибки	Определяет, произошла ли ошибка при исполнении процедур модуля

Интерфейс	Назначение
`Procedure` **`InitGraph`**`(var grDriver, grMode: integer; PathToDriver: string);` `GrDriver` — код драйвера, `grMode` — код графического режима; `PathToDriver` — путь к файлу используемого драйвера	Устанавливает заданный графический режим
`Function` **`InstallUserDriver`**`(Name: string; AutoDetectPtr: Pointer): integer;` `Name` — имя файла графического драйвера; `AutoDetectPtr` — указатель на процедуру, определяющую успешность запуска драйвера. Значение функции — цифровой код установленного драйвера	Инсталлирует пользовательский драйвер графического режима
`Procedure` **`RegisterBGIDriver`**`(Driver: Pointer): integer;` `Driver` — указатель на драйвер	Регистрирует драйвер графической системы
`Procedure` **`RestoreCrtMode`**`;`	Закрывает графический режим и восстанавливает текстовый режим, установленный ранее
`Procedure` **`SetGraphMode`**`(grMode: integer);` `grMode` — код графического режима	Устанавливает другой графический режим без изменения драйвера
Управление экраном и окнами	
`Procedure` **`ClearDevice`**`;`	Очищает экран, сбрасывает все графические установки к значениям по умолчанию, устанавливает графический указатель в положение (0, 0)
`Procedure` **`ClearViewPort`**`;`	Очищает экран, устанавливая фон, заданный в SetBkColor
`Function` **`GetMaxX`**`: integer;` Значение функции — максимальная координата `X`	Определяют максимальные координаты экрана в данном графическом режиме
`Function` **`GetMaxY`**`: integer;` Значение функции — максимальная координата `Y`	

Интерфейс	Назначение
`Procedure` **`GetAspectRatio`**`(var Xasp, Yasp: word);` `Xasp, Yasp` — коэффициенты по осям `X` и `Y`	Определяет коэффициенты, характеризующие неодинаковость линейного расстояния между пикселями по осям *X* и *Y*
`Function` **`GetBkColor`**`: word;` Значение функции — код цвета фона	Определяет установленный цвет фона
Экран в целом и окна	
`Procedure` **`GetViewSettings`**`(var ViewPort: ViewPortType);` `ViewPort` — параметры текущего окна	Запрашивает текущие параметры окна и отсечения
`Procedure` **`SetBkColor`**`(Color: word);` Значение функции — код цвета фона	Устанавливает цвет фона
`Procedure` **`SetActivePage`**`(Page: word);` `Page` — номер активной страницы	Устанавливает активную графическую страницу
`Procedure` **`SetAspectRatio`**`(var Xasp, Yasp: word);` `Xasp, Yasp` — коэффициенты по осям `X` и `Y`	Изменяет масштабный коэффициент отношения сторон экрана
`Procedure` **`SetVisualPage`**`(Page: word);` `Page` — номер страницы	Устанавливает номер видимой графической страницы
`Procedure` **`SetViewPort`**`(x1,y1,x2,y2: integer; Clip: boolean);` `(x1,y1)` и `(x2,y2)` — координаты противоположных углов окна; `Clip` определяет, отсекать ли изображение за пределами окна или нет	Устанавливает визуальный порт (окно) для вывода графического изображения
`Function` **`GetColor`**`: word;` Значение функции — код цвета	Возвращает текущий цвет (установленный SetColor)
`Procedure` **`GetDefaultPalette`**`(var Palette: PaletteType);` `Palette` — палитра (массив типа `PaletteType`)	Определяет, какая палитра действует в режиме по умолчанию
`Function` **`GetMaxColor`**`: word;` Значение функции — максимальный код цвета	Определяет максимальный код цвета в установленном режиме
`Procedure` **`GetPalette`**`(var Palette: PaletteType);` `Palette` — палитра (массив типа `PaletteType`)	Определяет, какая палитра установлена

Интерфейс	Назначение
`Function` **`GetPaletteSize`**`: word;` Значение функции — количество цветов в палитре	Возвращает размер таблицы палитры
`Procedure` **`SetAllPalette`**`(var Palette: PaletteType);` `Palette` — палитра (массив типа `PaletteType`)	Устанавливает все цвета палитры
`Procedure` **`SetColor`**`(Color: word);` `Color` — код цвета	Устанавливает цвет для линий
`Procedure` **`SetPalette`**`(ColorNum, Color: word);` `ColorNum` — код заменяемого цвета; `Color` — код нового цвета	Устанавливает один новый цвет в палитре
`Procedure` **`SetRGBPalette`**`(ColorNum, RedValue, GreenValue, BlueValue: word);` `ColorNum` — код устанавливаемого в палитре цвета; `RedValue, GreenValue, BlueValue` — интенсивность красного, зеленого и синего для этого цвета	Устанавливает в палитре цвет, заданный тремя компонентами: красным, зеленым и синим
Управление указателем	
`Function` **`GetX`**`: integer;` `GetX` — координата `X` указателя	Возвращает координату *X* текущего указателя
`Function` **`GetY`**`: integer;` `GetY` — координата `Y` указателя	Возвращает координату *Y* текущего указателя
`Procedure` **`MoveTo`**`(X: integer; Y: integer);` `(X,Y)` — координаты точки экрана	Перемещает указатель в точку, заданную координатами *X, Y*
`Procedure` **`MoveRel`**`(dx: integer; dy: integer);` `dx,dy` — вектор приращений координат точки экрана	Перемещает указатель в точку, координаты которой отличаются от текущих координат на величину *dx*, *dy*
Шаблоны, закраска областей	
`Procedure` **`FloodFill`**`(X,Y: integer; ColorBorder: word);` `(X,Y)` — координаты точки, вокруг которой идет закраска; `ColorBorder` — цвет, обозначающий границы области закраски	Закрашивает произвольную область по заданным шаблону и цвету вокруг заданной точки до границ, обозначенных определенным цветом

Интерфейс	Назначение
`Procedure` **`GetFillPattern`**`(var FillPattern: FillPatternType);` `FillPattern` — массив типа `FillPatternType`, содержащий информацию о шаблоне	Определяет установленный тип шаблона
`Procedure` **`GetFillSettings`**`(var FillInfo: FillSettingsType);` `FillInfo` — запись типа `FillSettingsType`, содержащая информацию о шаблоне	Определяет установленный тип шаблона
`Procedure` **`GetLineSettings`**`(var LineInfo: LineSettingsType);` `LineInfo` — запись типа `LineSettingsType`, содержащая информацию о типе линии	Определяет установленный тип линии
`Procedure` **`GetGraphBufSize`**`(Size: word);` `Size` — размер буфера	Изменяет размер буфера для функций заполнения
`Procedure` **`SetFillPattern`**`(Pattern: FillPatternType; Color: word);` `Pattern` — массив типа `FillPatternType`, содержащий информацию о шаблоне; `Color` — цвет раскраски	Устанавливает шаблон и цвет раскраски
`Procedure` **`SetFillStyle`**`(Pattern, Color: word);` `Pattern` — код стандартного шаблона; `Color` — цвет раскраски	Устанавливает один из стандартных шаблонов и цвет раскраски
`Procedure` **`SetLineStyle`**`(LineStyle, Pattern, Thickness: word);` `LineStyle` — код стиля линии, `Pattern` — собственный шаблон линии, `Thickness` — толщина линии	Устанавливает стиль линии, как один из стандартных или по собственному шаблону
Изображение геометрических фигур	
`Procedure` **`Arc`**`(X,Y: integer; StartAngle, EndAngle, R: word);` `(X,Y)` — координаты центра окружности; `StartAngle, EndAngle` — начальный и конечный углы дуги окружности в градусах (0,359); `R` — радиус окружности в пикселях в направлении `X`	Изображает дугу окружности. Процедура учитывает неодинаковость масштаба по осям

Интерфейс	Назначение
`Procedure` **`Bar`**`(x1,y1,x2,y2: integer);` `(x1,y1)` и `(x2,y2)` — координаты левого верхнего и правого нижнего углов прямоугольника	Изображает окрашенный прямоугольник без контура
`Procedure` **`Bar3D`**`(x1,y1,x2,y2: integer; Depth: word; Top: boolean);` `(x1,y1)` и `(x2,y2)` — координаты левого верхнего и правого нижнего углов параллелепипеда; `Depth` — его глубина; `Top` — определяет, изображать ли верхнюю грань фигуры (`true` — изображать)	Изображает трехмерный прямоугольный параллелепипед с заполнением передней грани. Верхняя грань может либо изображаться либо нет

`Procedure` **`Circle`**`(X,Y: integer; R: word);`
`(X,Y)` — координаты центра окружности;
`R` — радиус окружности в пикселях в направлении `X`
	Изображает окружность. Процедура учитывает неодинаковость масштаба по осям

`Procedure` **`DrawPoly`**`(NumPoints: word; var PolyPoints);` `NumPoints` — число точек, задающих ломаную линию; `PolyPoints` — массив точек (элементов типа `Point`), задающих ломаную линию	Ломаная линия, проходящая через данный массив точек

`Procedure` **`Ellipse`**`(X,Y: integer; StartAngle, EndAngle, Rx, Ry: word);`
`(X,Y)` — координаты центра эллипса;
`StartAngle`,
`EndAngle` — начальный и конечный углы дуги эллипса в градусах (0,359);
`Rx`, `Ry` — полуоси эллипса в пикселях в направлениях `X` и `Y`
	Изображает дугу эллипса. Полуоси эллипса в направлении *X* и *Y* заданы в пикселях

`Procedure` **`GetArcCoords`**`(var ArcCoords: ArcCoordsType);` `ArcCoordsType` — запись, содержащая координаты дуги	Возвращает координаты дуги, изображенной процедурами Arc и Ellips

`Procedure` **`FillEllipse`**`(X,Y: integer; Rx,Ry: word);`
`(X,Y)` — координаты центра эллипса;
`Rx`, `Ry` — полуоси эллипса в пикселях в направлениях `X` и `Y`
	Чертит закрашенный эллипс, полуоси эллипса в направлениях *X*, *Y* заданы в пикселях

Интерфейс	Назначение
`Procedure` **`FillPoly`**`(NumPoints: word; var PolyPoints);` `NumPoints` — число точек, задающих ломаную линию; `PolyPoints`— массив точек (элементов типа `Point`), задающих ломаную линию	Изображает закрашенный многоугольник, заданный массивом вершин
`Function` **`GetPixel`**`(X,Y: integer): word;` `(X,Y)` — координаты точки, результат — цвет точки	Возвращает цвет заданной точки
`Procedure` **`Line`**`(x1,y1,x2,y2: integer);` `(x1,y1), (x2,y2)` — координаты конечных точек отрезка прямой	Чертит отрезок прямой по двум заданным конечным точкам
`Procedure` **`LineRel`**`(dx,dy: integer);` `(dx,dy)` — смещение конечной точки относительно начального положения указателя	Чертит отрезок прямой от положения указателя до точки, смещенной относительно указателя на заданную величину
`Procedure` **`LineTo`**`(X,Y: integer);` `(X,Y)` — координаты точки, в которую проводится отрезок	Чертит отрезок прямой от точки положения указателя до заданной точки
`Procedure` **`Rectangle`**`(x1,y1,x2,y2: integer);` `(x1,y1), (x2,y2)` — координаты противоположных углов прямоугольника	Чертит контур прямоугольника

`Procedure` **`Sector`**`(X,Y: integer; StartAngle,EndAngle,Rx,Ry: word);`
`(X,Y)` — координаты центра эллипса;
`StartAngle, EndAngle` — начальный и конечный углы дуги эллипса в градусах (0,359);
`Rx, Ry` — полуоси эллипса в пикселях в направлениях `X` и `Y`
	Изображает закрашенный сектор эллипса, ограниченный углами от StartAngle до EndAngle. Заполняется сектор от минимального до максимального значения углов, независимо от их следования

`Procedure` **`PieSlice`**`(X,Y: integer; StartAngle, EndAngle, Rx: word);`
`(X,Y)` — координаты центра эллипса;
`StartAngle, EndAngle` — начальный и конечный углы дуги эллипса в градусах (0,359);
`Rx` — радиус окружности в пикселях в направлении `X`
	Изображает закрашенный сектор круга, ограниченный углами от StartAngle до EndAngle, радиус круга задан в пикселях в направлении *X* и учитывает масштаб изображения по осям. Заполняется сектор от минимального до максимального значения углов, независимо от их следования

Интерфейс	Назначение
`Procedure` **`PutPixel`**`(X,Y: integer; Color: word);` `(X,Y)` — координаты точки; `Color` — цвет точки	Окрашивает точку экрана в заданный цвет
Вывод текстов	
`Procedure` **`GetTextSettings`**`(var TextInfo: TextSettingsType);` `TextInfo` — запись, содержащая действующие установки вывода текста	Определяет установки вывода текста
`Procedure` **`OutText`**`(Text: String);` `Text` — выводимая строка текста	Выводит текст на графический экран, начиная с позиции графического указателя. Позиция графического указателя смещается на ширину выводимого текста
`Procedure` **`OutTextXY`**`(X,Y: Integer; Text: String);` `(X,Y)` — координаты точки, к которой привязывается выводимый текст; `Text` — выводимая строка текста	Выводит текст на графический экран относительно заданной точки, с учетом используемого типа юстировки
`Function` **`InstallUserFont`**`(FontFileName: string): integer;` `FontFileName` — имя файла, содержащего шрифт; результат — номер (код) установленного шрифта	Инсталлирует новый шрифт
`Procedure` **`RegisterBGIFont`**`(Font: Pointer);` `Font` — указатель на шрифт	Регистрирует шрифты для графической системы
`Procedure` **`SetTextJustify`**`(Horiz, Vert: word);` `Horiz, Vert` — коды привязки текста по горизонтали и вертикали	Устанавливает тип привязки текста к точке вывода по горизонтали и вертикали
`Procedure` **`SetTextStyle`**`(Font, Direction, CharSize: word);` `Font` — тип используемого шрифта, `Direction` — направление вывода надписи, `CharSize` — размер символов	Устанавливает стиль выводимого текста
`Procedure` **`SetUserCharSize`**`(MultX, DivX, MultY, DivY: word);` `MultX, DivX, MultY, DivY` — коэффициенты умножения (`Mult`) и деления (`Div`) по осям `X` и `Y` соответственно	Выполняет масштабирование шрифтов с произвольным дробным масштабом

Интерфейс	Назначение
`Function` **`TextHeight`**`(Text: string): integer;` `Text` — выводимая строка текста; результат — высота текста в пикселях	Определяет высоту текстовой строки при заданных установках стиля
`Function` **`TextWidth`**`(Text: string): integer;` `Text` — выводимая строка текста; результат — длина текста в пикселях	Определяет длину текстовой строки при заданных установках стиля
Копирование части экрана	
`Procedure` **`GetImage`**`(x1,y1,x2,y2: integer; var BitMap: Pointer);` `(x1,y1)` и `(x2,y2)` — координаты противоположных углов прямоугольника, ограничивающего копируемую область экрана; `BitMap` — указатель на область памяти, отведенную для хранения данного изображения	Копирует в ОЗУ прямоугольную область экрана
`Function` **`ImageSize`**`(x1,y1,x2,y2: integer): word;` `(x1,y1)` и `(x2,y2)` — координаты противоположных углов прямоугольника ограничивающего область экрана; результат — объем информации в байтах	Определяет размер в байтах памяти, необходимой для хранения прямоугольной области экрана
`Procedure` **`PutImage`**`(X,Y: integer; var BitMap: Pointer; BitBlt: word);` `(X,Y)` — координаты левого верхнего угла для выводимого изображения; `BitMap` — указатель на область памяти; `BitBlt` — код логической операции	Выводит изображение из области памяти ЭВМ в указанную область экрана с заданной логической операцией наложения нового изображения на старое

Приложение 3

Си++. Константы предельных значений

Таблица П3.1. Предельные значения для целостных типов — файл limits.h

Имя константы	Стандартные значения	Смысл
CHAR_BIT	8	Число битов в байте
SCHAR_MIN	−127	Минимальное значение для singed char
SCHAR_MAX	+127	Максимальное значение singed char
UCHAR_MAX	225	Максимальное значение unsigned char
CHAT_MIN	0	Минимальное значение для char
CHAT_MAX	SCHAR_MIN SCHAR_MAX UCHAR_MAX	Максимальное значение для char
MB_LEN_MAX	1	Минимальное число байтов в многобайтовом символе
SHRT_MIN	−32767	Минимальное значение для short
SHRT_MAX	+32767	Максимальное значение для short
USHRT_MAX	65535	Максимальное значение для unsigned short
INT_MIN	−32767	Минимальное значение для int
INT_MAX	+32767	Максимальное значение для int
UINT_MAX	65535	Максимальное значение unsigned int
LONG_MIN	−2147483647	Минимальное значение для long
LONG_MAX	+2147483647	Максимальное значение для long
ULONG_MAX	4294967295	Максимальное значение unsigned long

Таблица П3.2. Константы для вещественных типов — файл float.h

Имя константы	Стандартные значения	Смысл
FLT_RADIX	2	Основание экспоненциального представления, например: 2, 16
FLT_DIG	6	Количество верных десятичных цифр
FLT_EPSILON	1.19209290E-07F	Минимальное x, такое, что $1{,}0 + x \neq 1{,}0$
FLT_MANT_DIG	24	Количество цифр по основанию FLT_RADIX в мантиссе

Имя константы	Стандартные значения	Смысл
FLT_MAX	3.40282347E+38F	Максимальное число с плавающей точкой
FLT_MAX_EXP	+128	Максимальное n, такое, что $FLT\_RADIX^n - 1$ представимо в виде числа типа float
FLT_MAX_10_EXP	+38	Максимальное целое n, такое, что 10^n представимо как float
FLT_MIN	1.17549435E-38F	Минимальное нормализованное число с плавающей точкой типа float
FLT_MIN_EXP	-125	Минимальное n, такое, что 10^n представимо в виде нормализованного числа
FLT_MIN_10_EXP	-37	Минимальное отрицательное целое n, такое, что 10^n — в области определения чисел типа float
DBL_DIG	15	Количество верных десятичных цифр для типа double
DBL_EPSILON	2.2204460492503131E-16	Минимальное x, такое, что $1{,}0 + x \neq 1{,}0$, где x принадлежит типу double
DBL_MANT_DIG	53	Количество цифр по основанию FLT_RADIX в мантиссе для чисел типа double
DBL_MAX	1.7976931348623158E+308	Максимальное число с плавающей точкой типа double
DBL_MAX_EXP	+1024	Максимальное n, такое, что $FLT\_RADIX^n - 1$ представимо в виде числа типа double
DBL_MAX_10_EXP	+308	Максимальное целое n, такое, что 10^n представимо как double
DBL_MIN	2.2250738585072014E-308	Минимальное нормализованное число с плавающей точкой типа double
DBL_MIN_EXP	-1021	Минимальное n, такое, что 10^n представимо в виде нормализованного числа типа double
DBL_MIN_10_EXP	-307	Минимальное отрицательное целое n, такое, что, 10^n — в области определения чисел типа double

Приложение 4

Библиотека функций языка Си/Си++

Таблица П4.1. Математические функции — файл math.h

Функция	Прототип и краткое описание действий
abs	**int abs(int i);** Возвращает абсолютное значение целого аргумента *i*
acos	**double acos(double x);** Функция арккосинуса. Значение аргумента должно находиться в диапазоне от −1 до +1
asin	**double asin(double x);** Функция арксинуса. Значение аргумента должно находится в диапазоне от −1 до +1
atan	**double atan(double x);** Функция арктангенса
atan2	**double atan2(double y, double x);** Функция арктангенса от значения *y/x*
cabs	**double cabs(struct complex znum);** Вычисляет абсолютное значение комплексного числа znum. Определение структуры (типа) complex — в файле math.h
cos	**double cos(double x);** Функция косинуса. Угол (аргумент) задается в радианах
cosh	**double cosh(double x);** Возвращает значение гиперболического косинуса *x*
exp	**double exp(double x);** Вычисляет значение e^x (экспоненциальная функция)
fabs	**double fabs(double x);** Возвращает абсолютное значение вещественного аргумента *x* двойной точности
floor	**double floor(double x);** Находит наибольшее целое, не превышающее значения *x*. Возвращает его в форме double
fmod	**double fmod(double x, double y);** Возвращает остаток от деления нецелого *x* на *y*
hypot	**double hyport(double x, double y);** Вычисляет гипотенузу *z* прямоугольного треугольника по значениям катетов *x*, *y* ($z^2 = x^2 + y^2$)
labs	**double labs(long x);** Возвращает абсолютное значение целого аргумента long*x*
ldexp	**double ldexp(double v, int e);** Возвращает значение выражения $v*2^e$
log	**double log(double x);** Возвращает значение натурального логарифма (ln*x*)

Функция	Прототип и краткое описание действий
`log10`	**`double log10(double x);`** Возвращает значение десятичного логарифма ($\log_{10}x$)
`poly`	**`double poly(double x, int n, double c[]);`** Вычисляет значение полинома: $C[n]x^n + C[n-1]x^{n-1} + \ldots + C[1]x + C[0]$
`pow`	**`double pow(double x, double y);`** Возвращает значение x^y, т.е. x в степени y
`pow10`	**`double pow10(int p);`** Возвращает значение 10^p
`sin`	**`double sin(double x);`** Функция синуса. Угол (аргумент) задается в радианах
`sinh`	**`double sinh(double x);`** Возвращает значение гиперболического синуса x
`sqrt`	**`double sqrt(double x);`** Возвращает положительное значение квадратного корня $\sqrt{x}$
`tan`	**`double tan(double x);`** Функция тангенса. Угол (аргумент) задается в радианах
`tanh`	**`double tanh(double x);`** Возвращает значение гиперболического тангенса для x

Таблица П4.2. Функция проверки и преобразования символов — файл ctype.h

Функция	Прототип и краткое описание действий
`isalnum`	**`int isalnum(int c);`** Дает значение не нуль, если c — код буквы или цифры (A...Z, a...z, 0...9), и нуль — в противном случае
`isalal-pha`	**`int isalalpha(int c);`** Дает значение не нуль, если c — код буквы (A...Z, a...z), и нуль — в противном случае
`isascii`	**`int isascii(int c);`** Дает значение не нуль, если c — код ASCII, т.е.принимает значение от 0 до 127, в противном случае — нуль
`iscntrl`	**`int iscntrl(int c);`** Дает значение не нуль, если c — управляющий символ с кодами 0x00...0x01F или 0x7F, и нуль — в противном случае
`isdigit`	**`int isdigit(int c);`** Дает значение не нуль, если c — цифра (0...9) в коде ASCII, и нуль — в противном случае
`isgraph`	**`int isgraph(int c);`** Дает значение не нуль, если c — видимый (изображаемый) символ с кодом (021...0x7E), и нуль — в противном случае

Функция	Прототип и краткое описание действий
`islower`	`int islower(int c);` Дает значение не нуль, если `c` — код символа на нижнем регистре (a ... z), и нуль — в противном случае
`isprint`	`int isprint(int c);` Дает значение не нуль, если `c` — печатный символ с кодом (0x20 ... 0x7E), и нуль — в противном случае
`ispunct`	`int ispunct(int c);` Дает значение не нуль, если `c` — символ-разделитель (соответствует `iscntrl` или `isspace`), и нуль — в противном случае
`isspace`	`int isspace(int c);` Дает значение не нуль, если `c` — обобщенный пробел: пробел, символ табуляции, символ новой строки или новой страницы, символ возврата каретки (0x09 ... 0x0D, 0x20), и нуль — в противном случае
`isupper`	`int isupper(int c);` Дает значение не нуль, если `c` — код символа на верхнем регистре (A ... Z), и нуль — в противном случае
`isxdig-it`	`int isxdigit(int c);` Дает значение не нуль, если `c` — код шестнадцатеричной цифры (0 ... 9, A ... F, a ... f), и нуль — в противном случае
`toascii`	`int toascii(int c);` Преобразует целое число `c` в символ кода ASCII, обнуляя все биты, кроме младших семи. Результат от 0 до 127
`tolower`	`int tolower(int c);` Преобразует код буквы `c` к нижнему регистру, остальные коды не изменяются
`toupper`	`int toupper(int c);` Преобразует код буквы `c` к верхнему регистру, остальные коды не изменяются

Таблица П4.3. Функции ввода-вывода для стандартных файлов — файл stdio.h

Функция	Прототип и краткое описание действий
`getch`	`int getch(void);` Считывает один символ с клавиатуры без отображения на экране
`getchar`	`int getchar(void);` Считывает очередной символ из стандартного вводного файла (stdin)
`gets`	`char*gets(char*s);` Считывает строку `s` из стандартного вводного файла (stdin)
`printf`	`int printf(char*format [,argument,...]);` Функция форматированного ввода в файл stdout

Функция	Прототип и краткое описание действий
putchar	**int putchar(int c);** Записывает символ c в стандартный файл вывода (stdout)
puts	**int puts(const char*s);** Записывает строку s в стандартный файл вывода (stdout)
scanf	**int scanf(char*format[,argument,...]);** Функция форматированного ввода из файла stdin
sprintf	**int sprintf(char*s, char*format [,argument,...]);** Функция форматированной записи в строку s
sscanf	**int sscanf(char*s, char*format [,address,...]);** Функция форматированного чтения из строки s
ungetch	**int ungetch(int c);** Возвращает символ c в стандартный файл ввода stdin, заставляя его быть следующим считываемым символом

Таблица П4.4. Функции для работы со строками — файлы string.h, stdlid.h

Функция	Прототип и краткое описание действий
atof	**double atof(char *str);** Преобразует строку str в вещественное число типа double
atoi	**int atoi(char *str);** Преобразует строку str в десятичное целое число
atol	**long atol(char *str);** Преобразует строку str в длинное десятичное целое число
itoa	**char *itoa(int v, char *str, int baz);** Преобразует целое v в строку str. При изображении числа используется основание baz ($2 \le baz \le 36$). Для отрицательного числа и baz = 10 первый символ — минус(−)
ltoa	**char*ltoa(long v, char*str, int baz);** Преобразует длинное целое v в строку str. При изображении числа используется основание baz ($2 \le baz \le 36$)
strcat	**char *strcat(char *sp, char*si);** Приписывает строку si к строке sp (конкатенация строк)
strchr	**char *strchr(char *str, int c);** Ищет в строке str первое вхождение символа c
strcmp	**int strcmp(char *str1, char *str2);** Сравнивает строки str1 и str2. Результат отрицателен, если str1<str2; равен нулю, если str1 = str2 и положителен, если str1>str2 (сравнение беззнаковое)
strcpy	**char *strcpy(char *sp, char *si);** Копирует байты строки si в строку sp

Функция	Прототип и краткое описание действий
strcspn	**int strcspn(char *str1, char *str2);** Определяет длину первого сегмента строки str1, содержащего символы, не входящие во множество символов строки str2
strdup	**char *strdup(const char *str);** Выделяет память и переносит в нее копию строки str
strlen	**unsigned strlen(char *str);** Вычисляет длину строки str
strlwr	**char *strlwr(char * str);** Преобразует буквы верхнего регистра в строке в соответствующие буквы нижнего регистра
strncat	**char *strncat(char *sp, char *si, int kol);** Приписывает kol символов строки si к строке sp (конкатенация)
strncmp	**int strncmp(char *str1, char *str2, int kol);** Сравнивает части строк str1 и str2, причем рассматриваются первые kol символов. Результат отрицателен, если str1<str2; равен нулю, если str1 = str2, и положителен, если str1>str2
strncpy	**char *strncpy(char *sp, char *si, int kol);** Копирует kol символов строки si в строку sp («хвост» отбрасывается или дополняется пробелами)
strnicmp	**char*strnicmp(char *str1, char *str2, int kol);** Сравнивает не более kol символов строки str1 и строки str2, не делая различия регистров
strnset	**char *strnset(char *str,int c,int kol);** Заменяет первые kol символов строки str символом c
strpbrk	**char *strpbrk(char *str1,char *str2);** Ищет в строке str1 первое появление любого из множества символов, входящих в строку str2
strrchr	**char *strrchr(char *str,int c);** Ищет в строке str последнее вхождение символа c
strset	**int strset(char *str,int c);** Заполняет строку str символом c
strspn	**int strspn(char *str1,char *str2);** Определяет длину первого сегмента строки str1, содержащего только символы, из множества символов строки str2
strstr	**char *strstr(const char *str1,const char *str2);** Ищет в строке str1 подстроки str2. Возвращает указатель на тот элемент в строке str1, с которого начинается подстрока str2
strtod	**double *strtod(const char *str,char **endptr);** Преобразует символьную строку str в число двойной точности. Если endptr не равен null, то *endptr возвращает как указатель на символ, при достижении которого прекращено чтение строки str

Функция	Прототип и краткое описание действий
strtok	**char *strtok(char *str1,const char *str2);** Ищет в строке str1 лексемы, выделенные символами из второй строки
strtol	**long *strtol(const char *str,char **endptr,int baz);** Преобразует символьную строку str к значению «длинное число» с основанием baz ($2 \le baz \le 36$). Если endptr не равен null, то *endptr возвращается как указатель на символ, при достижении которого прекращено чтение строки str
strupr	**char *strupr(char *str);** Преобразует буквы нижнего регистра в строке в соответствующие буквы верхнего регистра
ultoa	**char *ultoa(unsigned long v,char *str,int baz);** Преобразует беззнаковое длинное целое v в строку str
calloc	**void *calloc(unsiped n,unsiped m);** Возвращает указатель на начало области динамически распределенной памяти для размещения n элементов по m байт каждый. При неудачном завершении возвращает значение null
coreleft	**unsigned coreleft(void);** — для моделей памяти tiny, small, medium. **unsigned long coreleft(void);** — для других моделей памяти. Возвращает значение объема неиспользованной памяти, функция, уникальная для DOS, где приняты упомянутые модели памяти
free	**void free(void *bl);** Освобождает ранее выделенный блок динамически распределенной памяти с адресом первого байта bl
malloc	**void *malloc(unsigned s);** Возвращает указатель на блок 0 динамически распределенной памяти длиной *s* байт. При неудачном завершении возвращает значение null
realloc	**void *realloc(void *bl,unsigned ns);** Изменяет размер ранее выделенной динамической памяти с адресом bl до размера ns байт. Если bl равен null, то функция выполняется как malloc()

Таблица П4.5. Функции для работы с терминалом в текстовом режиме — файл conio.h

Функция	Прототип и краткое описание действий
clreol	**void clreol(void);** Стирает символы от позиции курсора до строки в текстовом окне
clrscr	**void clrscr(void);** Очищает экран

Функция	Прототип и краткое описание действий
`cgets`	**`char *cgets(char *str);`** Выводит на экран строку `str`
`cprintf`	**`int cprintf(char *format [, argument, ...]);`** Выводит форматированную строку в текстовое окно, созданное функцией `window()`
`cputs`	**`int cputs(char *str);`** Считывает в символьный массив `str` строку с клавиатуры (консоли)
`cscanf`	**`int cscanf(char *format [, address, ...]);`** Функции ввода-вывода информации, которые используются при работе с терминалом в текстовом режиме
`delline`	**`void delline(void);`** Удаляет строку в текстовом окне (где находится курсор)
`gotoxy`	**`void gotoxy(int x, int y);`** Перемещает курсор в позицию текстового окна с координатами `(x,y)`
`highvideo`	**`void highvideo(void);`** Повышает яркость символов, выводимых после нее на экран
`movetext`	**`int movetext(int x0, int y0, int x1, int y1, int x, int y);`** Переносит текстовое окно в область экрана, правый верхний угол которой имеет координаты `(x,y)`. Координаты угловых точек окна — `(x0,y0)`, `(x1,y1)`
`normvideo`	**`void normvideo(void);`** Устанавливает нормальную яркость выводимых на экран символов
`textattr`	**`void textattr(int newattr);`** Устанавливает атрибуты (фон, цвет) символов, выводимых на экран
`textbackground`	**`void textbackground(int c);`** Устанавливает цвет фона по значению параметра `c`
`textcolor`	**`void textcolor(int c);`** Устанавливает цвет символов по значению параметра `c`
`textmode`	**`void textmode(int m);`** Переводит экран в текстовый режим по значению параметра `m`
`wherex`	**`int wherex(void);`** Возвращает значение горизонтальной координаты курсора
`wherey`	**`int wherey(void);`** Возвращает значение вертикальной координаты курсора
`window`	**`void window(int x0, int y0, int x1, int y1);`** Создает текстовое окно по координатам угловых точек `(x0, y0)`, `(x1, y1)`

Таблица П4.6. Специальные функции

Функция	Прототип и краткое описание действий	Место-нахождение прототипа
`delay`	**`void delay(unsigned x);`** Приостанавливает выполнение программы на *x* мс	`dos.h`
`kbhit`	**`int kbhit(void);`** Возвращает ненулевое целое, если в буфере клавиатуры присутствуют коды нажатия клавиш, в противном случае — нулевое значение	`conio.h`
`memcmp`	**`int memcmp(void s1, void s2, unsigned n);`** Сравнивает посимвольно две области памяти `s1` и `s2` длиной `n` байт. Возвращает значение меньше нуля, если `s1<s2`, нуль, если `s1=s2`, и больше нуля, если `s1>s2`	`mem.h`
`memcpy`	**`void *memcpy(const void *p, const void *i, unsigned n);`** Копирует блок длиной `n` байт из области памяти `i` в область памяти `p`	`mem.h`
`memicmp`	**`int memicmp( oid *s1, void *s2, unsigned n);`** Подобна `memcpy`, за тем исключением, что игнорируются различия между буквами верхнего и нижнего регистров	`mem.h`
`memmove`	**`void *memmove(void *dest, const void *src, int n);`** Копирует блок длиной `n` байт из `scr` в `dest`. Возвращает указатель `dest`	`mem.h`
`memset`	**`void *memset(void *s, int c, unsigned n);`** Записывает во все байты области `s` значение `c`. Длина области `s` равна `n` байт	`mem.h`
`nosound`	**`void nosound(void);`** Прекращает подачу звукового сигнала, начатого функцией `sound()`	`dos.h`
`peek`	**`int peek(unsigned s, unsigned c);`** Возвращает целое значение (слово), записанное в сегменте `s` со смещением `c`	`dos.h`
`peekb`	**`char peekb(unsigned s, unsigned c);`** Возвращает один байт, записанный в сегменте `s` со смещением `c`, т.е. по адресу `s:c`	`dos.h`
`poke`	**`void poke( unsigned s, unsigned c, int v);`** Перемещает значение `v` в слово сегмента `s` со смещением `c`, т.е. по адресу `s:c`	`dos.h`

Функция	Прототип и краткое описание действий	Место-нахождение прототипа
`pokeb`	**`void pokeb(unsigned s, unsigned c, char v);`** То же, что и `poke`, но помещает один байт `v` по адресу `s:c`	`dos.h`
`rand`	**`int rand(void);`** Возвращает псевдослучайное целое число из диапазона $0...2^{15}-1$, может использовать `srand()`	`stdlib.h`
`signal`	**`int signal(int sig);`** Вызывает програмный сигнал с номером `sig`. Используется для обработки исключительных ситуаций в языке Си	`signal.h`
`sound`	**`void sound(unsigned f);`** Вызывает звуковой сигнал с частотой `f` Гц	`dos.h`
`srang`	**`void srang(unsigned seed);`** Функция инициализации генератора случайных чисел `(rand)`; `seed` — любое беззнаковое целое число	`stdlib.h`

Графическая библиотека в C++

Таблица П4.7. Функции для управления графической системой

Функция	Прототип и краткое описание действий
`closegraph`	**`void far closegraph(void);`** Перевод системы в текстовый режим (из графического)
`graphdefaults`	**`void far graphdefaults(void);`** Устанавливает по умолчанию все параметры графической системы (параметры заполнения, палитру, правила выравнивания текста и т.д.)
`_graphfreemem`	**`void far _graphfreemem(void far *ptr, unsigned size);`** Введение этой функции в программу позволяет программисту отслеживать запросы на освобождение `size` байт памяти функциями графической библиотеки
`_graphgetmem`	**`void far *far _graphgetmem(unsigned size);`** Введение этой функции в программу позволяет программисту отслеживать запросы на выделение `size` байт памяти функциями графической библиотеки

Функция	Прототип и краткое описание действий
`initgraph`	**`void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);`** Обеспечивает перевод системы в графический режим (из текстового), инициализация графики
`installuserdriver`	**`int far installuserdriver(char far *name, int huge *detect)(void));`** Добавляет новый графический драйвер `name(.BGI)` в таблицу драйверов BGI (Borland Graphics Interface)
`installuserfont`	**`int far installuserfont(char far *name);`** Устанавливает шрифты, содержащиеся в файле `name(.CHR)`
`registerbgidriver`	**`int far registerbgidriver(void (*driver)(void));`** Регистрирует драйвер `driver`, встроенный в текущую выполняемую программу
`registerbgifont`	**`int far registerbgifont(void (*font)(void));`** Регистрирует шрифт `font`, встроенный в текущую выполняемую программу
`restorectrmode`	**`void far restorectrmode(void);`** Осуществляет временный переход в текстовый режим, из которого была вызвана функция `initgraph()`
`setgraphbufsize`	**`unsigned far setgraphbufsize(unsigned bufsize);`** Устанавливает размер внутреннего буфера для графических функций
`setactivepage`	**`void far setactivepage(int page);`** Устанавливает активной для вывода графики страницу `page`
`setallpallete`	**`void far setallpallete(struct palettetype far *palette);`** Устанавливает все цвета палитры
`setaspectrratio`	**`void far setaspectrratio(int xasp, int yasp);`** Устанавливает коэффициент сжатия по координатам `(x, y)`
`setbkcolor`	**`void far setbkcolor(int color);`** Устанавливает цвет фона
`setcolor`	**`void far setcolor(int color);`** Устанавливает цвет точки (изображения)
`_setcursortype` (прототип содержится в файле `cotio.h`)	**`void far setcursortype(int cur_t);`** Устанавливает тип отображения курсора для текстовых режимов

Функция	Прототип и краткое описание действий
`setfillpattern`	**`void far setfillpattern(char far *upattern, int color);`** Устанавливает заданный пользователем шаблон закраски экрана или области экрана
`setgraphmode`	**`void far setgraphmode(int mode);`** Выполняет переход к графическому режиму, отличному от установленного функцией `initgraph()`
`setfillstyle`	**`void far setfillstyle(int pattern, int color);`** Устанавливает один из стандартных шаблонов заполнения экрана или области экрана
`setlinestyle`	**`void far setlinestyle(int linestyle, unsigned upattern, int thickness);`** Устанавливает толщину и тип изображаемой линии
`setpalette`	**`void far setpalette(int colornum, int color);`** Устанавливает один из цветов палитры
`setrgbpallete`	**`void far setrgbpallete(int colornum, int red, int green, int blue);`** Устанавливает цвета для графического адаптера IBM 8514
`settextjustify`	**`void far settextjustify(int horiz, int vert);`** Устанавливает правила выравнивания текста при горизонтальном и вертикальном выводе функцией `outtext()`
`settextstyle`	**`void far settextstyle(int font, int direction, int charsize);`** Устанавливает стиль (шрифт, размеры символов) текста, выводимого функцией `outtext()`
`setusercharsize`	**`void far setusercharsize(int multx, int divx, int multy, int divy);`** Устанавливает размеры текущего окна экрана для вывода изображений или текста
`setviewport`	**`void far setviewport(int left, int top, int righ, int bottom, int clip);`** Устанавливает размеры текущего окна экрана для вывода изображений или текста
`setvisualpage`	**`void far setvisualpage(int page);`** Делает видимой графическую страницу `page`
`setwritemode`	**`void far setwritemode(int mode);`** Устанавливает режим вывода линий в графическом режиме

Таблица П4.8. Функции для получения изображения на экране

Функция	Прототип и краткое описание действий
arc	**void far arc(int x, int y, int stangle, int endangle, int radius);** Вычерчивает дугу окружности с центром (x, y)
bar	**void far bar(int left, int top, int right, int bottom);** Вычерчивает закрашенный прямоугольник
bar3	**void far bar3(int left, int top, int right, int bottom, int depth, int topflag);** Вычерчивает закрашенный параллелепипед
circle	**void far circle(int x, int y, inr radius);** Вычерчивает окружность с центром (x, y)
cleardivice	**void far cleardivice(void);** Очищает экран цветом фона
clearviewport	**void far clearviewport(void);** Очищает ранее установленное окно графического экрана
drawpoly	**void far drawpoly(int numpoints, int far *polypoints);** Вычерчивает контур многоугольника с numpoints вершинами
ellipse	**void far ellipse(int x, int y, int xradius, int yradius);** Вычерчивает дугу эллипса с центром (x, y)
fillellipse	**void far fillellipse(int x, int y, int xradius, int yradius);** Вычерчивает эллипс с центром в точке (x, y) и заполняет его установленным ранее шаблоном закраски
fillpoly	**void far fillpoly(int numpoints, int far *polypoints);** Вычерчивает закрашенный многоугольник с numpoints вершинами и заполняет его установленным ранее шаблоном закраски
floodfill	**void far floodfill(int x, int y, int border);** Заполняет установленным ранее шаблоном закраски ограниченную область экрана, в которую попадает точка с координатами (x, y)
line	**void far line(int x1, int y1, int x2, int y2);** Вычерчивает линию от (x1, y1) до (x2, y2)

Функция	Прототип и краткое описание действий
`linerel`	**`void far linerel(int dx, int dy);`** Вычерчивает линию из текущей точки в точку, отстоящую от нее на величину `(dx, dy)`
`lineto`	**`void far lineto(int x, int y);`** Проводит линию из текущей точки в точку с абсолютными координатами `(x, y)`
`moverel`	**`void far moverel(int dx, int dy);`** Перемещает указатель позиции из текущей точки в точку отстоящую от нее на величину `(dx, dy)`
`moveto`	**`void far moveto(int x, int y);`** Перемещает указатель позиции из текущей точки в точку, с абсолютными координатами `(x, y)`
`outtext`	**`void far outtext(char far *textstring);`** Выводит текстовую строку `textstring`, начиная с текущей позиции
`outtextxy`	**`void far outtextxy(int x, int y, char far *textstring);`** Выводит текстовую строку `textstring`, начиная с точки с координатами `(x, y)`
`pieslice`	**`void far pieslice(int x, int y, int stangle, int endangle, int radius);`** Вычерчивает закрашенный сектора круга с центром в точке `(x, y)`
`putimage`	**`void far putimage(int left, int top, void far *bitmap, int op);`** Выводит ранее сохраненное графическое изображение в окно экрана с левым верхним углом `(left, top)`
`putpixel`	**`void far putpixel(int x, int y, int color);`** Вычерчивает точки по координатам `(x, y)`
`rectangle`	**`void far rectangle(int left, int top, int right, int bottom);`** Вычерчивает прямоугольник с заданными вершинами
`sector`	**`void far sector(int x, int y, int stangle, int endangle, int xradius, int yradius);`** Вычерчивает сектор эллипса с центром в точке `(x,y)` и заполняет его установленным ранее шаблоном закраски

Таблица П4.9. Функции для получения параметров изображения

Функция	Прототип и краткое описание действий
`detectgraph`	**`void far detectgraph(int far *graphdriver, int far *graphmode);`** Функция возвращает тип вашего графического адаптера `graphdriver` и режим `graphmode` с разрешением, максимально возможным для этого адаптера
`getarccoords`	**`void far getarccoords(struct arccoordstype far *arcoords);`** Возвращает в структуре `arcoords` значения координат дуги, построенной при последнем обращении к `arc()`
`getaspectratio`	**`void far getaspectratio(int far * xasp, int far * yasp);`** Возвращает коэффициент сжатия `(yasp/xasp)` по координатам `x` и `y`
`getbkcolor`	**`int far getbkcolor(void);`** Возвращает номер текущего цвета фона
`getcolor`	**`int far getcolor(void);`** Возвращает номер текущего цвета изображения
`getdefaultpallete`	**`char *far getdefaultpallete(void);`** Возвращает указатель на структуру типа `palettype`, содержащую информацию о палитре (наборе цветов), устанавливаемой по умолчанию функцией `initgraph()`
`getddivername`	**`char *far getddivername(void);`** Возвращает указатель на строку, содержащую название текущего графического драйвера
`getfillpattern`	**`void far getfillpattern(char far *pattern);`** Получение кодов, применяемых пользователем для задания шаблона заполнения экрана или его области
`getfillsettings`	**`void far getfillsettings (struct fillsettingstype far *fillinfo);`** Возвращает в структуре `fillinfo` значения параметров заполнения и цвета экрана
`getgraphmode`	**`int far getgraphmode(void);`** Возвращает номер графического режима
`getimage`	**`void far getimage(int left, int top, int right, int bottom, void far *bitmap);`** Обеспечивает получение и сохранение в области памяти, на которую указывает `bitmap`, окна экрана, заданного координатами вершин

Функция	Прототип и краткое описание действий
`getlinesettings`	**`void far getlinesettings(struct linesettingstype far *lineinfo);`** Возвращает в структуре `lineinfo` значения параметров линии
`getmaxcolor`	**`int far getmaxcolor(void);`** Возвращает наибольший номер цвета, который можно установить в текущем режиме графического драйвера с помощью функции `setcolor()`
`getmaxmode`	**`int far getmaxmode(void);`** Возвращает наибольший номер режима, который возможно установить для текущего графического драйвера
`getmodename`	**`char * far getmodename(int mode_number);`** Возвращает указатель на строку с названием графического режима `mode_number`
`getmoderange`	**`void far getmoderange(int graphdriver, int far *lomode, int far *himode);`** Возвращает диапазон доступных графических режимов для графического драйвера `graphdriver`
`getmaxx`	**`int far getmaxx(void);`** Возвращает целое значение, равное размеру экрана по горизонтали (максимальное значение `x`)
`getmaxy`	**`int far getmaxy(void);`** Возвращает целое значение, равное размеру экрана по вертикали (максимальное значение `y`)
`getpalette`	**`void far getpalette(struct palettetype far *palette);`** Выдает указатель `palette` на структуру типа `palette`, содержащую информацию о текущей палитре (наборе цветов)
`getpalettesize`	**`int far getpalettesize(void);`** Возвращает количество цветов, доступных в текущем графическом режиме
`getpixel`	**`unsigned far getpixel(int x, int y);`** Возвращает цвет заданной точки `(x, y)`
`gettextsettings`	**`void far gettextsettings(struct textsettingstype far *texttypeinfo);`** Возвращает в структуре `texttypeinfo` значения параметров текста

Функция	Прототип и краткое описание действий
`getviewsettings`	**`void far getviewsettings(struct viewportstype far *viewport);`** Возвращает в структуре `viewport` значения параметров окна экрана.
`getx`	**`int far getx(void);`** Возвращает целое значение координаты `x` текущей позиции экрана
`gety`	**`int far gety(void);`** Возвращает целое значение координаты `y` текущей позиции экрана
`graphresult`	**`int far graphresult(void);`** Возвращает номер ошибки графической операции (целое число от —15 до —1); значение 0 говорит об отсутствии ошибок
`grapherrormsg`	**`char *far grapherrormsg(int errorcode);`** Возвращает указатель на строку, содержащую описание ошибки номера `errorcode`
`imagesize`	**`unsigned farimagesize(int left, int top, int right, int bottom);`** Возвращает объем буфера, нужного для сохранения графической информации в окне экрана с заданными вершинами
`textheight`	**`int far textheight(char far *textstring);`** Возвращает целое значение высоты в пикселях символов из строки `textstring`
`textwidth`	**`int far textwidth(char far *textstring);`** Возвращает в пикселях целое значение ширины строки символов `textstring`

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов В.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль. — М.: Наука, 1988.

2. Березин Б.И., Березин С.Б. Начальный курс С и С++. — М.: ДИАЛОГ-МИФИ, 1996.

3. Бондарев В.М., Рублинецкий В.И., Качко Е.Г. Основы программирования. — Харьков: Фолио, Ростов н/Д: Феникс, 1997.

4. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. — М.: Мир, 1981.

5. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989.

6. Гладков В.П. Задачи по информатике на вступительном экзамене в вуз и их решения: Учебное пособие. — Пермь: Перм. техн. ун-т, 1994.

7. Гладков В.П. Курс лабораторных работ по программированию: Учебное пособие для специальностей электротехнического факультета ПГТУ. Пермь: Перм. техн. ун-т, 1998.

8. Грогоно П. Программирование на языке Паскаль. — М.: Мир, 1982.

9. Дагене В.А., Григас Г.К., Аугутис К.Ф. 100 задач по программированию. — М.: Просвещение, 1993.

10. Епашников А.М., Епашников В.А. Программирование в среде Турбо Паскаль 7.0. — М.: МИФИ, 1994.

11. Заварыкин В.М., Житомирский В.Г., Лапчик М.П. Основы информатики и вычислительной техники. — М.: Просвещение, 1989.

12. Задачи по программированию / С.А.Абрамов, Г.Г.Гнездилова, Е.Н.Капустина, М.И.Селюн. — М.: Наука, 1988.

13. Зубов В.С. Программирование на языке Turbo Pascal (версии 6.0 и 7.0). — М.: Информационно-издательский дом «Филинъ», 1997.

14. Зуев Е.А. Практическое программирование на языке Turbo Pascal 6.0, 7.0. — М.: Радио и связь, 1994.

15. Информатика. Задачник-практикум: В 2 т. / Под ред. И.Г.Семакина, Е.К.Хеннера. — М.: Лаборатория Базовых Знаний, 1999.

16. Йенсен К., Вирт Н. Паскаль — руководство для пользователей и описание языка. — М.: Мир, 1982.

17. Касаткин В.Н. Информация. Алгоритмы. ЭВМ. — М.: Просвещение, 1991.

18. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. — М.: Финансы и статистика, 1992.

19. Культин Н.Б. Программирование в Turbo Pascal и Delphi. — СПб.: BHV — Санкт-Петербург, 1998.

20. Ляхович В.Ф. Руководство к решению задач по основам информатики и вычислительной техники. — М.: Высшая школа, 1994.

21. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0 / Под ред. В.П.Тарасенко. — Киев: ВЕК+; М.: Бином Универсал, 1998.

22. Миков А.И. Информатика. Введение в компьютерные науки. — Пермь: Изд-во ПГУ, 1998.

23. Могилев А.В., Пак Н.И., Хеннер Е.К. Информатика: Учеб. пособие для студ. пед. вузов / Под ред. Е.К.Хеннера. — М.: Изд. центр «Академия», 1999.

24. Олимпиады по информатике. Задачи и решения: методические рекомендации для учителей и учащихся школ. — Красноярск, 1991.

25. Основы информатики и вычислительной техники. Ч. 1, 2 / Под ред. А.П. Ершова и В.М. Монахова. — М.: Просвещение, 1988.

26. Основы информатики и вычислительной техники в базовой школе / Л.А. Залогова, С.В. Русаков , И.Г. Семакин и др. — Пермь, 1995.

27. Паппас К., Мюррей У. Программирование на С и С++. — Киев: «Ирина»; BHV, 2000.

28. Пильщиков В.Н. Сборник упражнений по языку Паскаль. — М.: Наука, 1989.

29. Подбельский В.В., Фомин С.С. Программирование на языке Си. — М.: Финансы и статистика, 1999.

30. Подбельский В.В. Язык Си++. — М.: Финансы и статистика, 1996.

31. Попов Б.В. TURBO PASCAL для школьников. Версия 7.0. — М.: Финансы и статистика, 1996.

32. Романов Е.Л. http://ermak.cs.nstu.ru/cprog — электронный учебник по дисциплине «Информатика».

33. Сборник задач по программированию / Авт.-сост. А.П.Шестаков. — Пермь: Перм. ун-т, 1999.

34. Сычев Н.А. Задания для вступительных экзаменов по информатике в НГУ // Информатика и образование. — 1995. — №2.

35. Хонсбергер Р. Математические изюминки. — М.: Наука, 1992.

36. Шень А. Программирование: теоремы и задачи. — М.: МЦНМО, 1995.

37. http://eldorado.mirea.ac.ru/zao5

ОГЛАВЛЕНИЕ

Учебное издание

Семакин Игорь Геннадьевич,
Шестаков Александр Петрович

Основы программирования

Учебник

Редактор *Е. Б. Стефанова*
Технический редактор *Е. Ф. Коржуева*
Компьютерная верстка: *Н. В. Соколова*
Корректоры *А. В. Черкасова, Л. В. Яковлева*

Диапозитивы предоставлены издательством.

Подписано в печать 20.11.2001. Формат 60×90/16. Гарнитура «Таймс». Бумага тип. № 2. Печать офсетная. Объем 27,0 усл. печ. л. Тираж 30 000 экз. (1-й завод 1 – 10 000 экз.). Заказ № 1165.

Лицензия ИД № 00520 от 03.12.1999. Издательство «Мастерство».
117342, г. Москва, ул. Бутлерова, 17-Б, к. 223. Тел./факс: (095) 330-1092, 334-8337.

Лицензия ИД № 02025 от 13.06.2000. Издательский центр «Академия».
117342, г. Москва, ул. Бутлерова, 17-Б, к. 223. Тел./факс: (095) 330-1092, 334-8337.

Отпечатано на Саратовском полиграфическом комбинате.
410004, г. Саратов, ул. Чернышевского, 59.