В.И. БУДИН
Е.А. КРАЙНОВА
С.Н. МАЙОРОВА
А.В. ТАРАКАНОВ

ПРОГРАММИРОВАНИЕ

Учебное пособие

Самара 2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

К а ф е д р а «Информатика и системы управления»

В.И. БУДИН, Е.А. КРАЙНОВА,
С.Н. МАЙОРОВА, А.В. ТАРАКАНОВ

ПРОГРАММИРОВАНИЕ

Учебное пособие

Самара
Самарский государственный технический университет
2014

Печатается по решению редакционно-издательского совета СамГТУ

УДК 004.43
ББК 32.973-018

Будин В. И.

Программирование: учеб. пособие / *В. И. Будин, Е. А. Крайнова, С. Н. Майорова, А. В. Тараканов.* – Самара: Самар. гос. техн. ун-т, 2014. – 199 с.: ил. 22, таб.: 22, Библиогр.: 11.

Излагаются вопросы программирования на языках высокого уровня Turbo Pascal и C++ с применением основных типов данных, начиная от простых и заканчивая сложными структурированными типами. Материал расположен в порядке усложнения и сопровождается значительным количеством примеров. Приводятся контрольные вопросы и упражнения для самостоятельного решения.

Для бакалавров, обучающихся по направлениям 230100 и 051000, изучающих основы алгоритмизации и программирования в рамках дисциплин "Программирование" и "Языки и системы прогаммирования". Может быть также полезно преподавателям, инженерам и научным работникам.

УДК 004.43
ББК 32.973-018

Р е ц е н з е н т ы : доцент кафедры «Авиционное радоэлектронное оборудованиие» ВУНЦ ВВС «ВВА» к.т.н., Ф. В. Дремов;

доцент каф ЭИКТ филиала ФГБОУ ВПО СамГТУ в г. Сызрани к.п.н. Раецкая О. В.

© В. И. Будин, Е. А. Крайнова
С. Н. Майорова, А. В. Тараканов 2014
© Самарский государственный
технический университет, 2014

ПРЕДИСЛОВИЕ

В настоящем учебном пособии рассматриваются основные подходы к созданию структурных программ, что является в свою очередь, базовым при изучении множества методов и технологий программирования. Основное внимание уделено базовым приемам и методам составления программ на алгоритмических языках программирования Pascal и C++.

Материал учебного пособия соответствует содержанию дисциплины «Программирование» направления 230100 «Информатика и вычислительная техника». Данное учебное пособие дополняет существующие учебники и учебные пособия по дисциплине и отличается от имеющихся изданий методикой изложения, в основе которой лежат принципы последовательного перехода от известной информации к новой, а также принцип сравнения подходов к написанию программ на языках Pascal и C++. При этом упор делается на практическую сторону процесса программирования: приводится большое количество примеров программ, заданий для самостоятельного решения, контрольных вопросов, рассматриваются зарекомендовавшие себя приемы и методы программирования.

Пособие состоит из глав, освоение которых предполагается осуществлять последовательно. Каждая тема начинается краткой, но достаточной по объему и содержанию теоретической частью с приведением множества иллюстрационных примеров и заканчивается типовым примером программы с подробными комментариями.

В первой главе рассматриваются элементы языков программирования Pascal и C++, а также структура программ. Во второй главе изложены элементарные подходы к написанию линейных программ. Третья глава посвящена разработке программ, реализующих разветвления. Четвертая глава описывает составление и отладку программ с циклической структурой. Пятая глава рассматривает вопросы создания и применения подпрограмм. В шестой главе даются основы обработки строковой и символьной информации. Седьмая глава посвяще-

на введению в структуры данных. В восьмой главе изложены основы работы с файлами на языке Pascal. Девятая и десятая главы раскрывают основы программирования в Pascal с использованием модулей CRT и GRAPH, соответственно. В одиннадцатой главе изложены основы работы с указателями на языке C++.

Каждая глава заканчивается контрольными вопросами для проверки усвоения изложенного материала. Кроме того, после каждой главы предлагается самостоятельное выполнение набора упражнений для углубленного изучения материала и получения дополнительных навыков в области структурного программирования.

Методическое построение и содержание учебного пособия способствуют успешному формированию следующих компетенций у бакалавров направления 230100 «Информатика и вычислительная техника»: - владеет культурой мышления, способен к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения; - осознает сущность и значение информации в развитии современного общества, владеет основными методами, способами и средствами получения, хранения, переработки информации; - разрабатывать интерфейсы «человек – электронно-вычислительная машина»; - разрабатывать компоненты программных комплексов и баз данных, использовать современные инструментальные средства и технологии программирования.

Материал предлагаемого пособия основан на курсе «Программирование», читаемом авторами в течение последних десяти лет в филиале Самарского государственного технического университета в г. Сызрани.

Замечания и предложения по улучшению книги просьба направлять по адресу: 446001, Самарская область, г. Сызрань, ул. Советская, 45, Филиал ФГБОУ ВПО «СамГТУ» в г. Сызрани.

ВВЕДЕНИЕ

Современное программирование к настоящему моменту фактически выделилось в отдельное направление в рамках информационных технологий с соответствующими подразделами. Программирование получило достаточно широкое распространение для решения различных прикладных задач, при котором используются базовые подходы и методы, начиная от структурного программирования и заканчивая паттерным.

Освоение программирования начинается, как правило, с изучения структурного подхода к написанию программ, что и закладывает основы алгоритмического мышления, формирует базовые знания в этой области и умения грамотно писать программы, а также постепенно подводит к дальнейшему освоению объектно-ориентированного программирования.

Изучение программирования прямо или косвенно связано с освоением алгоритмических языков и их практическим использованием для обработки числовой и символьной информации.

Среди множества алгоритмических языков особое место занимает Pascal, который, по общему мнению специалистов, с точки зрения обучения программированию является незаменимым. Pascal был создан швейцарским ученым Никлаусом Виртом как средство для обучения своих студентов основам программирования. Он был назван в честь французского математика и философа Блеза Паскаля (1623-1662). Язык оказался чрезвычайно удачным и приобрел более широкую сферу применения, чем предусматривалось его создателем. Для Pascal был разработан ряд международных стандартов, также этот язык получил дальнейшее развитие в вид Object Pascal. Особенно большое признание среди пользователей пришло к языку после разработки фирмой Borland интегрированной среды программирования Turbo Pascal для персональных компьютеров.

Другим широко используемым для разработки программного обеспечения профессиональным языком программирования является язык C++, который был создан в начале 1980-х годов сотрудником фирмы Bell Labs Бьёрном Страуструпом, как усовершенствованный язык C. Страуструп добавил к языку C возможность работы с классами и объектами. Таким образом, был получен достаточно мощный инструмент, получивший дальнейшее развитие и использующийся в настоящее время профессиональными программистами во всем мире. Область применения языка C++ очень широка и включает создание операционных систем; драйверов устройств; приложений для встраиваемых систем, высокопроизводительных платформ, ну и конечно же решение разнообразных прикладных задач. В своем дальнейшем развитии язык C++ оказал существенное влияние на такие языки как Java и C#. C++ осуществляет поддержку различных стилей программирования: структурное, объектно-ориентированное, обобщенное, функциональное программирование, порождающее метапрограммирование. Все это обусловило достаточно широкий спектр задач, решаемых с помощью языка C++ и выбор его в качестве основного инструмента профессионалами.

Возможности языков Pascal и C++ рассматриваются поэтапно, придерживаясь принципа "от простого к сложному". Поэтому желательно последовательное изучение материала, начиная с первой главы, в которой рассматриваются элементы языков и непосредственно вопросы программирования с использованием различных типов данных, начиная с простых (целых и вещественных) и заканчивая сложными структурированными типами (записи и файлы), включая указатели.

Учебное пособие рассчитано на активное изучение языков Pascal и C++, а также освоение основ работы в соответствующих интегрированных средах разработки.

Освоение материала данного учебного пособия способствует дальнейшему успешному изучению технологий и подходов програм-

мирования для решения различных задач на самых разных программных и аппаратных платформах.

1. ЭЛЕМЕНТЫ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ И СТРУКТУРА ПРОГРАММ

1.1. Элементы языка программирования Turbo Pascal

К элементам любого языка программирования относят алфавит, т.е. набор символов, используемых в программе, зарезервированные слова, знаки операций, типы данных и базовые конструкции.

Алфавит языка Turbo Pascal включает следующее множество символов:

- прописные и строчные латинские буквы **A .. Z**, **a .. z**;
- десятичные цифры **0 .. 9**;
- специальные символы:

+ – * / = < > . ′ , : ; ( ) [ ] { } ^ @ $ # _

При обработке текста программы компилятор не делает различий между верхним и нижним регистрами клавиатуры, то есть строчные и прописные символы латинского алфавита интерпретируются одинаково. Однако это не распространяется на переменные символьного и строкового типов.

Зарезервированные (служебные) слова имеют строго определенное назначение, которое не может быть изменено. Их нельзя использовать в качестве идентификаторов.

Основные служебные слова:

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

Зарезервированные слова выделяются на экране белым цветом, что исключает необходимость их запоминания.

Знаки операций. Основные операции языка Turbo Pascal в соответствии с их приоритетом приведены в табл. 1.1.

Таблица 1.1

Основные операции TP

Операции	Знак операции	Действие
Унарные	**not**	Отрицание
Мультипликативные	*****	Умножение
	/	Деление
	div	Целочисленное деление
	mod	Остаток от целочисленного деления
	and	Логическое И
Аддитивные	**+**	Сложение
	–	Вычитание
	or	Логическое ИЛИ
	xor	Исключающее ИЛИ
Отношения	**=**	Равно
	<>	Не равно
	<	Меньше
	>	Больше
	<=	Меньше или равно
	>=	Больше или равно

Типы данных. Данные – это конкретные значения, которые обрабатываются во время выполнения программы. В языке Turbo Pascal любые данные принадлежат к тому или иному типу.

Тип данных определяется

1) множеством допустимых значений;

2) множеством допустимых операций;

3) форматом внутреннего представления данных в памяти компьютера.

Наиболее часто используемые при решении вычислительных задач типы данных приведены в табл. 1.2 и 1.3. Другие типы будут рассматриваться далее после изучения основ программирования.

Таблица 1.2

Типы целых чисел

Название типа		Размер	Диапазон чисел
Короткое целое	Shortint	1 байт	-128…127
Целое	Integer	2 байта	-32768…32767
Длинное целое	Longint	4 байта	-2147483648…2147483647
Байт	Byte	1 байт	0…255
Слово	Word	2 байта	0…65535

Таблица 1.3

Типы вещественных чисел

Название типа		Размер	Диапазон чисел
Одинарной точности	Single	4 байта	$1.5\cdot10^{-45}...3.4\cdot10^{38}$
Действительное	Real	6 байт	$2.9\cdot10^{-39}...1.7\cdot10^{38}$
Двойной точности	Double	8 байт	$5.0\cdot10^{-324}...1.7\cdot10^{308}$
Расширенное	Extended	10 байт	$3.4\cdot10^{-4932}...1.1\cdot10^{4932}$
Составное	Comp	8 байт	$-9.2\cdot10^{18}...9.2\cdot10^{18}$

Базовые конструкции языка. К базовым конструкциям языка относятся идентификаторы, константы, метки, переменные, стандартные функции, выражения.

Идентификаторы – это имена, используемые для записи констант, переменных, типов, функций, процедур, модулей и т.д. Идентификатор может содержать буквы, цифры и символы подчеркивания, но он не должен иметь разрывов и не может начинаться с цифры. Например: **a**, **x**, **y5**, **Primer_1**, **logarifm**, **Smirnov_I_N**.

Длина идентификатора не ограничена, однако компилятор распознает только первые 63 символа.

Константы – это не изменяющиеся в программе величины. Они могут принадлежать к одному из следующих типов: целому, действительному, логическому, символьному и строковому.

Метки – целые числа в диапазоне от 0 до 9999 или идентификаторы, используемые в операторе перехода goto.

Переменные – изменяющиеся в ходе выполнения программы величины. Переменным следует давать короткие, отражающие их назначение имена.

Стандартные функции служат для облегчения вычисления наиболее часто встречающихся математических функций и входят в состав системы программирования. Основные стандартные функции приведены в табл. 1.4.

Таблица 1.4

Основные стандартные математические функции TP 7.0

Имя функции	Назначение функции	Тип аргумента	Тип функции
abs(x)	|x| – модуль аргумента	real или integer	real или integer
sqr(x)	x^2 – квадрат аргумента		
sin(x)	sin x – синус, x – в радианах	real	real
cos(x)	cos x – косинус, x – в радианах		
exp(x)	e^x – показательная функция		
ln(x)	ln x – натуральный логарифм		
sqrt(x)	$\sqrt{x}$ - корень квадратный		
arctan(x)	arctg x – арктангенс (в радианах)		
pi	Значение $\pi = 3.141592653\ldots$		
int(x)	Выделение целой части аргумента		
frac(x)	Выделение дробной части аргумента		
trunc(x)	Выделение целой части числа	real	longint
round(x)	Округление числа до целого		
odd(x)	Проверка аргумента на нечетность	longint	boolean

Некоторые математические функции можно выразить через стандартные. Например: **lg x = ln(x)/ln(10),** $\mathbf{a^x = exp(x * ln(a))}$**.**

Выражения определяют действия и последовательность вычисления значения. Они могут состоять из констант, переменных, функций, разделенных скобками и знаками операций.

Например: **a*d*(x/sqrt(pi*r))**, **(4.57*y)/sqr(x)**, **exp(sqrt(x*y))**.

Комментарии – заключенная в фигурные { } или комбинированные (* *) скобки информация, облегчающая восприятие программы. В комментариях допускается использование букв русского алфавита.

Например: **{поиск наименьшего элемента}**, **(*Ввод массива*)**.

Компилятор игнорирует комментарии, а в окне редактора они высвечиваются серым цветом.

1.2. Структура программы на языке Turbo Pascal

Все программы, составленные на языке программирования Turbo Pascal, имеют следующую общую структуру.

program <идентификатор>; {заголовок программы}

Раздел описаний:

uses <идентификатор>; {Uses-часть}
label <целое число>, <идентификатор>; {описание меток}
const <идентификатор>=<выражение>; {описание констант}
type <идентификатор>=<тип>; {описание типов}
var <идентификатор>: <тип>; {описание переменных}
procedure <идентификатор>; {описание процедур}
function <идентификатор>; {описание функций}

begin
<операторная часть> {программный блок}
end.

Заголовок программы не обязателен, т.к. компилятор его игнорирует, но лучше его указывать, причем желательно, чтобы имя программы совпадало с именем файла на диске.

Например: **Program** Sort;

Uses-часть определяет все модули (отдельно транслируемые программные единицы), которые будут использоваться в данной программе.

Например: **Uses** Printer;

где Printer – стандартный модуль, обеспечивающий вывод результатов программы на печать.

Если в программе модули не используются, то uses-часть опускается. Следует заметить, что основной модуль TP System, в который входят все математические функции и другие важные процедуры, всегда подключается по умолчанию и поэтому в uses-предложении не указывается.

Раздел описаний в общем случае включает описания меток, констант, типов, переменных, процедур и функций.

Описание меток начинается со слова **label**, за которым следует перечисление меток, используемых в программе, через запятую. Например: **label** 156, Start, L1, L2;

Описание констант открывается ключевым словом **const**, далее идет список имен констант и их значений. Имя и значение разделены знаком =, каждое описание заканчивается точкой с запятой (;). В качестве констант можно использовать выражения, составленные из них и некоторые стандартные функции.

Например: **const** Max=255; N=25*Pi; sym=’B’; K=Max/N;

Описание переменных начинается с ключевого слова **var**, за которым следует перечисление через запятую имен переменных, затем ставится двоеточие, после которого указывается идентификатор типа.

Например: **var** x,y: real; mult: double; c,ch: char;

Этот раздел присутствует всегда, т.к. переменные лежат в основе программы.

Другие описания (**type**, **procedure**, **function**) будут рассмотрены позже.

В разделе объявлений каждое описание заканчивается точкой с запятой(;). Состав этого раздела не постоянен и может меняться в зависимости от участвующих в программе конструкций. Например, если в программе не используются метки и константы, то соответственно разделы **label** и **const** в программе будут отсутствовать. Кроме того, порядок следования описаний произволен, требуется выполнять только одно условие – используемые элементы должны быть сначала описаны (определены).

Раздел операторов включает последовательность операторов и команд, образующих собственно программу (вычислительный процесс).

Оператор – элементарная структурная единица программы, которая задает некоторое законченное действие, логически эквивалентное элементарному шагу алгоритма.

Операторы отделяются друг от друга точкой с запятой. Любой из них может быть снабжен меткой, используемой для передачи управления этому оператору.

Операторная часть начинается с ключевого слова **begin** и заканчивается ключевым словом **end**, после которого ставится точка.

1.3. Элементы языка программирования C++

Язык C++ состоит из следующих элементов:

- алфавит языка - основные неделимые символы;
- лексемы - элементарные конструкции, образованные посредством алфавита;
- выражения, которые образованы из лексем и символов для организации вычисления некоторого значения;
- операторы, которые задают описание законченного действия.

Алфавит языка C++ включает следующее множество символов:

- прописные и строчные латинские буквы **A .. Z**, **a .. z**;

- арабские цифры: **0 .. 9**;

- специальные символы:

+ – * / % = < > ! . ′ ," : ; ( )[ ] { } | \ ? _ # & ~ ^

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые слова, знаки операций, константы, разделители.

Идентификаторы – это имена программных объектов, которые используются для записи констант, переменных, типов, функций и т.д. Идентификатор может содержать латинские буквы, цифры и знак подчеркивания и должен начинаться с латинской буквы или знака подчеркивания, не иметь разрывов и не совпадать с ключевым словом. Следует заметить, что в языке С++ различаются прописные и строчные буквы.

Например:

Mass, MASS и mass – три разных имени.

Длина идентификатора не ограничена, однако компиляторы фирмы Borland различают не более 32-х первых символов любого идентификатора.

Объектам рекомендуется давать осмысленные имена и не начинать идентификатор со знака подчеркивания, так как он может совпасть с системным именем.

Например:

matr_a, Ivanov_N_1, Program_5.

Константы – это не изменяющиеся в программе величины. Они могут принадлежать к одному из следующих типов: целому, вещественному, символьному и строковому. Компилятор различает константы по внешнему виду и относит их к одному из типов по умолчанию.

Целые константы разделяются на десятичные, восьмеричные (начинаются с О) и шестнадцатеричные (начинаются с ОХ).

Например:

6, 69, 264 - десятичные;

01, 07, 0475 - восьмеричные;

0X95, 0X3A6 - шестнадцатеричные.

Вещественные константы имеют два формата представления:

• десятичный, где целая часть числа отделяется от дробной части десятичной точкой;

Например: 5.8; -4.25

• экспоненциальный формат состоит из трех частей: мантиссы, знака экспоненты и десятичного порядка.

Например:

-1.25E+5,

где -1.25 - мантисса; E - основание числа 10; +5 - порядок.

Символьные константы – один или два символа, заключенные в апострофы.

Например:

'A' , '\n', 'AB'.

Символ **** используется для представления кодов, не имеющих графического изображения и для изображения некоторых символов:

\a – звуковой сигнал;

\b- возврат на шаг;

\f – перевод страницы;

\n – перевод строки;

\t - горизонтальная табуляция;

\v – вертикальная табуляция;

\\ - обратная косая черта;

\` - апостроф;

\" – кавычка;

\? – знак вопроса.

Строковые константы – последовательность символов, заключенных в кавычки.

Например:

"Borland C++ 5.0"

Если константа имеет большой размер, то для ее переноса на следующую строку можно использовать символ **** - обратный слэш.

Например:

```
"Borland \
      C++ 5.0"
```

При выводе на экран или печать второй вариант будет выглядеть как первый: Borland C++ 5.0.

Если внутри строковой константы имеются кавычки (слова, заключенные в кавычки), то перед ними ставится символ ****.

Например:

```
"Программирование на C++. Издательство \"Бином\" 2004"
```

Ключевые (зарезервированные) слова имеют строго определенное функциональное назначение, которое изменять нельзя. Стандарт языка C++ содержит 63 ключевых слова, однако различные компиляторы могут увеличивать их количество, учитывая возможности компьютеров, для которых они созданы.

Список основных ключевых слов

asm	do	if	reinterpret_cast	true
auto	double	int	return	try
bool	else	long	short	typedef
break	enam	mutable	signed	typeid
case	explixcit	near	siseof	typename
char	export	new	static	union
class	extern	operator	struct	unsigned
const	falsh	private	switch	using
continue	float	protected	template	virtual
default	for	public	this	void
delete	goto	register	throw	while

Типы данных. Данные – это конкретные значения, которые обрабатываются во время выполнения программы. В языке C++ любые данные принадлежат к тому или иному типу.

Тип данных определяет:

1) множество допустимых значений;

2) множество допустимых операций;

3) формат внутреннего представления данных в памяти компьютера.

Для определения и описания базовых типов данных используются следующие ключевые слова:

- char - символьный;
- int - целый;
- bool - логический;
- float - вещественный;
- double - вещественный с двойной точностью.

Основные типы данных, часто используемые при решении вычислительных задач, приведены в табл. 1.5.

Таблица 1.5

Типы числовых данных

Название	Обозначение	Размер в байтах [1]	Диапазон значений
1	2	3	4
Короткое целое	[signed] **short** [int]	2	-32768 ÷ 32767
Короткое целое без знака	**unsigned short** [int]	2	0 ÷ 65535
Целое	[signed] **int**	2	-32768 ÷ 32767
Целое без знака	**unsigned** [int]	2	0 ÷ 65535
Длинное целое	[signed] **long** [int]	4	-2147483648÷ 2147483647
Длинное целое без знака	**unsigned long** [int]	4	0 ÷ 4294967295
Вещественное одинарной точности	**float**	4	3.4e-38 ÷ 3.4e+38
Вещественное двойной точности	**double**	8	1.7e-308 ÷ 1.7e+308
Вещественное увеличенной точности	**long double**	10	3.4e-4932 ÷ 3.4e+4932

[1] Для 16 – разрядной памяти.

Разделители - знаки пунктуации, которые входят в лексемы языка. Некоторые основные функции разделителей указаны в табл. 1.6.

Таблица 1.6

Разделители

Разделитель	Краткое описание
[ ]	Ограничивают индексы массивов
( )	Выделяют условное выражение, определяют указатель на функцию, изменяют последовательность выполнения операции и др.
{ }	Ограничивают составной оператор или блок программы, используются при инициализации массивов и структур
,	Разделяют элементы списков
;	Завершают каждый оператор
:	Отделяет метку и помечаемый ей оператор
…	Обозначает переменное число параметров функции
*	Операция умножения, операция разыменования
=	Операция присваивания
#	Обозначение директив препроцессора
&	Разделитель при определении переменных типа ссылки

Знаки операций – это один или несколько символов, определяющих действие над операндами. В зависимости от количества операндов, участвующих в операции они делятся на:

1) унарные (операции с одним операндом);

2) бинарные (операции с двумя операндами);

3) тернарные (операции с тремя операндами).

В табл. 1.7 приведены основные бинарные арифметические операции, в соответствии убывания приоритетов. Остальные операции будут вводиться по ходу изложения материала.

Таблица 1.7

Основные бинарные операции

Знак операции	Вид операции
*	Умножение
/	Деление
%	Остаток от деления
+	Сложение
-	Вычитание

Основные **математические функции**, применяемые при программировании на С++, приведены в табл. 1.8.

Таблица 1.8

Математические функции

Функция	Краткое описание действий
abs(x)	int **abs**(int **x**) – возвращает абсолютное значение целого аргумента x
acos(x)	double **acos**(double **x**) – арккосинус при $\|x\| \le 1$
asin(x)	double **asin**(double **x**) – арксинус при $\|x\| \le 1$
atan(x)	double **atan**(double **x**) – арктангенс
atan2(y,x)	double **atan2**(double **y**, double **x**) - арктангенс от значений y/x
cos(x)	ddouble **cos**(double **x**) – функция косинуса (аргумент x задается в радианах)
exp(x)	double **exp**(double **x**) – возвращает значение e^x
fabs(x)	double **fabs**(double **x**) – возвращает абсолютное значение аргумента x двойной точности
log(x)	double **log**(double **x**) – возвращает значение натурального логарифма ln(x)
log10(x)	double **log10**(double **x**) – возвращает значение десятичного логарифма log(x)
pow(x,y)	double **pow**(double **x**, double **y**) – возвращает значение x^y
pow10(p)	double **pow10**(int **p**) – возвращает значение 10^p

Окончание табл. 1.8

sin(x)	double **sin**(double **x**) – функция синуса (угол (аргумент) задается в радианах)
sqrt(x)	double **sqrt**(double **x**) – возвращает положительное значение квадратного корня $\sqrt{x}$
tan(x)	double **tan**(double **x**) – функция тангенса (угол (аргумент) задается в радианах)

1.4. Структура программы

Все программы, составленные на языке программирования С++ состоят из функций, описаний, директив препроцессора, обычно снабжаются подробными комментариями и имеют следующую общую структуру:

```
# директивы препроцессора
описание глобальных объектов
void main( )                    //Заголовок глобальной функции
{ описание локальных объектов
  операторы главной функции
}
int func_1(параметры)           //Заголовок локальной функции_1
{ описание локальных объектов
  операторы функции func_1
 }
. . . . . .
 double func_n(параметры)       //Заголовок локальной функции_n
{ описание локальных объектов
  операторы функции func_n
 }
```

Директивы препроцессора

Директива – это инструкция для компилятора. Он распознает ее по знаку **#** и выполняет в первую очередь, вставляя в программу за-

ранее подготовленные тексты из включаемых файлов:

#include <имя включаемого файла>

Например:

#include <iostream.h> //подключение файла, служащего для

// управления вводом и выводом данных

#include <math.h> //подключение математических функций

Описание объектов

Объект в С++ определяется как некоторая область памяти, в которой располагаются константы, функции, массивы и т.д.

Одним из наиболее используемых объектов как именованной области памяти является **переменная** (идентификатор). С ее именем можно связывать различные значения, совокупность которых определяется типом переменной.

Общий вид описания переменных:

<тип> <список идентификаторов>;

Например:

int delta; //переменная delta типа int

double x, y, z; // список переменных типа double

При описании переменные могут быть инициализированы. В этом случае переменной можно присвоить начальное значение двумя способами:

<идентификатор>=значение

или

<идентификатор>(значение).

Например:

float x=7.4; int c(2);

Если включить в описание переменной модификатор **const**, то в результате получится именованная константа, которую в дальнейшем изменять нельзя.

Например:

 const int a=5;

const float b=7.24;

Вводить константы также можно с помощью препроцессорной директивы:

#define <идентификатор> <значение>;

Например:

#define F 1.713232;

Функции

В языке С++ функция – это основное понятие, так как любая программа состоит из функций. Это соответствует модульному принципу программирования, согласно которому исходная задача разбивается на несколько подзадач. Программа, как правило, состоит из нескольких функций, одна из которых имеет имя **main**. С нее начинается выполнение программы. Функция **main** имеет следующий вид:

void main( )
{тело функции}

Тип **void** показывает, что функция не типизирована, т.е. не должна возвращать значения, а круглые скобки указывают, что этот программный блок называется функцией. Тело функции заключается в фигурные скобки **{ }** и представляет собой последовательность **операторов**, которыми являются выражения, заканчивающееся символом точка с запятой (**;**).

Например, простейшая программа вычисления значений функции $y=2\sqrt{x}$ будет иметь следующий вид:

```
#include <iostream.h>
#include <math.h>
void main()
{int x; float y;                    //Описание переменных
 cout<<"Введите значение x – целое положительное число";
 cout<<"x=";cin>>x;     //Ввод переменной x с клавиатуры
```

```
  y=2*sqrt(x);            //Вычисление значения функции
  cout<<"y="<<y;          //Вывод результата
}
```

Комментарии

Комментарии или пояснения начинаются с двух символов "косая черта" (//) и заканчиваются переходом на новую строку или заключаются в скобки /* и */. Компилятор комментарии игнорирует, поэтому внутри них можно размещать любой текст, в котором разрешено использовать не только символы из алфавита языка С++, но и русские буквы.

Например:

```
// вывод полученных результатов
// и исходных данных
```

или

```
/* вывод полученных результатов
   и исходных данных */
```

Контрольные вопросы

1. Какие арифметические операции реализованы в ТР?
2. Какие типы целых чисел Вы знаете?
3. Какие типы вещественных чисел Вы знаете?
4. Перечислите признаки, по которым данные относят к тому или иному типу.
5. Перечислите основные стандартные функции, используемые в среде ТР.
6. Приведите примеры описания переменных и констант.
7. Какие символы включает в себя алфавит языка ТР?
8. Приведите примеры зарезервированных слов.
9. Перечислите базовые конструкции языка ТР 7.0.

10. Приведите примеры записи выражений.

11. Опишите общую структуру программ, составленных на языке ТР.

12. Что такое оператор и какими словами начинается и заканчивается операторная часть программы?

13. Перечислите обязательные и необязательные разделы программы.

14. Из каких элементов состоит язык С++?

15. Какие лексемы формируются из символов алфавита?

16. Перечислите основные математические функции, используемые в языке С++.

17. Приведите примеры описания переменных и констант.

18. Что такое зарезервированные слова.

19. Какие арифметические операции реализованы в С++ ?

20. Что определяет тип данных?

21. Какие типы целых чисел Вы знаете?

22. Какие типы вещественных чисел Вы знаете?

23. Поясните особенности структуры программы на С++.

24. Что такое директивы препроцессора?

25. Каким образом подключаются заголовочные файлы ввода-вывода?

26. Поясните особенности главной функции **main**.

2. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

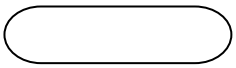
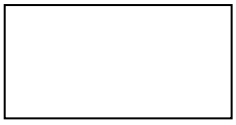
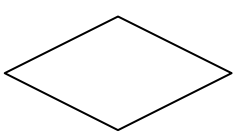
Линейные алгоритмы - это простейшие алгоритмические структуры, отображающие вычислительный процесс, в котором все операции выполняются последовательно, в порядке их записи, без каких-либо условий.

Распространены следующие формы представления алгоритмов:

- словесно-формульная – это запись алгоритма на естественном языке с приведением формульных соотношений;
- графическая форма представления алгоритма в виде блок-схемы, состоящей из условных графических обозначений, в которых обычно указываются производимые операции (табл. 2.1);
- программная – представление алгоритма в виде формализованного текста на алгоритмическом языке.

Таблица 2.1

Условные графические обозначения основных блоков схем алгоритмов

Обозначение блока	Наименование	Функции
	Пуск - останов	Начало, конец и прерывание процесса обработки данных
	Процесс	Выполнение операции или группы операций
	Ввод-вывод	Ввод или вывод данных в независимости от физического носителя
	Решение	Проверка условия и выбор направления выполнения алгоритма
	Предопределенный процесс	Выполнение подпрограммы

Окончание табл. 2.1

Обозначение блока	Наименование	Функции
	Модификация	Заголовок оператора цикла For

Для развития алгоритмического мышления на начальном этапе целесообразно использовать последовательно все три формы представления алгоритмов.

В общем случае процесс разработки программы предполагает выполнение следующих этапов.

1. Разработка алгоритма решения задачи в словесной форме и, если требуется, - в виде блок-схемы.
2. Составление текста программы по разработанному алгоритму.
3. Ввод программы в компьютер.
4. Компиляция программы – проверка на наличие синтаксических ошибок. Если в программе были допущены ошибки, на экране появится соответствующее сообщение, а курсор укажет ориентировочное место ошибки.
5. Отладка программы - это этап поиска логических ошибок на основе анализа данных выполнения программы. Если получаются неправильные результаты, необходимо исправить допущенные алгоритмические ошибки и затем снова запустить программу на выполнение.
6. Запуск программы на выполнение.

2.1. Программирование линейных алгоритмов на языке Turbo Pascal

Разработка программы предполагает выполнение следующих этапов:

1. Составление текста программы по разработанному алгоритму.
2. Ввод программы в компьютер.

3.Запуск программы на выполнение (команда **Ctrl+F9**).

При этом сначала осуществляется проверка программы на наличие синтаксических ошибок. Если в программе были допущены ошибки, на экране появится соответствующее сообщение, а курсор укажет ориентировочное место ошибки. В этом случае последнюю необходимо исправить и снова запустить программу. После исправления последней ошибки программа переводится компилятором в машинные коды и далее автоматически запускается на выполнение.

4. Отладка программы.

Это этап поиска логических ошибок на основе анализа данных выполнения программы. Если получаются неправильные результаты, необходимо исправить допущенные алгоритмические ошибки и затем снова запустить программу на выполнение.

Для организации программ линейной структуры используются операторы присваивания, составной оператор, операторы ввода и вывода данных.

Оператор присваивания служит для присваивания переменной значения выражения. Общий вид записи оператора:

```
v:=s;
```

где v – имя переменой,

s – выражение,

: = – знак присваивания.

Например: b:=0.125; x:=sqrt(z*z+f*f);

Переменная и выражение должны принадлежать одному типу.

Составной оператор – последовательность операторов, заключенных в begin и end.

Например:

```
begin
 g:=k;
 x:=x+a;
end;
```

Ввод (считывание) информации осуществляется с помощью операторов (процедур ввода) **Read** и **Readln**:

Read (v1, v2, ... ,vn); Readln (v1, v2, ... ,vn);

где vi – имена переменных, значения которых вводятся с клавиатуры.

Если требуется сделать в программе останов, то используется оператор Readln без списка параметров. По существу, это ввод пустой строки (ожидание нажатия на Enter).

Например: Read (x, y, z); Readln (a); Readln;

Вывод информации на экран осуществляется с помощью операторов **Write** и **Writeln** (процедур вывода):

Write (v1, v2, ... ,vn); Writeln (v1, v2, ... ,vn);

где vi – имена переменных, значения которых выводятся на экран.

Если в пограмме требуется вывести пустую строку, то используется оператор Writeln без параметров.

Например: Write (x, y); Writeln (b); Writeln;

В процедуре вывода при необходимости указывается ширина поля под запись числа:

- для целых чисел в виде write (v : m);
- для действительных чисел – write (v : m : n);

где m – число позиций под запись всего числа (включая точку для действительного числа);

n – число позиций под дробную часть.

Например: writeln (x : 12); write (s : 10 : 3);

Если для вещественного типа параметр n не указан, то число выводится в нормализованной форме.

Выводимые на экран сообщения заключаются в апострофы.

Например: Writeln ('Введите переменные:'); Write ('y=', y : 12 : 4);

В Турбо Паскале имеется возможность использования *типизированных констант*. В отличие от простых констант они могут изменять свое значение в ходе выполнения программы.

Типизированные константы задаются в разделе объявлений в следующем виде:

Const

<имя константы> : <тип> = <значение константы>;

Например:

Const

k : integer = 10;

eps : real = 0.001;

Фактически типизированные константы представляют собой переменные определенного типа с заданными начальными значениями.

Пример. Вычислить площадь треугольника по формуле Герона: $S = \sqrt{p(p-a)(p-b)(p-c)}$, где a, b, c – длины сторон треугольника, $p = \frac{a+b+c}{2}$ - полупериметр;

при a=3.45, b=4.21, c=5.32.

Словесно-формульный алгоритм:

1. Ввод значений a, b и c.
2. Вычисление полупериметра $p = \frac{a+b+c}{2}$.
3. Вычисление площади треугольника $S = \sqrt{p(p-a)(p-b)(p-c)}$
4. Вывод значения S на печать.

Текст программы:

```
Program AreaTriangl;
   var a,b,c,p,s: real;
    Begin
      writeln ('Введите длины сторон треугольника');
```

```
{ввод длин сторон треугольника}
        write ('a='); readln(a);
        write ('b='); readln(b);
        write ('c='); readln(c);
        p:=(a+b+c)/2;              {вычисление полупериметра}
        s:=sqrt(p*(p-a)*(p-b)*(p-c));      {вычисление площади}
{вывод результатов расчета на экран}
        writeln ('Площадь треугольника S=',s:12);
        readln;
{вывод результатов расчета на печать}
        writeln('Результаты решения задачи');
        writeln;
        writeln ('Площадь треугольника S=',s:12);
        writeln
      End.
```

Результат решения задачи:

```
Площадь треугольника S= 7.25470E+00
```

2.2. Составление линейных программ на языке программирования C++

Вычислить значения функций:

$$z = \sqrt{kx^2 + 7{,}75} - e^{-m}\sin\left(m + \ln\left|y^3 - 0{,}5\right|\right);$$

$$x = \frac{\ln\left(\text{arctg}(ab + k)\right) - 5}{\cos^4(a+b) + \sqrt{m}};\ y = \frac{\lg\left(ax^2 + bx + m\right)}{2^{kx-1} + 0{,}125},$$

где a, b – переменные; k=0.3, m=2 - константы.

Словесно-формульный алгоритм

Ввиду того, что переменная **z** зависит от **x** и **y**, а **y** – только от **x**, сначала вычисляется переменная **x**, затем **y** и в заключение **z** следующим образом.

1. Ввод исходных значений a, b.

2. Задание значений констант k=0.3, m=2.

3. Вычисление **x** с использованием промежуточных переменных: x1, x2 и x3:

x1=arctg(ab+k);

x2=ln(x1)-5;

$x3 = \cos^4(a+b) + \sqrt{m}$;

x=x2/x3.

4. Вычисление **y** с использованием промежуточных переменных y1 и y2:

$y1=lg(ax^2+bx+m)$;

$y2=2^{kx-1}+0.125$;

y=y1/y2.

5. Вычисление **z** с использованием промежуточных переменных: z1, z2 и z3:

$z1=ln|y^3-0.5|$;

$z2=e^{-m} \sin(m+z1)$;

$z3 = \sqrt{kx^2 + 7{,}75}$;

z=z3-z2.

6. Вывод полученных результатов (**x, y, z**) и исходных данных (**a, b**).

Необходимыми элементами всех программ являются операции присваивания, функции ввода и вывода данных.

Операции присваивания

Оператор присваивания служит для присваивания переменной значения выражения. Общий вид записи операции присваивания:

<имя переменной><знак присваивания><выражение>;

Для простого присваивания используется знак равенства (**=**).

Например:

a=1.475; x=sin(a);

Ввод и вывод данных

В С++ нет встроенных средств ввода-вывода данных. Данные операции осуществляются при помощи специальных функций, которые содержатся в стандартных библиотеках. Данные функции присоединяются с помощью соответствующих директив препроцессора.

С помощью директивы препроцессора **#include <iostream.h>** подключаются к программе стандартные объекты-потоки, использующие потоковый способ ввода-вывода:

cout<< - вывод данных на экран;

cin>> - ввод данных с клавиатуры.

Например**:**

```
cout<<"введите переменную a=";
cin>>a;
```

После выполнения команды cin программа останавливается и ждет ввода с клавиатуры данных соответствующего типа. Затем компьютер продолжает выполнение программы.

С помощью директивы препроцессора **#include<stdio.h>**, унаследованной из языка С, можно присоединять функции ввода-вывода:

printf( ) - вывод данных на экран;

scanf( ) - ввод данных с клавиатуры.

Например:

```
printf("введите переменную a=");
scanf(&a);        //знак & означает операцию получения адреса
```

Однако следует учитывать, что для корректной работы программы в одной программе не рекомендуется использовать одновременно оба способа ввода-вывода данных. В настоящее время предпочтение отдается потоковому вводу-выводу.

По разработанному в примере линейному алгоритму можно написать текст программы, записав алгоритм на языке С++ с применением простых операций присваивания, ввода и вывода (лист. 2.1).

Листинг 2.1. *lin_prog.cpp*

```
#include <iostream.h>          // директива подключения средств
                               //ввода-вывода:cin, cout
#include <math.h>              // директива подключения
                              //математических функций
void main()                    //заголовок главной функции
{
double a,b,x1,x2,x3,x,y1,y2,y,z1,z2,z3,z;     //объявление переменных
                    // вывод сообщения-подсказки на экран
cout<<" Введите исходные данные"<<endl; // endl- манипулятор
                                          // перевода строки
cout<<"a=";cin>>a;                        //ввод переменной a
cout<<"b=";cin>>b;                        //ввод переменной b
const int m=2;const double k=0.3;         //объявление констант k,m
//программирование выражений словесно-формульного алгоритма
     x1=atan(a*b+k);
     x2=log(x1)-5;
    x3=pow(cos(a+b),4)+sqrt(m);
    x=x2/x3;
    y1=log10(a*pow(x,2)+b*x+m);
    y2=pow(2,k*x-1)+0.125;
    y=y1/y2;
    z1=log(fabs(pow(y,3)-0.5));
    z2=exp(-m)*sin(m+z1);
    z3=sqrt(k*pow(x,2)+7.75);
    z=z3-z2;
                 // вывод сообщения-подсказки на экран
     cout<<"\n Значения вспомогательных переменных x,y\
              и конечного результата z равны:";
   //Символ \ - перенос литерной константы
        //вывод полученных результатов и исходных данных
     cout<<"\n x="<<x;                  // \n - символ перевода строки
     cout<<", y="<<y;
```

```
    cout<<", z="<<z<<endl;
    cout<<" при a="<<a<<", b="<<b<<endl;
}
```

!! *Проанализируйте программу, сопоставьте ее со словесным представлением алгоритма. Создав новый файл проекта с именем lin_prog.ide, наберите в нем текст данной программы, откомпилируйте и произведите запуск программы на выполнение.*

Результат выполнения программы

```
Введите исходные данные
a=1
b=1
Значения вспомогательных переменных x,y и конечного результата z равны:
x=-3.52355, y=2.839, z=3.51229
при a=1, b=1
```

!! Ознакомьтесь с организацией в программе технологических приемов вывода подсказок и результатов выполнения программы на экран монитора.

Упражнения

Вычислить значения функций **x, y** и **z,** зависящих от переменных **a** и **b** и констант **k, m**. Составить словесно-формульный алгоритм, затем записать его на языке программирования С++. Данные приведены в табл. 2.2.

!! *Создайте новый файл проекта и введите текст составленной программы в окно редактора. Произведите компиляцию программы для выявления синтаксических ошибок. Запустите программу на выполнение и проанализируйте ее результаты.*

Таблица 2.2

Варианты заданий

1	$z = \operatorname{arctg}\lvert y+1\rvert + \dfrac{2^{x+m}}{k}, \quad x = \dfrac{(a^3+3{,}5)\sqrt{b+8}}{a+bk}; \quad y = \dfrac{m+e^{x-1}}{1+a^2\cdot\lvert b-\operatorname{ctg}ak\rvert},$ при a=1, b=-1, k=0.5, m=3.14.
2	z=e(x+ky) + m4, $x = \dfrac{\sqrt{\lvert a-1\rvert+\sqrt[3]{\lvert b\rvert}}}{1+\dfrac{a^2}{2}+\dfrac{b^2}{4}}; \quad y = \operatorname{arctg}^2 x + \lg(1{,}4+b),$ при a=-0.3, b=2.5, k=1, m=2.1.
3	$z = k\cos x\cdot\sqrt{m+\ln\lvert m+2{,}7y\rvert}, \quad x = \dfrac{\sqrt[3]{e^a-\sin a}}{m+\ln(\sqrt{b^2+5}+0{,}82)}; \quad y = \dfrac{x^5}{5}+\log_2\dfrac{k}{5},$ при a=0.75, b=6.3, k=2, m=4.6.
4	$z = \dfrac{m^8}{kx+y^2}, \quad x = 2^{-a}\cdot\sqrt{a+\sqrt[4]{b}}; \quad y = \dfrac{0{,}75+\sin^4(x+k)}{2+m\left\lvert a-\dfrac{2a}{1+\lvert\sin(x+b)\rvert}\right\rvert},$ при a=2, b=3.7, k=-1, m=1.8.
5	$z = m\operatorname{tg}(x-5)^2\cdot\arccos ky, \quad x = \dfrac{e^{-2b}+\sqrt[m]{\lvert a^3\rvert}}{b^4\cdot\left(1{,}5+\dfrac{a-b}{a+b}\right)}; \quad y = \left(\dfrac{ab}{k}+\dfrac{m}{ab}\right)\cdot\ln\lvert x+a\rvert.,$ при a=-1.1, b=-3, k=0.1, m=2.
6	$z = 2a\lg(\sqrt{x^2+1}+m)+\cos^2(3y-k), \; x = \sin\operatorname{arctg}a-\dfrac{3^{m-b}}{k}; \; y = \dfrac{1{,}5b-\lg(x^2+2m)}{\cos(2e^{0{,}5k}-a)}$ при a=-2.7, b=1, k=3.14, m=-0.5.
7	$z = mx^y+2kxy-0{,}81; \quad x = ay-\dfrac{kb^5}{1{,}2+\sin^2(a+b)}; \quad y = \dfrac{1+tg^2\dfrac{a}{2}}{\left(\dfrac{a}{a+b}+2{,}1\right)\lg m},$ при a=0,6, b=-1,3, k=-1, m=15.
8	$z = -13{,}4\cdot10^{-3}x+e^{\lvert\sin y+k\rvert};$ $x = \dfrac{\sqrt{\lvert a^3-\ln m\rvert}}{\cos^2(ab-0{,}6)+1}; \quad y = \dfrac{\sin 5x^2-1{,}8tg(b+3m)}{\lg\left\lvert x+\sqrt{x^4-k}\right\rvert},$ при a=-3.6, b=0.4, k=1, m=2.4.

Контрольные вопросы

1. Чем характеризуется линейная структура алгоритма?
2. В чем заключается сущность словесно-формульного способа описания алгоритма?
3. Чем характеризуется линейная структура алгоритма?
4. Перечислите основные этапы составления программы?
5. Поясните структуру программы, составленную на ТР.
6. Расскажите о правилах записи составного оператора.
7. Поясните правила записи и использования оператора ввода.
8. Какой оператор служит для вывода данных?
9. Как задаются поля под запись значений чисел в операторах вывода?
10. Приведите примеры операций присваивания на С++.
11. Для чего в программе используют комментарии?
12. Каким образом можно организовать в программе на С++ поточный ввод и вывод данных?
13. Каким образом можно организовать в программе ввод и вывод данных с помощью операторов printf и scanf ?
14. Как организовать в программе комментарии на разных языках программирования?
15. Каким образом подсоединяются к программе файлы ввода-вывода?

3. РАЗРАБОТКА ПРОГРАММ С РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРОЙ

Базовая алгоритмическая структура **ветвление** обеспечивает в зависимости от результата проверки условия (**истина - true** или **ложь - false**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к общему выходу.

3.1. Программирование разветвляющихся алгоритмов на языке Turbo Pascal

Для организации ветвлений в программах применяются оператор перехода, оператор выбора (переключатель) и условный оператор. Они имеют соответственно следующий вид.

1. Оператор перехода

goto n ;

где n - метка. При этом меткой можно помечать любой оператор, включая пустой.

2. Оператор выбора

Case <селектор> **of**
<список выбора 1> **:** <оператор 1> **;**
. . . . . .
<список выбора N> **:** <оператор N>
[**else** <оператор>]
End ;

где <селектор> - выражение порядкового типа;

<список выбора> - константа или интервал из констант;

<оператор> - любой оператор языка Turbo Pascal (TP), включая составной.

Например:

вычислить значение функции $f = \begin{cases} b + x, & \text{если } b = 1, \\ bx + x^2, & \text{если } b = 2, \\ b\cos x, & \text{если } b = 3. \end{cases}$

```
. . . . . .
case b of
  1: f := b + x;
  2: f := b*x + sqr(x);
  3: f := b*cos x
end;
. . . . . .
```

3. Условный оператор:

а) в полной форме

if <логическое выражение> **then** <оператор 1>
else <оператор 2> **;**

б) в сокращенной форме

if <логическое выражение> **then** <оператор 1> **;**

где <оператор 1>, <оператор 2> - любые операторы ТР, включая условные.

Логическое выражение представляет собой условие, записанное с помощью логических операций **not**, **and**, **or**, **xor** и операций отношения (>, <, =, >=, <=, <>). Результат логического соотношения имеет два значения типа **Boolean**: **true** и **false**. При этом логические операции выполняются согласно табл. 3.1.

Например, при определении принадлежности точки (x,y) области $\{0 \le x \le x_{max},\ 0 \le y \le y_{max}\}$ условный оператор можно записать в следующем виде:

```
if (X>=0) and (X<=Xmax) and (Y>=0) and (Y<=Ymax)
   then writeln ('Точка принадлежит указанной области')
else writeln ('Точка не принадлежит указанной области');
```

Таблица 3.1

Логические операции

Переменные		**Операции**			
A	**B**	**not A**	**A and B**	**A or B**	**A xor B**
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

Пример. Составить программу вычисления значения функции

$$f = \begin{cases} 2ax + |a - 1|, & \text{если } ax < 0; \\ \dfrac{e^x}{\sqrt{1+a^2}} - 1, & \text{если } 0 \le ax < 4; \\ x^3 - 3a + 4, & \text{если } ax \ge 4 \end{cases}$$

при заданных значениях a и x.

Словесно-формульный алгоритм

1. Ввод значений a и x.
2. Вычисление значения промежуточной переменной b = ax.
3. Расчет функции f в зависимости от выполнения условия:

а) если b<0, то $f = 2b + |a - 1|$;

б) если 0≤b<4, то $f = \dfrac{e^x}{\sqrt{1+a^2}} - 1$;

в) если b≥4, то $f = x^3 - 3a + 4$.

4. Вывод значения f.

Блок-схема алгоритма представлена на рис. 3.1.

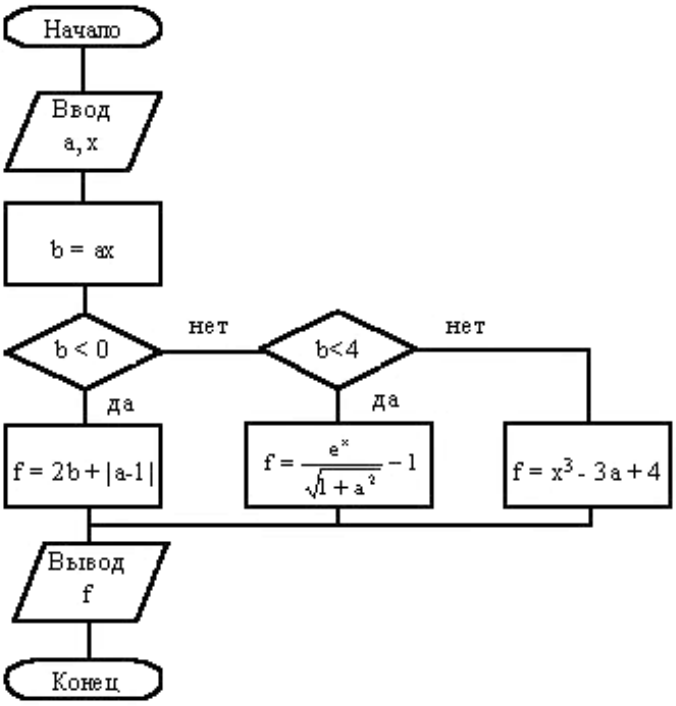


Рис. 3.1. Блок-схема алгоритма

Текст программы

```
Program Vetvi;                              {заголовок программы}
  Var a,b,x,c,d,f:real;                  {описание переменных}
   BEGIN                                  {начало операторной части}
      writeln('Введите переменные:');
      write('a='); readln(a);
      write('x='); readln(x);
       b:=a*x;
       if b<0 then f:=2*b+abs(a-1)
       else if b<4 then
                  begin
                    c:=exp(x);
                    d:=sqrt(1+a*a);
                    f:=c/d-1
                  end
            else f:=exp(3*ln(a))-3*a+4;
       writeln('f=',f:12,' при x=',x:5:2, ' и a=',a:5:2);
    END.                                  {конец операторной части}
```

!! *Запустите среду программирования Turbo Pascal, наберите текст вышеприведенной программы и сохраните файл с именем* ***Vetvi.pas***.

3.2.Программирование разветвляющихся алгоритмов на языке С++

Для организации ветвлений в программах применяются два оператора: условный оператор (**if**) и переключатель (**switch**).

Условный оператор имеет два вида:

а) полная форма

if ***(условие) оператор1;*** **else** ***оператор2;***

б) сокращенная форма

if ***(условие) оператор1***;

где ***оператор1, оператор2*** - любые операторы, включая условные и составные;

условие – в общем случае логическое выражение.

Например:

```
if (x<y) min=x; else min=y;
```

Если ***оператор1*** и ***оператор2*** представляют собой короткие выражения, то вместо условного оператора можно использовать условную операцию **? :**, в частности, для выше рассмотренного примера имеем:

```
min = (x<y) ? x : y;
```

Блок-схемы условного оператора для обеих форм приведены на рис.3.2.

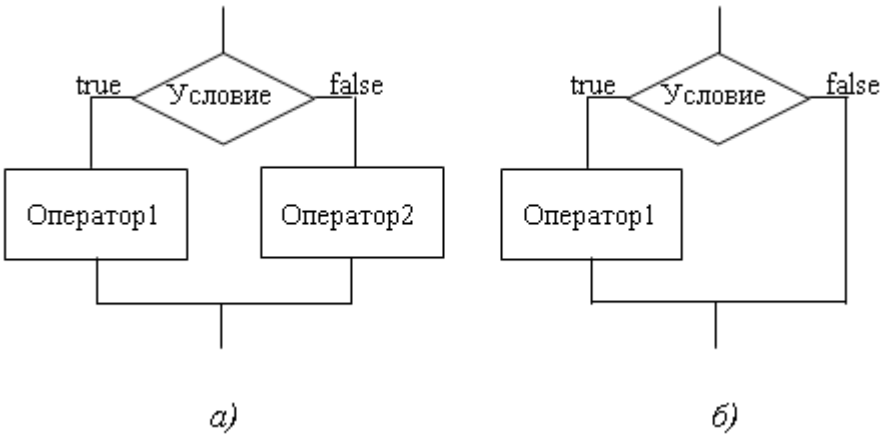


Рис. 3.2. Блок-схемы условного оператора:
а – полная форма, *б* – сокращенная форма

Оператор выбора (переключатель) имеет следующий вид:

switch ***(выражение)***

{ case ***константа1: оператор1*** **break;**

.....

case ***константаN: операторN*** **break;**

default: ***оператор*** **//**эта строка может отсутствовать

}

Здесь ***выражение*** - целочисленная переменная или соотношение; ***константа№*** – метка в виде константы или константного выражения; **default –**метка на оператор, который выполняется в том случае, если выражение не совпадает ни с одной константной меткой; **break** – оператор выхода из переключателя.

Блок – схема оператора выбора представлена на рис. 3.3.

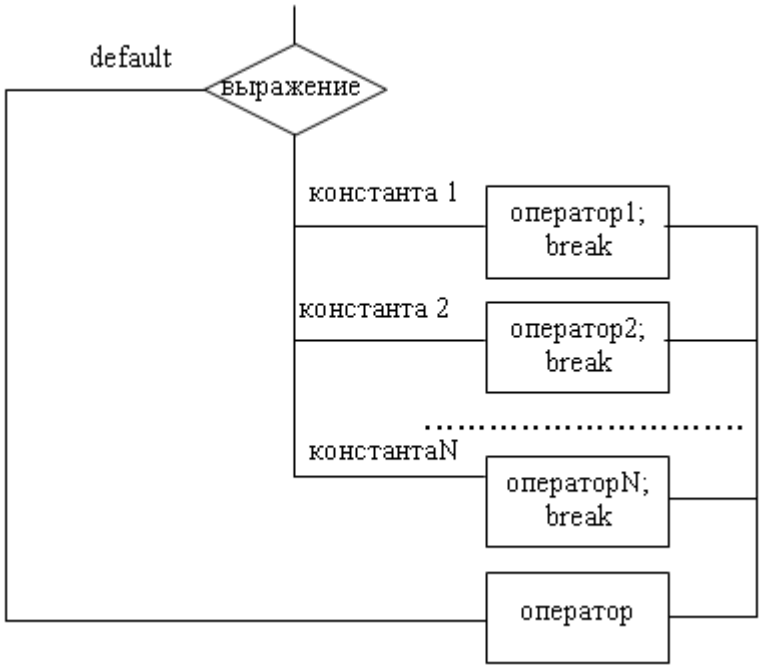


Рис. 3.3. Блок-схема оператора выбора

Пример:

```
switch  (rez)
        { case 5: cout <<"Отлично"; break;
  case 4: cout <<"Хорошо"; break;
  case 3: cout <<"Удовлетворительно"; break;
  case 2: cout <<"Неудовлетворительно"; break;
  default: cout <<"Неверное значение rez ";
}
```

Логический тип данных и базовые логические операции

При программировании алгоритмов с ветвлениями часто используется логический тип данных **bool**. Для величин этого типа существуют только два возможных значения: **true** (истина) и **false** (ложь). Логические значения обычно являются результатом операций сравнения (табл. 3.2).

Таблица 3.2

Операции сравнения

Знак операции	Действие
= =	Равно
! =	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

В языке C^{++} допускается преобразовывать в логические значения числа: ноль соответствует значению **false**, а любое отличное от нуля число преобразуется в значение **true.**

Для типа **bool** определены три стандартные логические операции и соответствующие им знаки операций: логическое умножение - **И (&&)**, логическое сложение - **ИЛИ (||)** и логическое отрицание - **НЕ (!)** (табл. 3.3).

Таблица 3.3

Логические операции

Переменные		Операции		
A	b	! a	a && b	a || b
False	false	true	false	false
False	true	true	false	true
True	false	false	false	true
True	true	false	true	true

При решении задач с множеством логических условий программа существенно упрощается, если эти условия удается объединить с помощью операций И, ИЛИ, НЕ в более крупные выражения.

Например, при определении принадлежности точки (x,y) области $[0 \le x \le x_{max},\ 0 \le y \le y_{max}]$ можно ввести одну интегрированную логическую переменную в следующем виде:

bool d =(x>=0) **&&** (x<=Xmax) **&&** (y>=0) **&&** (y<=Ymax);

В результате для программной реализации алгоритма достаточно одного условного оператора:

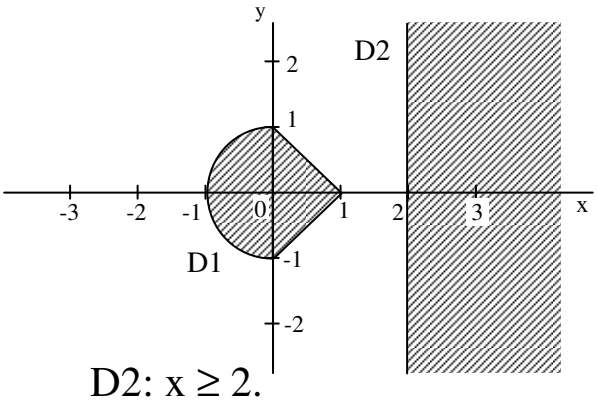
if (d) cout <<"*Точка принадлежит указанной области"*;

else cout << "*Точка не принадлежит указанной области"*;

Пример. Составить программу, которая для заданной точки **(x, y)** позволяет вычислять функцию **z** по выражению

$$z = \begin{cases} \dfrac{e^{x}\sqrt{\left|x^{3}+1\right|}}{\left(y^{2}+2\right)+\sin\left(y-x\right)}, & \text{при } (x, y) \in D1; \\ \dfrac{\sqrt{y^{2}+1}\left(2x^{3}+1\right)}{\operatorname{arctg}\left(y/x\right)+2}, & \text{при } (x, y) \in D2; \\ \dfrac{xe^{y}tg\ y}{y+\ln\left|x+1\right|}, & \text{при } (x, y) \notin D1 \cup D2, \end{cases}$$

где области **D1** и **D2** представлены графически и аналитически:



D2: x ≥ 2.

$$\text{D1: } \begin{cases} x^2 + y^2 \le 1, \\ y \le -x + 1, \\ y \ge x - 1. \end{cases}$$

Словесно-формульный алгоритм

1. Ввод координат точки (x, y).

2. Формирование интегрального условия принадлежности точки (x, y) области **D1** с помощью логической операции И:

d1 = $(x^2 + y^2 \le 1)$ И $(y \le -x+1)$ И $(y \ge x - 1)$.

3. Вычисление значения функции **z** в области **D1** с использованием промежуточных переменных:

если d1=true, то z1=|x^3+1|;

$z2 = e^x \sqrt{z1};$

z3=y^2+2;

z4=sin(y-x);

$$z = \frac{z2}{z3 + z4}.$$

4. Вычисление значения функции **z** в области **D2** (x ≥ 2) с использованием промежуточных переменных:

если x ≥ 2, то $z1 = \sqrt{y^2 + 1};$

z2=2x^3+1;

z3=arctg(y/x);

$$z = \frac{z1\, z2}{z3 + 2}.$$

5. Вычисление значений функции **z** вне областей **D1** и **D2** с использованием промежуточных переменных:

z1=x e^y tg y;

z2=ln|y+1|;

z3=y+z2;

z=z1/z3.

6. Вывод на печать значения **z** и координат исходной точки **(x,y).**

На основе словесно-формульного алгоритма можно построить блок-схему (рис. 3.4).

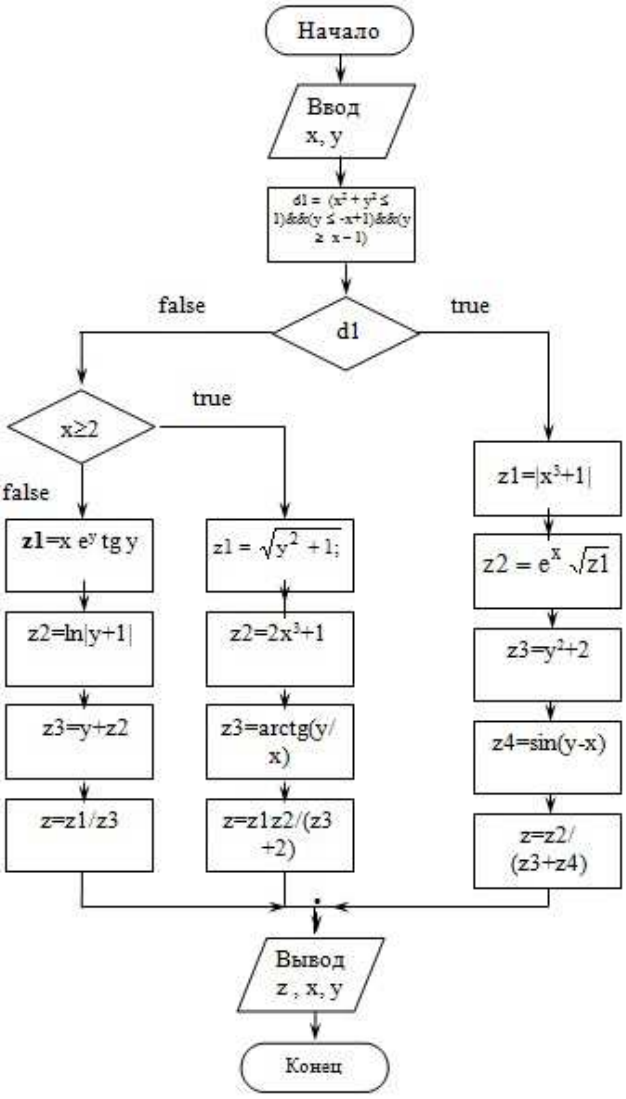


Рис. 3.4. Блок-схема алгоритма

Для решения данной задачи можно составить программу с использованием оператора ***if*** (лист. 3.1) и с использованием оператора ***swich*** (лист. 3.2).

Листинг 3.1. ***vetv_1.cpp***

```
#include <iostream.h>
#include <math.h>
void main()
{
  double x,y,z,z1,z2,z3,z4;
 cout<<"Введите координаты исходной точки"<<endl;
```

```
cout<<"x=";cin>>x;
cout<<"y=";cin>>y;
//Определение условий принадлежности области D1:
 bool d1=x*x+y*y<=1 && y<=-(x-1) && y>=x-1;
 if (d1)
    {                          //Вычисление z в области D1
     z1=fabs(pow(x,3)+1);
     z2=exp(x)*sqrt(z1);
     z3=pow(y,2)+2;
     z4=sin(y-x);
     z=z2/(z3+z4);
    }
 else if(x>=2)                 //Вложенный оператор if
       {                       //Вычисление z в области D2
        z1=sqrt(pow(y,2)+1);
        z2=2*pow(x,3)+1;
        z3=atan2(y,x);
        z=z1*z2/(z3+2);
       }
       else
       {                    //Вычисление z вне областей D1 и D2
        z1=x*exp(y)*tan(y);
        z2=log(fabs(y+1));
        z3=y+z2;
        z=z1/z3;
       }

 cout<<"\n Полученный результат:"<<endl;
 cout<<"z="<<z<<"  при x="<<x<<"  и y="<<y;
       }
```

Результат выполнения программы

```
Введите координаты исходной точки
x=1
y=1
 Полученный результат:
z=2.50036  при x=1  и y=1
```

!! *Проанализируйте программу. Создав новый файл проекта с именем vetv_1.ide, наберите в нем текст данной программы, откомпилируйте и произведите запуск программы на выполнение.*

Листинг 3.2. ***vetv_2.cpp***

```
#include <iostream.h>
#include <math.h>
void main()
{
  double x,y,z,z1,z2,z3,z4;
  short d;
 cout<<"Введите координаты исходной точки"<<endl;
 cout<<"x=";cin>>x;
 cout<<"y=";cin>>y;
 //Определение условий принадлежности областям D1 и D2:
  bool d1=x*x+y*y<=1 && y<=-(x-1) && y>=x-1;
  bool d2=x>=2;
    if (d1) d=1; if (d2) d=2;  //Формирование значений селектора
    switch(d)
      {
       case 1:
          {
        z1=fabs(pow(x,3)+1);
        z2=exp(x)*sqrt(z1);
        z3=pow(y,2)+2;
```

```
        z4=sin(y-x);
        z=z2/(z3+z4);    //Вычисление z в области D1
        break;
       }
      case 2:
       {
        z1=sqrt(pow(y,2)+1);
        z2=2*pow(x,3)+1;
        z3=atan2(y,x);
        z=z1*z2/(z3+2);       //Вычисление z в области D2
        break;
       }
      default:
       {
        z1=x*exp(y)*tan(y);
        z2=log(fabs(y+1));
        z3=y+z2;
        z=z1/z3;          //Вычисление z вне областей D1 и D2;
       }
     }
  cout<<"\n Полученный результат:"<<endl;
  cout<<"z="<<z<<"  при x="<<x<<"  и y="<<y;
 }
```

Результат выполнения программы

```
Введите координаты исходной точки
x=1
y=1
 Полученный результат:
z=2.50036  при x=1  и y=1
```

!! *Проанализируйте программу. Создав новый файл проекта с именем vetv_2.ide, наберите в нем текст данной программы, откомпилируйте и произведите запуск программы на выполнение.*

Упражнения

Составьте программу, которая для заданной точки **(x, y)** позволяет вычислять функцию **z** по выражению, приведенному в табл. 3.4, где области D1 и D2 представлены графически и аналитически в табл. 3.5.

Таблица 3.4

Варианты заданий

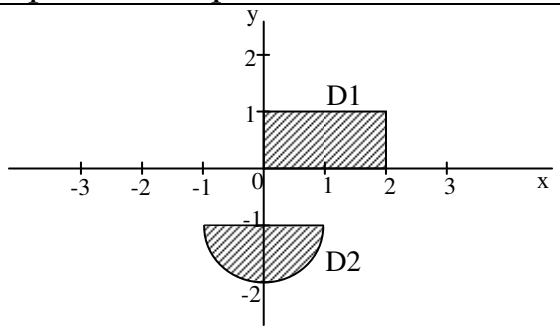
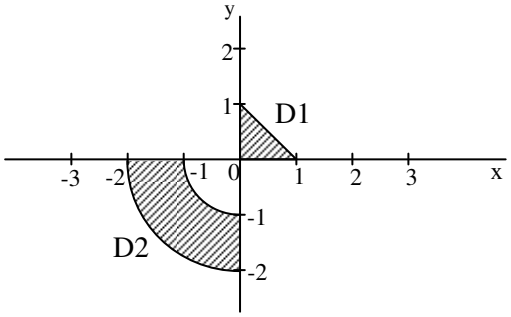
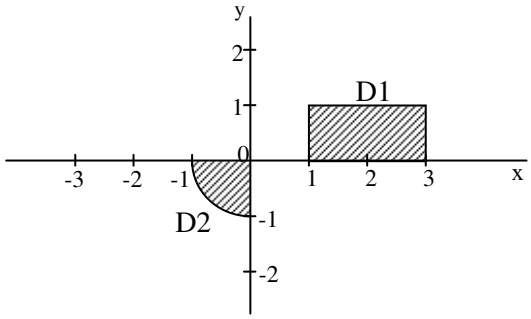
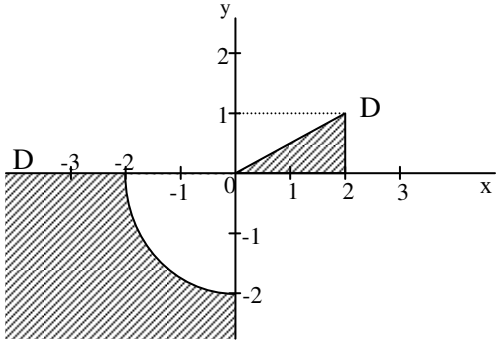
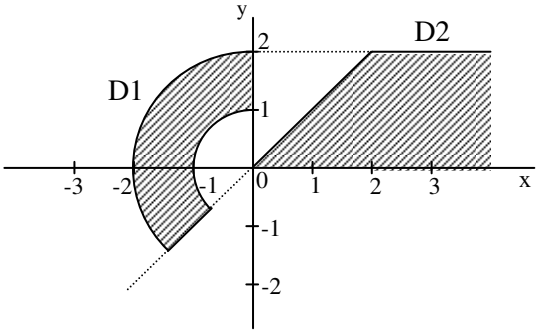
№ Вар	Функциональное выражение
1	$z=\begin{cases}\dfrac{3{,}5\lg\lvert x^3+3y\rvert}{e^x+2\sin y}, & \text{при } (x,y)\in D1,\\ \cos(x^2-1)+\sqrt{3{,}75+\lvert 2y^5-5\rvert}, & \text{при } (x,y)\in D2,\\ \operatorname{arctg}(y+1)+\dfrac{2^x}{y}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$
2	$z=\begin{cases}\dfrac{2{,}5e^{x+2}}{\sqrt{\lvert x^3+3y\rvert+1}}, & \text{при } (x,y)\in D1,\\ \dfrac{\ln\lvert x-\sqrt{y^2+3{,}4}\rvert}{x^2+y^2}, & \text{при } (x,y)\in D2,\\ \dfrac{2y^2+\sin x}{\sqrt{5x^2+1}}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$
3	$z=\begin{cases}\dfrac{1{,}2e^{3x}}{\operatorname{tg}\sqrt{1+y^2}}, & \text{при } (x,y)\in D1,\\ 2\lg\sqrt{x^4+1}+\cos^2(3y-0{,}75), & \text{при } (x,y)\in D2,\\ \dfrac{x^y+2xy}{\operatorname{tg}^2(x^4+e^y)}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$
4	$z=\begin{cases}\dfrac{y^{\frac{3}{2}}\sqrt{\lvert x+1\rvert}}{2y+\ln\lvert x-5{,}3\rvert}, & \text{при } (x,y)\in D1\\ -4{,}75y+\operatorname{arctg}\lg\lvert x^2+4\rvert, & \text{при } (x,y)\in D2\\ \dfrac{(e^x+\cos^2(y^2-4))^2}{\operatorname{ctg}(y^x+\sin y)}, & \text{при } (x,y)\notin D1\cup D2\end{cases}$

Продолжение табл. 3.4

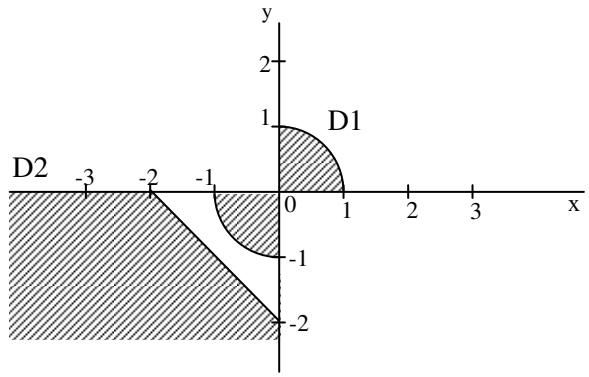
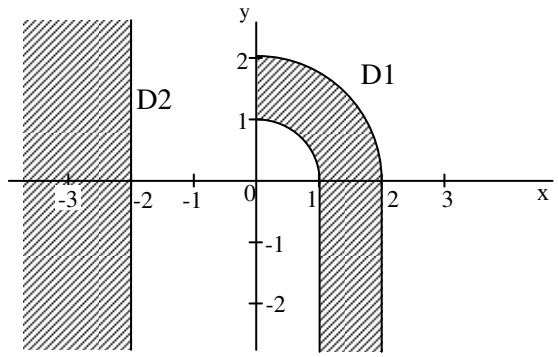
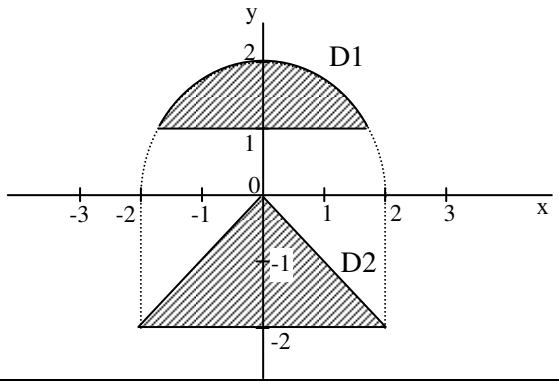
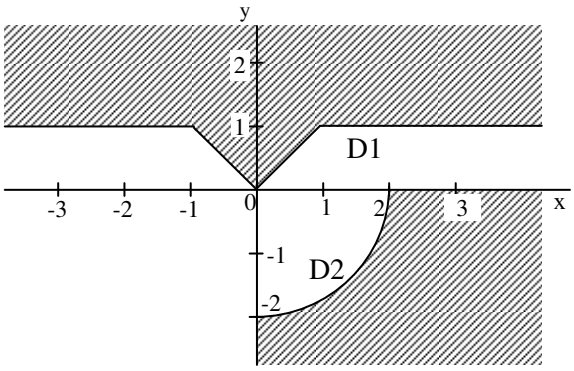
5	$z=\begin{cases}\dfrac{x^2\sin(y+2)}{\sqrt{\lvert x\rvert+e^y+1}}, & \text{при } (x,y)\in D1,\\ \dfrac{\operatorname{arctg}(x^2+y)-3{,}83}{\sin(1{,}75x+1)+\cos y}, & \text{при } (x,y)\in D2,\\ e^{x-y}+\cos(y^2-x^2)/e^{\lvert\sin y+3\rvert}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$	
6	$z=\begin{cases}\dfrac{x^3e^y}{\sqrt{\lvert y+\ln\lvert x\rvert+1\rvert}}, & \text{при } (x,y)\in D1,\\ xe^{-y}-\ln\lvert x^3+y\rvert, & \text{при } (x,y)\in D2,\\ \dfrac{x-e^{-(x+1)}}{\sin^2(x+y)+\cos^3 x}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$	
7	$z=\begin{cases}\dfrac{2{,}5x^3+1}{\sqrt{\lvert\ln\lvert x+1\rvert+5{,}3\rvert}}, & \text{при } (x,y)\in D1,\\ (e^{-0{,}5x}+e^{0{,}5y})/\sin(xy+0{,}25), & \text{при } (x,y)\in D2,\\ \dfrac{2^x-0{,}93\cos(y^2+3x)}{\operatorname{arcctg}(x+y)}, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$	
8	$z=\begin{cases}\dfrac{2\ln(x+1)}{\sqrt{y^4+2x^2+1}}, & \text{при } (x,y)\in D1,\\ \dfrac{x^3-1}{3y+x}+\lg\lvert\sin^3(x+1)+\cos y\rvert, & \text{при } (x,y)\in D2,\\ \sqrt{x^2+7{,}75}-\ln\lvert y^3-0{,}5x\rvert, & \text{при } (x,y)\notin D1\cup D2.\end{cases}$	
9	$z=\begin{cases}\dfrac{2e^{3y}+\ln\lvert x+2{,}5\rvert}{\sqrt{\sin x^5+1}}, & \text{при } (x,y)\in D1,\\ \dfrac{\sqrt[3]{x^2}+\operatorname{arctg}(y+\lg\lvert x\rvert)}{1{,}1+\cos^2(xy+10{,}3)}, & \text{при } (x,y)\in D2,\\ \ln\sqrt{\lvert x^2-y^2\rvert}+0{,}64^x/(\operatorname{tg} y+9{,}75), & \text{при } (x,y)\notin D1\cup D2.\end{cases}$	

Таблица 3.5

Области определения функции z (x,y)

№	Графическое представление	Аналитическое представление
1		D1: $0 \le x \le 2,\ 0 \le y \le 1;$ D2: $y \le -1;\ x^2 \le 1 - (y+1)^2.$
2		D1: $y \le -x + 1,\ x \ge 0,\ y \ge 0;$ D2: $1 \le x^2 + y^2 \le 4;\ x \le 0;\ y \le 0.$
3		D1: $1 \le x \le 3;\ 0 \le y \le 1;$ D2: $x^2 + y^2 \le 1,\ y \le 0,\ x \le 0.$
4		D1: $y \le \frac{1}{2}x,\ y \ge 0,\ x \le 2.$ D2: $x^2 + y^2 \ge 4,\ y \le 0,\ x \le 0.$
5		D1: $\begin{cases} 1 \le x^2 + y^2 \le 4, \\ x \le 0, \quad y \ge x. \end{cases}$ D2: $\begin{cases} y \le x, \\ 0 \le y \le 2. \end{cases}$

Продолжение табл. 3.5

6		D1: $x^2 + y^2 \le 1,\ xy \ge 0.$ D2: $y \le -x-2,\ y \le 0,\ x \le 0$
7		D1: $\begin{cases} 1 \le x^2 + y^2 \le 4,\ x \ge 0,\ y \ge 0, \\ 1 \le x \le 2,\ y \le 0. \end{cases}$ D2: $x \le -2$
8		D1: $x^2 + y^2 \le 4,\ y \ge 1;$ D2: $\begin{cases} y \le x,\ y \le -x, \\ y \ge -2. \end{cases}$
9		D1: $y \ge x;\ y \ge -x;\ y \ge 1.$ D2: $x^2 + y^2 \ge 4,\ x \ge 0,\ y \le 0.$

Контрольные вопросы

1. Поясните действие оператора выбора на примере.
2. Объясните взаимосвязь селектора и списка выбора в операторе Case.
3. Приведите пример применения оператора перехода.
4. Поясните логические операции or и and для двух переменных.
5. Приведите примеры записи условных операторов в полной и сокращенной формах на языке Turbo Pascal.
6. Какой тип данных получается в результате выполнения операций отношения?
7. Какие формы имеет условный оператор?
8. Приведите блок-схемы условного оператора.
9. Поясните структуру оператора **swich**.
10. В чем заключаются достоинства и недостатки оператора выбора?
11. Для чего в операторе выбора используется дополнительно оператор **break**?
12. Какие значения принимает логическая переменная?
13. Поясните правила выполнения логических операций И, ИЛИ, НЕ.
14. Какие операции сравнения Вы знаете и когда они используются?
15. Для чего в программах желательно использовать интегрированные логические переменные?
16. Сколько места занимает в памяти логическая переменная?
17. Расположите в порядке убывания приоритетов изученные Вами операции?
18. Как выполняются операции, имеющие одинаковый приоритет? Приведите примеры.
19. Почему в программах с разветвлениями не рекомендуется использовать оператор передачи управления **goto**?

4. СОСТАВЛЕНИЕ И ОТЛАДКА ПРОГРАММ С ЦИКЛАМИ

Цикл является одной из важнейших алгоритмических структур и представляет собой многократное выполнение последовательности операторов. В программах, связанных с однотипными вычислениями и обработкой массивов данных приходится часто выполнять повторяющиеся действия. С помощью циклов такие действия записываются в компактной форме. При этом один цикл может выполняться внутри другого, т.е. иметь вложенную структуру.

4.1. Организация программ с циклами на языке Turbo Pascal

Для программирования алгоритмов циклической структуры в Turbo Pascal (TP) применяются три вида операторов цикла (повторения).

1. Оператор цикла **For**, имеющий два варианта записи:

а) инкрементный (с возрастанием переменной цикла)

For i := n1 to n2 do s ;

б) декрементный (с убыванием переменной цикла)

For i := n1 downto n2 do s ;

где **i** - параметр цикла - переменная порядкового типа (integer и т.п.);

s - тело цикла (любой оператор, включая оператор цикла);

n1, **n2** - начальное и конечное значения переменной цикла; при этом для инкрементной формы **n1<n2** и шаг изменения счетчика цикла равен **+1**, а для декрементного варианта оператора **n1>n2** и шаг цикла равен **-1**.

Пример: Составить программу для вычисления произведения

$$P = \prod_{i=1}^{n} \frac{2x_i + i}{(2i)^2 + 1}$$

при n = 10, x_1 = 2,1 (начальное значение x), x_n = 3 (конечное значение x).

```
Program Example_1;
  Var P,x,h,x1,xn:real; n,i:integer;
   BEGIN
      writeln('Введите переменные:');
      write('n='); readln(n);
      write('x1='); readln(x1);
      write('xn='); readln(xn);
       h:=(xn-x1)/(n-1);          {вычисление шага изменения  x}
       P:=1; x:=x1;               {задание нач. значений переменным}
       for i:=1 to n do
         begin
           P:=P*(2*x+i)/(sqr(2*i)+1);
           x:=x+h;
         end;
       writeln('P=',P:12,' при x1=',x1:4:1,', xn=',xn:4:1);
   END.
```

2. Оператор цикла **While**:

While b do s ;

где **b** - выражение логического типа (условие); **s** - тело цикла.

В этом операторе сначала проверяется условие **b**. Если оно имеет значение **true**, то выполняется **s**, иначе управление передается оператору, следующему за оператором **while**.

3. Оператор цикла **Repeat**:

Repeat s Until b ;

где **b** - логическое условие; **s** - тело цикла (не требует заключения в операторные скобки **begin .. end**).

В отличие от **while** здесь сначала выполняется **s**, а затем проверяется условие **b**. Если оно примет значение **true**, то цикл завершается.

Блок-схемы выше рассмотренных операторов повторения представлены в табл. 4.1.

Для более гибкого управления циклическим вычислительным процессом в состав TP 7.0 включены две процедуры, которые практически исключают необходимость использования оператора **goto**:

Break - реализует немедленный выход из цикла; при этом управление передается оператору, следующему за циклом.

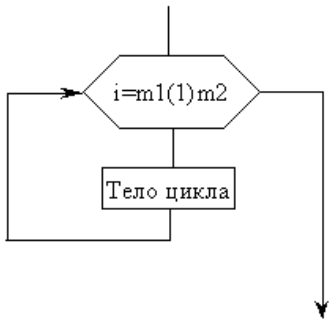
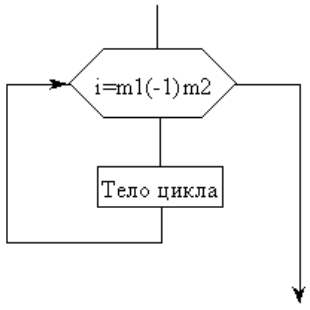
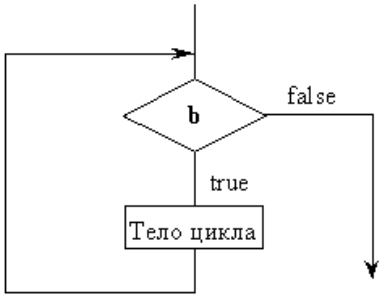
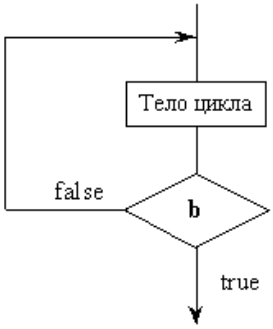
Continue - обеспечивает досрочное завершение очередного прохода цикла.

Они обычно включаются в тело цикла через дополнительно введенный условный оператор.

В операторах **While** и **Repeat** возможно зацикливание, если условие завершения цикла в операторе **While** постоянно сохраняет значение **true**, а в операторе **Repeat** - значение **false**. Для исключения этой ситуации необходимо предусмотреть дополнительную проверку на достижение заданного предельного числа повторений цикла с выходом из цикла, в частности с помощью оператора **break**.

Таблица 4.1

Блок-схемы операторов цикла

<table>
<tr><td>Инкрементный оператор цикла For
 
</td><td>Декрементный оператор цикла For
 
</td></tr>
<tr><td>Оператор While
 
</td><td>Оператор Repeat
 
</td></tr>
</table>

Использование операторов цикла во многих случаях обусловлено необходимостью применения в программах переменных с индексами (*массивов*). Их описание осуществляется двумя эквивалентными способами:

1) с определением типа массива

Type

<имя типа> = **array** [<список диапазонов индексов>]

of <тип элементов>;

Var

<имя массива> **:** <имя типа> ;

2) без определения типа массива

Var

<имя массива> **: array** [<список диапазонов индексов>]

of <тип элементов>;

Здесь <имя типа>, <имя массива> - идентификаторы;

<список диапазонов индексов> - в общем случае список выражений порядкового типа, кроме Longint, в частном случае - список диапазонов индексов;

<тип элементов> - тип элементов массива (real, integer и т.д.).

Первый способ обладает большей общностью. Однако чаще используется второй, так как он проще.

В практических задачах массивы обычно используются для представления векторов и матриц.

Примеры определения массивов:

```
Type m = array [1..5] of byte;
 Var  n : m;
      dates : array [1980..2000] of integer;
      x,y,z : array [1..10,1..10] of real;
```

Доступ к каждому элементу массива осуществляется с помощью *индекса*. Он не должен выходить за пределы указанного диапазона. Например,

```
for i:=1 to 10 do
  for j:=1 to 10 do
```

```
begin  z[i,j]:=0;
     for k:=1 to 10 do  z[i,j]:=z[i,j]+x[i,k]*y[k,j]
end;
```

В ТР одним оператором присваивания все элементы одного массива можно передать другому массиву того же типа, например:

```
Var p,q : array [-5..5] of single;
    begin … p : = q; … end.
```

Однако необходимо учитывать, что массивы можно сравнивать только поэлементно.

Пример. Составить таблицу значений функции:

$$z = f(x, y) = \frac{e^{xy}}{xy} + \frac{yx^4}{4},$$

где x и y заданы в виде одномерных массивов:

x = 0.1, 0.3, 0.5; y = 1, 2, 3, 4.

Текст программы

```
Program Tab_1;
  Const m=3;n=4;
  Var x,y:array[1..10]of real;
    f:array[1..10,1..10]of real;
    i,j:integer; a,b:real;
   BEGIN
     writeln('Введите элементы массивов x и y:');
     for i:=1 to m do    {Ввод элементов массива x}
       begin
         write('x[',i,']='); readln(x[i]);
       end;
     for j:=1 to n do     {Ввод элементов массива y}
       begin
         write('y[',j,']='); readln(y[j]);
       end;
```

```
    for i:=1 to m do
    for j:=1 to n do
      begin
        a:=exp(x[i]*y[j])/(x[i]*y[j]);
        b:=y[j]*exp(4*ln(x[i]))/4;
        f[i,j]:=a+b;
        writeln('f=',f[i,j]:10,' при x=', x[i]:4:1,' и y=',y[j]:4:1);
      end;
    END.
```

!! *Проанализируйте программу, наберите ее текст и осуществите компиляцию. Сохраните программу в файле с названием* ***Tab_1.pas***.

При задании диапазонов индексов в объявлении массивов можно использовать интервальный (диапазонный) тип данных.

Объявление интервального типа осуществляется заданием двух констант, указывающих верхнюю и нижнюю границы диапазона, и описывается двумя способами:

с определением типа

Type

<имя интервального типа>= <k1> . . <k2>;

Var

<имя переменной>:<имя интервального типа>;

без определения типа

Var

<имя переменной>: <k1> . . <k2>;

где k1 и k2 соответственно верхняя и нижняя границы диапазона, причем k1>k2.

Например,

```
  Type        index = 1 . . 50;
            years = 1900 . . 2010;
        Var   birthday: years;
              mass : array [index, index] of real;
```

При отладке программ с массивами удобно использовать типизированные константы-массивы в качестве исходных данных, что избавляет пользователя от многократного ввода элементов массива в циклах.

В частности, при объявлении типизированной константы-вектора в круглых скобках указываются значения элементов массива через запятую. При объявлении двумерных констант-массивов множество элементов, соответствующее каждой строке матрицы, заключается в дополнительные круглые скобки.

Например:

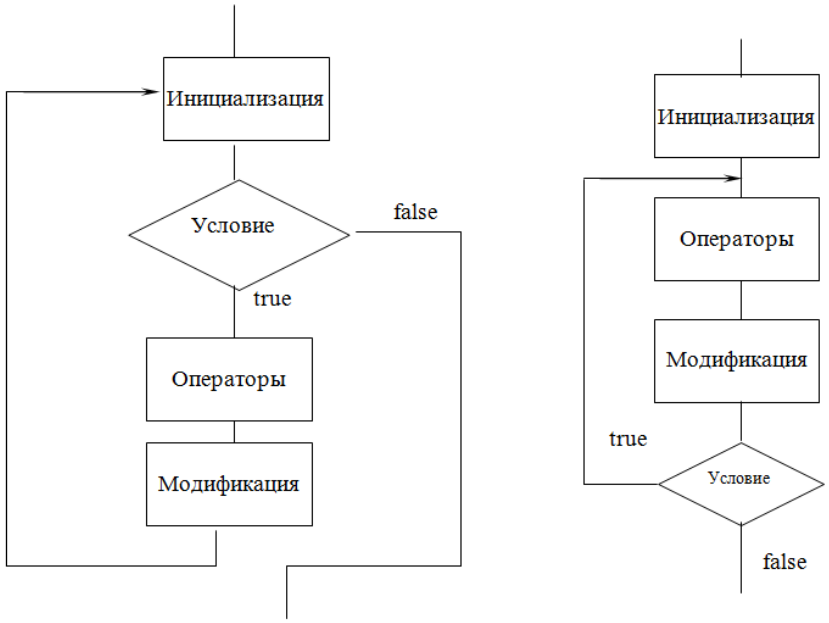
```
const
  vector : array [0..3] of integer = (1,2,3,4);
  matrix : array [0..2,0..3] of real =
           ((1,2,3,4),(1.1,1.2,1.3,1.4),(2.3,2.4,2.5,7));
```

4.2. Организация программ с циклами на языке C++

Любой цикл в общем случае состоит из тела цикла – одного или нескольких операторов, выполняющихся некоторое количество раз, начальных установок, проверки условия выхода из цикла и модификации параметра цикла - переменной, изменяющейся в цикле и используемой в формировании условия.

В зависимости от местоположения условной операции выделяют две основные схемы операторов цикла: цикл с предусловием (рис. 4.1а) и цикл с постусловием (рис. 4.1б).

Первая схема в C^{++} реализуется операторами **while** и **for,** а вторая **do**.



а) *б)*

Рис. 4.1. Блок-схемы циклического алгоритма:
а - цикл с предусловием; *б* – цикл с постусловием

Оператор цикла **while** имеет вид

while ***(условие) тело_цикла***,

где ***тело_цикла*** – блок операторов циклического вычислительного процесса вместе с оператором модификации параметра цикла.

Пример. Сумма **n** первых чисел натурального ряда.

```
. . . . . .
int n;  cin>>n;
int s=0, i=1;
  while (i<=n)
  {s=s+i; i=i+1;}
   cout<<s;
. . . . . .
```

Оператор **for** удобнее, чем **while,** так как его заголовок более информативен (содержит все основные параметры цикла). Он имеет следующую форму:

for ***(инициализация; условие; модификация) операторы***

Любая из частей оператора **for** может быть опущена, но точки с запятой необходимо оставить на своих местах (пустые операторы), причем если отсутствует ***условие***, то результат проверки всегда истинен.

Пример. Вычисление факториала **n!.**

```
. . . . . .
int n,r=1;  cin>>n;
for (int i=1; i<=n; i=i+1)
r=r*i; cout<<"n!="<<r;
. . . . . .
```

Из-за частого применения оператора **for** для его заголовка было введено специальное графическое обозначение – модификация (см. табл. 4.1). Примером использования модификатора служит блок-схема вычисления факториала n!, приведенная на рис. 4.2.

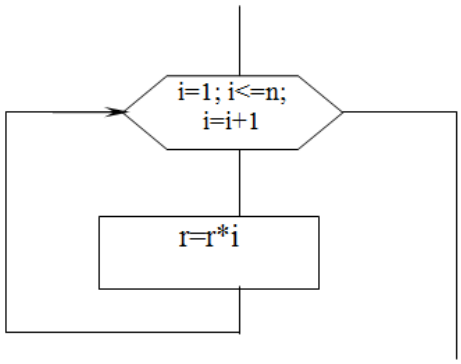


Рис. 4.2. Блок-схема вычисления факториала

Оператор **do** является циклом с постусловием (см. рис. 4.1б) и используется в тех случаях, когда условие формировать в начале итерационного процесса не совсем удобно, а также когда цикл необходимо выполнить хотя бы один раз. Этот цикл имеет следующую форму записи:

do ***тело_цикла*** **while** ***(условие)***

Пример. Вычислить $\sqrt{x}$ с заданной точностью **eps** по итерационной формуле $y_i=(y_{i-1}+x/y_{i-1})/2$ при начальном приближении $y_0=1$.

```
 . . . . . .
double x, eps, y0, y=1;
cin>>x>>eps;
do { y0=y; y=(y0+x/y0)/2;
    } while (fabs (y-y0)>=eps);
cout<<y;
 . . . . . .
```

В практических задачах часто используются ***вложенные циклы***, т.е. когда один цикл выполняется внутри другого. Во вложенных циклах в соответствии с логикой построения программы можно использовать любые комбинации операторов **while, for, do**.

В циклических структурах возможно ***зацикливание*** (бесконечный цикл) из-за вычислительной неустойчивости алгоритма или его неправильного программирования. Для исключения этой ситуации в операторах цикла на этапе отладки программы необходимо предусмотреть дополнительную проверку на сходимость или достижение заданного предельного числа повторений цикла. Существует два оператора выхода из цикла: **goto** и **break**.

Оператор **goto** идет вразрез с правильным стилем программирования и его в настоящее время использовать не рекомендуется, кроме исключительных случаев, когда без него сложно обойтись.

Обычно выполнение цикла прерывают с помощью оператора **break**. В результате управление передается оператору, следующему непосредственно за циклом.

Инкремент, декремент и составные операции присваивания

Для сокращения записи операторов итерационного вида в C^{++} предусмотрен ряд специальных операций. К ним относятся инкремент, декремент и составные операции присваивания. Их часто при-

меняют в циклах.

Инкремент (++) – автоувеличение переменной на единицу. Он имеет две формы: префиксную и постпрефиксную.

Префиксный инкремент (++x) – увеличение значения операнда (x) на единицу до его использования.

Например:

int x=1; int y=++x; // в результате y=2.

Постпрефиксный инкремент (x++) - увеличение значения операнда (x) на единицу после его использования.

Например:

int x=1; int z=x++; // в результате z=1.

Декремент (--) – автоматическое уменьшение переменной на единицу. Он так же, как и инкремент, имеет две формы: префиксную и постпрефиксную.

Составные операции присваивания содержат два разных знака операций, один из которых (второй слева) является простым оператором присваивания (=). Наиболее часто используемые арифметические составные операции и их эквиваленты приведены в табл. 4.2.

В C^{++} можно использовать ***множественное присваивание*** для однотипных переменных. Оно часто применяется для сокращения записи начальных установок.

Например: z=y=x=0.

Таблица 4.2

Составные операции присваивания

Знак операции	Вид операции	Пример	Эквивалент
*** =**	Умножение с присваиванием	x*=2	x=x*2
/ =	Деление с присваиванием	x/=5-y	x=x/(5-y)
+ =	Сложение с присваиванием	x+=y	x=x+y
- =	Вычитание с присваиванием	x-=y+0.5	x=x-(y+0.5)

Пример. Вычислить значение гиперболического синуса **shx** с заданной точностью **eps** с помощью разложения в бесконечный ряд:

$$shx = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + ..., \quad |x| < \infty.$$

Вычисление заканчивается, когда абсолютная величина очередного члена ряда, прибавляемого к сумме, станет меньше **eps**.

```
. . . . . .
double y,x,s;
cin>>x;
y=s=x;                          //множественное присваивание
for (int i=0; fabs(y)>eps; i++)
{ y*=pow(x,2)/(2*i+2)/(2*i+3);  //очередной член ряда
  s+=y;
  if (i>500)                    //проверка на зацикливание
     { cout<<"\n Ряд расходится!";
       break;
      }
 }
cout<<"\n Значение s="<<s;
    . . . . . .
```

Пример 1. Найти значение степенного ряда

$$y = \ln(1+x) = x - \frac{x^2}{2} + ... + (-1)^{n+1}\frac{x^n}{n}..., \quad при \quad |x| < 1$$

Вычисление осуществлять до выполнения условия $\left|\frac{x^n}{n}\right| < \varepsilon = 0{,}0001.$

В основе алгоритма лежит рекуррентная формула расчета следующего члена ряда по предыдущему: $z = -z \cdot x\frac{i}{i+1}$.

Для решения данной задачи можно использовать циклы, реализуемые операторами while и do (лист. 4.1).

Листинг 4.1. ***st_ryd.cpp***

```
#include <iostream.h>
#include <math.h>
void main()
{
 float x,y,z,eps;
 cout<<"Задайте точность вычислений eps=";cin>>eps;
 cout<<"Введите переменную x=";cin>>x;
 while (fabs(x)>=1)              // Цикл проверки правильности ввода
  {
    cout<<"Введите переменную x в диапазоне |x|<1";
    cout<<"\nx=";cin>>x;
  }
  const int MaxIter=100;       //Ограничитель количества итераций
 y=z=x;                        //Множественное присваивание
 int i=1;                      //Задание параметра цикла
 do
        {
   z*=-x*i/(i+1);           //Определение члена ряда
   y+=z;                    //Текущая сумма ряда
   i++;                     //Модификация параметра цикла
   if(i>MaxIter)            //Проверка зацикливания
       {cout<<"\nРяд расходится!";
     break;
     }
  }
  while (fabs(z)>=eps);          //Условие выхода из цикла
   cout<<"y="<<y<<" при x="<<x<<" eps="<<eps;
}
```

Результат выполнения программы

```
Задайте точность вычислений eps=0.00001
Введите переменную x=0.9
y=0.64185 при x=0.9 eps=1e-05
```

!! *Проанализируйте программу. Создав новый файл проекта с именем st_ryd.ide, наберите в нем текст данной программы, откомпилируйте и произведите запуск программы на выполнение.*

Пример 2. Вычислить значение функции $s = k\sum_{k=1}^{n}\sum_{m=0}^{k}\frac{k}{1+a_m^{m+k}}$, при n =8, где a_0=1, h_a=1.2 – соответственно начальное значение и шаг изменения переменной a.

Для решения данной задачи следует использовать циклы, реализуемые операторами while и for (лист. 4.2).

Листинг 4.2. ***summa.cpp***

```
#include <iostream.h>
#include <math.h>
void main()
{
 float a,a0,h; int n;
 cout<<"Введите значение n"<<endl;
 cout <<"n=";cin>>n;
 cout<<"Введите начальное значение a0"<<endl;
 cout<<"a0=";cin>>a0;
 cout<<"Введите шаг h"<<endl;
 cout <<"h=";cin>>h;
 int k=1; double s=0;       //Инициализация параметров цикла while
 while (k<=n)
  {
```

```
        a=a0;          //Задание начального значения a для цикла for
     for (int m=0;m<=k;m++)
     {
      s+=k/(1+pow(a,m+k));      //Накопление суммы
      a+=h;                      //Изменение параметра a на h
     k++;                        //Постпрефиксный инкремент

   }
   cout<<"Сумма ряда s="<<s
      <<"\n при a0="<<a0<<" h="<<h<<" n="<<n;
 }
```

Результат выполнения программы

```
Введите значение n
n=8
Введите начальное значение a0
a0=1
Введите шаг h
h=1.2
Сумма ряда s=18.6548
 при a0=1 h=1.2 n=8
```

!! *Проанализируйте программу. Создав новый файл проекта с именем summa.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Упражнения

Составить и отладить программу решения задачи согласно приведенным в табл. 4.3 вариантам заданий.

Таблица 4.3

Варианты заданий

	Вычислить значение интеграла $\int_a^b f(x)dx$ по формуле трапеций: $J \approx h\left[\frac{f(x_0)+f(x_n)}{2}+\sum_{i=1}^{n-1} f(x_i)\right]$, где $h=\frac{b-a}{n}$, $x_i = x_0 + ih$, $x_0 = a$, $x_n = b$, $f(x)=\frac{x^3}{x^4+1}$, при a = 3, b = 8, n = 40
	Найти значение степенного ряда $f(x)=\sum_{i=0}^{\infty}\frac{x^i}{i!}=1+\frac{x}{1!}+\frac{x^2}{2!}+\ldots$, при x = 2.2. Вычисление осуществлять до выполнения условия $\left\|\frac{x^i}{i!}\right\| < 0.00001$
	Вычислить $S=\sum_{i=1}^{n}\prod_{j=1}^{i}\sum_{k=1}^{j}\sin\left(ix^2+jx+k\right)$, используя три разных цикла, при x=9,81, n=11.
	Составить таблицу значений функции $z=f(x,y)=\frac{2{,}75\cos\left(x^2+1\right)}{e^x\sqrt{\|y-5\|}}$ с использованием операторов цикла while и do, при x = 1..5 , h_x= 1 , y = 2,1..3 , h_y= 0,2
	Составить таблицу значений функции $z=f(x,y)=\frac{3x+x\ln\sqrt{y}}{1+e^{2x}}$ с использованием операторов цикла for и do, при x = 0..5 , h_x= 0,7 , y = 4..7 , h_y= 0,5
	Составить таблицу значений функции $z=f(x,y)=\frac{5x^3+0{,}8y^2}{3\ln\left\|e^x-e^y\right\|}$ с использованием операторов цикла for и while, при x = -1..2 , h_x= 0,4 , y = 2..5 , h_y= 0,9

Контрольные вопросы

1. Чем отличается инкрементный оператор цикла **For** от его декрементного варианта?
2. Поясните работу двух вложенных операторов цикла **For**.
3. Поясните особенности работы операторов цикла **While** и **Repeat**.
4. Каким образом исключается зацикливание в операторах **While** и **Repeat**?
5. Какими способами осуществляется описание массивов в ТР?
6. Покажите на примерах, как осуществляется доступ к элементам массива и как сравниваются два однотипных массива.
7. В чем заключается отличие индекса массива от порядкового номера элемента массива?
8. Поясните сущность процесса зацикливания и с помощью какой процедуры можно выйти при этом из цикла?.
9. Охарактеризуйте операторы **while** и **do**…**while**?
10. Зарисуйте блок-схему оператора **for** и поясните правила записи выражений в его заголовке.
11. Что подразумевается под операциями инициализация и модификация?
12. Поясните по блок-схемам принципы действия операторов с предусловием и постусловием.
13. В чем заключается сущность префиксных и постпрефиксных операций инкремента и декремента? Приведите примеры.
14. Какие составные операции присваивания Вы знаете? Приведите примеры.
15. В каких случаях можно применять операцию множественного присваивания?
16. Как преобразовать цикл **for** в цикл **while** и наоборот?

4.3. Составление программ с массивами на языке С++

Массив представляет собой конечный упорядоченный набор однотипных объектов, имеющих общее имя. Элементы массива занимают один непрерывный участок памяти компьютера и располагаются последовательно друг за другом.

Массив состоит из **элементов**, являющихся данными одного типа (возможно структурированного), имеет **размерность** – количество элементов, которое содержит массив. Элементы массива имеют **индексы** – порядковые номера, которые приписываются элементам для того, чтобы отличать их друг от друга.

Описание одномерного массива имеет следующий формат:

Тип *имя_массива* [n];

где n – ***размерность*** (количество элементов) массива является в общем случае константным выражением.

Такие ***одномерные*** массивы часто называют ***векторами.***

Например:

```
int days [12];          //массив days из 12 целых чисел
double mass [50];       //массив mass из 50 вещественных чисел
```

Для ***доступа*** к элементу массива используется имя массива и его индекс в диапазоне от ***0*** до ***n-1***, заключенный в квадратные скобки:

имя_массива [индекс]

Например:

```
mass [0] = 37.5; . . . mass [49] = 36.6;
```

Массивы можно инициализировать списком значений или выражений, отделенных запятой в фигурных скобках.

Например:

```
int days [12] = {27,15,11,5,28,9,14,31,21,15,19,27};
```

Если список значений короче длины массива, то инициализации подвергаются первые элементы массива, а остальным присваивается нулевое значение.

Например:

int a[5] = {3,12,4}; //a[0]=3, a[1]=12, a[2]=4, a[3]=0, a[4]=0

Массивы могут инициализироваться без указания индекса. В этом случае размерность определяется числом элементов в фигурных скобках.

Например:

int numbers [ ] = {1,2,3,4}; //размерность равна четырем

В практических задачах часто используются ***двумерные массивы***, т.е. массивы с двумя индексами. Они описываются следующим образом:

Тип *имя_массива* [*размер_1*] [*размер_2*]

С помощью таких массивов обычно отображаются ***матрицы.***

Например:

const int k=5, r=3;

long m[k][k]; //матрица целых чисел размерности 5x5

float d[k][r]; //матрица целых чисел размерности 5x3

Инициализация двумерного массива осуществляется двумя способами:

1) представлением массива из массивов с заключением каждого массива-строки в свои фигурные скобки (в этом случае размерности можно не указывать);

2) заданием общего списка элементов в том порядке, в котором они располагаются в памяти.

Например:

int matr [ ][ ] = {{3,-4},{25,0},{-1,4}};

int matr [3][2] = {3,-4,25,0,-1,4};

Как и в одномерных массивах при инициализации двухмерных массивов можно присваивать значения не всем элементам.

Пример. Формирование треугольной матрицы.

int x [5][4] = {{11},{25,4},{-3,18,14},{2,17,9,15}};

что эквивалентно отображению матрицы:

$$X = \begin{vmatrix} 11 & 0 & 0 & 0 \\ 25 & 4 & 0 & 0 \\ -3 & 18 & 14 & 0 \\ 2 & 17 & 9 & 15 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

При работе с массивами широко используются операторы цикла, особенно, **for**. Примером может служить задача ***сортировки*** массива с помощью "пузырькового" метода.

```
 #include <iostream.h>
#include <math.h>
void main()
{
 const int m=5;                          //Размерность массива чисел
 double z[m];                             //Тип чисел массива z
 for(int i=0;i<m;i++)                    //Ввод элементов массива
 { cout<<"z["<<i<<"]=";cin>>z[i];}
 for(int i=0;i<m-1;i++)          //Начало алгоритма сортировки
 { double a;
  for(int j=i+1;j<m;j++)
    if (z[j]<z[i])                     //'<'- сортировка по возрастанию
    {a=z[i];z[i]=z[j];z[j]=a;}
 }                                          //Конец алгоритма сортировки
 cout<<"z["<<m<<"]={";
 for(int i=0;i<m;i++)
 cout<<z[i]<<"  ";cout<<"}";  //Вывод элементов массива
}
```

Манипуляторы и форматирование ввода-вывода

В программах часто требуется осуществлять ввод и вывод данных в определенном формате. Для этого обычно используются так называемые ***манипуляторы,*** основные из которых приведены в табл.

4.4. Некоторые из них имеют аргумент (тип). Для использования последних к программе требуется подключить заголовочный файл ***<iomanip.h>.***

С помощью манипуляторов легко управлять представлением выводимой информации. В частности манипулятор ***setw (int n)*** удобен при формировании таблиц в циклах.

Например:

```
. . . . . .
cout<<setprecision (2);          //две цифры после запятой
for (int i=0; i<10; i++)
cout<<setw (6) <<a[i]<<endl;   //ширина поля – 6 цифр
. . . . . .
```

Таблица 4.4

Манипуляторы

Манипуляторы	Действия
1	2
endl	Переход на новую строку при выводе

Окончание табл. 4.4

1	2
dec	Вывод чисел в десятичной системе (действует по умолчанию)
hex	Вывод чисел в шестнадцатеричной системе
oct	Вывод чисел в восьмеричной системе
setw (int n)	Устанавливает минимальную ширину поля в n символов
setprecision (int n)	Устанавливает количество цифр после запятой при выводе вещественных чисел
setfill (int n)	Устанавливает символ-заполнитель с кодом n; этим символом выводимое значение будет заполняться до необходимой ширины
setbase (int n)	Устанавливает систему счисления n (2, 8, 10,16)

Пример 1. Получить вещественную матрицу A(7,7), первая строка которой задается формулой a_{1j}=2,13j+3,7, (j=1,..,7), вторая строка задается формулой $a_{2j} = j - \frac{29}{2 + \frac{1,3}{j}}$, (j=1,..,7), а каждая следующая строка есть сумма двух предыдущих. Определить в четных столбцах матрицы A число элементов, не принадлежащих интервалу (5, 10), а в седьмой строке определить произведение отрицательных элементов в нечетных столбцах матрицы A.

Программа для решения данной задачи приведена в лист. 4.2.

Листинг 4.3. ***mass_1.cpp***

```
#include <iostream.h>
#include <math.h>
//Включение заголовочного файла, подключающего /манипуляторы
#include <iomanip.h>
void main()
{
 double a[8][8];
  //Получение матрицы a
  for(int i=1;i<8;i++)
  for(int j=1;j<8;j++)
  {
    if(i==1) a[i][j]=2.13*j+3.7;
    if(i==2) a[i][j]=j-29/(2+1.3/j);
    if(i>2) a[i][j]=a[i-1][j]+a[i-2][j];
  }
  //Определение числа элементов в четных столбцах матрицы a,
  // не принадлежащих интервалу (5,10)
  int k=0;
  for(int i=1;i<8;i++)
  for(int j=1;j<8;j++)
  if ((j%2==0)&&((a[i][j]<5)||(a[i][j]>10))) k++;
```

```
        //Определение произведения отрицательных
        //элементов нечетных столбцов в 7 строке матрицы a
        double p=1;
        for(int i=7,j=1;j<8;j++)
              if((a[i][j]<0)&&(j%2==1)) p*=a[i][j];
          //Вывод результатов вычислений
          //Вывод матрицы a
        cout<<"Матрица a:"<<"\n";
        cout<<setfill('.')            //Введение символов '.' в поле вывода
           <<setprecision(4);      //Установка точности вывода
         for(int i=1;i<8;i++)
         for(int j=1;j<8;j++)
         {cout<<"  "<<setw(8)<<a[i][j];       //Фоматирование вывода
         if(j==7)cout<<"\n";                   //Формирование строки
         }
         cout<<"\nЧисло элементов в четных столбцах матрицы a,\n";
         cout<< "не принадлежащих интервалу (5,10):"<<endl;
         cout<<"k="<<k<<"\n";
         cout<<"\nПроизведение отрицательных элементов в 7 строке\n";
         cout<<"нечетных столбцов матрицы a:"<<endl;
         cout<<"p="<<p;
      }
```

Результат выполнения программы

```
Матрица a:
 ....5.83    ....7.96   ...10.09  ...12.22  ...14.35  ...16.48  ...18.61
 ..-7.788  ..-8.943  ..-8.918  ..-8.473  ..-7.832  ..-7.083  ..-6.268
 ..-1.958  .-0.9834  ...1.172  ...3.747  ...6.518  ...9.397  ...12.34
 ..-9.746  ..-9.927  ..-7.746  ..-4.726  ..-1.314  ...2.315  ...6.074
 ...-11.7  ..-10.91  ..-6.573  .-0.9794  ...5.204  ...11.71  ...18.42
 ..-21.45  ..-20.84  ..-14.32  ..-5.706  ...3.891  ...14.03  ...24.49
 ..-33.15  ..-31.75  ..-20.89  ..-6.685  ...9.095  ...25.74  ...42.91
```

```
Число элементов в четных столбцах матрицы a,
не принадлежащих интервалу (5,10):
k=19
Произведение отрицательных элементов в 7 строке
нечетных столбцов матрицы a:
p=692.6
```

!! *Проанализируйте программу. Создав новый файл проекта с именем mass_1.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Пример 2. Дана целочисленная матрица A(4,3). Получить матрицу B(4,3) по правилу $b_{ij} = \begin{cases} 0 & при \quad a_{ij} < 0 \\ 1 & при \quad a_{ij} \geq 0 \end{cases}$, (i=1,…,4; j=1,…,3).

В матрице A найти произведение элементов, стоящих ниже первой строки, а во втором столбце матрицы B определить количество единичных элементов.

Программа для решения данной задачи приведена в лист. 4.4.

Листинг 4.4. ***mass_2.cpp***

```
#include <iostream.h>
#include <math.h>
//Подключение заголовочного файла, включающего
//модификатор setw (установка ширины вывода)
#include <iomanip.h>
void main()
{
 long b[4][3],a[4][3]={{ 1,-2 , 3 },      //Инициализация матрицы a
                       {4 , 5 , -6},
                      {-7, -8 , 9 },
                       {10,11,12}};
  //Получение матрицы b
```

```
    for(int i=0;i<4;i++)
    for(int j=0;j<3;j++)
     b[i][j]=(a[i][j]<0)? 0:1;
    //Вывод матриц a и b
    cout<<"Матрица a:"<<"\n";
     for(int i=0;i<4;i++)
     for(int j=0;j<3;j++)
     {cout<<"  "<<setw(4)<<a[i][j];          //Фоматирование вывода
     if(j==2)cout<<"\n";                     //Формирование строки
     }
     cout<<"Матрица b:"<<"\n";
     for(int i=0;i<4;i++)
     for(int j=0;j<3;j++)
     {cout<<"  "<<setw(4)<<b[i][j];
      if(j==2)cout<<"\n";
     }
    //Определение произведения элементов матрицы a,
    //стоящих ниже первой строки
     long p=1;
     for(int i=1;i<4;i++)
     for(int j=0;j<3;j++)
         p*=a[i][j];
     cout<<"p="<<p<<endl;
    //Определение количества единичных элементов
    //во втором столбце матрицы b
    int s=0;
    for(int i=0,j=1;i<4;i++)
    if (b[i][j]==1) s++;
    cout<<"s="<<s<<endl;
  }
```

Результат выполнения программы

```
Матрица a:
   1   -2    3
   4    5   -6
  -7   -8    9
  10   11   12
Матрица b:
   1    0    1
   1    1    0
   0    0    1
   1    1    1
p=-79833600
s=2
```

!! *Проанализируйте программу. Создав новый файл проекта с именем mass_2.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Упражнения

Составить и отладить программу преобразования двумерных массивов согласно приведенным в табл. 5.2 вариантам заданий.

Таблица 4.5

Варианты заданий

	Дана вещественная матрица A(3,3). Получить матрицу B(3,3), каждый элемент которой вычисляется по формуле $b_{ij} = \begin{cases} a_{ij} & при \quad a_{ij} \geq 0 \\ -1 & при \quad a_{ij} < 0 \end{cases}$ Сформировать матрицу C(3,3), являющуюся суммой матриц A и B: $c_{ij} = a_{ij} + b_{ij}$; (i=1,…,3; j=1,…,3)

Продолжение табл. 4.5

	Вещественную матрицу A(3,4) преобразовать в матрицу B(3,4) по правилу $b_{ij}=\begin{cases} a_{ij} & при \quad a_{ij}=0 \\ 2{,}1\,a_{ij} & при \quad a_{ij}\neq 0 \end{cases}$ (i=1,…,3; j=1,…,4). Найти количество нулевых элементов матрицы A и сумму отрицательных элементов второго столбца матрицы B
	Даны целочисленные матрицы B(4,4) и C(4,4). Построить матрицу A(4,4), каждый элемент которой вычисляется по формуле $a_{ij}=\begin{cases} b_{ij} & при \quad \lvert b_{ij}\rvert > \lvert c_{ij}\rvert \\ c_{ij} & при \quad \lvert b_{ij}\rvert \leq \lvert c_{ij}\rvert \end{cases}$, (i=1,…,4; j=1,…,4). В матрице A найти произведение элементов, стоящих выше главной диагонали, а в матрице B – максимальный элемент в третьей строке
	Дана целочисленная матрица A(4,6). Получить матрицу B(4,6), каждый элемент которой вычисляется по формуле $b_{ij}=\begin{cases} a_{ij} & при \quad j\geq i; \\ a_{ij} & при \quad j<i, \end{cases}$ (i=1,…,4; j=1,…,6). В матрице A найти количество отрицательных элементов в четных столбцах. Найти минимальный элемент в четвертом столбце матрицы B и его индекс (номер строки)
	Получить матрицу A(5,5), для которой $a_{ij}=\sin\left(\frac{i^2+j^2}{5}\right)$, (i=1,…,5; j=1,…,5). Вычислить Z=S/K, где S – сумма элементов побочной диагонали матрицы A; K – количество положительных элементов матрицы A. Построить матрицу B(5,5) по правилу $b_{ij}=\begin{cases} z\,a_{ij} & при \quad a_{ij}\geq 0; \\ a_{ij} & при \quad a_{ij}<0 \end{cases}$
	Получить матрицу A(4,4), для которой $a_{ij}=\begin{cases} \sin(i+j) & при\ i<j; \\ 1 & при\ i=j; \\ \sin\left(\frac{1+j}{2i+j}\right) & при\ i>j, \end{cases}$ (i=1,…,4; j=1,…,4). Найти сумму элементов, стоящих правее второго столбца. В третьей строке определить количество отрицательных элементов

Окончаниее табл. 4.5

	Дана целочисленная матрица A(5,3). Найти amax – максимальный элемент в ее нечетных строках. Сформировать матрицу B(5,3) по правилу: $b_{ij} = \begin{cases} \dfrac{a_{ij}}{a\max} & при \quad i - нечет. \\ a_{ij}\, a\max & при \quad i - чет. \end{cases}$, (i=1,…,5; j=1,…,3). Определить среднее арифметическое элементов второй строки матрицы B

Контрольные вопросы

1. Дайте определение массива.
2. Поясните формат описания массива. Приведите примеры.
3. Каким образом осуществляется доступ к элементам массива? Приведите примеры.
4. Какие способы инициализации массива Вы знаете?
5. Как связаны максимальный индекс и размерность массива?
6. Как описываются двумерные массивы?
7. Как можно инициализировать двумерный массив? Приведите примеры.
8. В чем заключается сущность задачи сортировки массива?
9. Для чего в программах используются манипуляторы?
10. Приведите примеры использования манипуляторов вывода чисел.
11. Какой заголовочный файл необходимо подключить к программе для использования манипуляторов с аргументом?
12. Какие манипуляторы наиболее часто используются для формирования таблиц?

5. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ

Методика нисходящего проектирования больших программ предполагает их последовательное структурирование до уровня функционально независимых подпрограмм.

5.1. Организация подпрограмм на языке Turbo Pascal

В Turbo Pascal (TP) применяются два вида подпрограмм: *функции* и *процедуры*. Функции ориентированы на получение результата в виде одного скалярного значения, а процедуры – нескольких переменных, в том числе массивов.

Подпрограммы разделяются:

1) на *стандартные*, являющиеся неотъемлемой частью интегрированной среды TP (библиотека Turbo Pascal); их не требуется описывать в программах;

2) на *собственные*, т.е. разработанные пользователем; их необходимо описывать в разделе определений программ.

Подпрограммы в TP имеют ту же структуру, что и программы. Они состоят из заголовка, раздела определений и описаний и тела подпрограммы (операторная часть). В конце подпрограммы в отличие от программы ставится точка с запятой.

Общая структура функции и процедуры имеет соответственно вид, изображенный на рис. 5.1 и 5.2.

Список параметров в заголовке функций и процедур обеспечивает связь подпрограммы с вызывающей их программой. Через него в подпрограмму передаются исходные данные и возвращается результат.

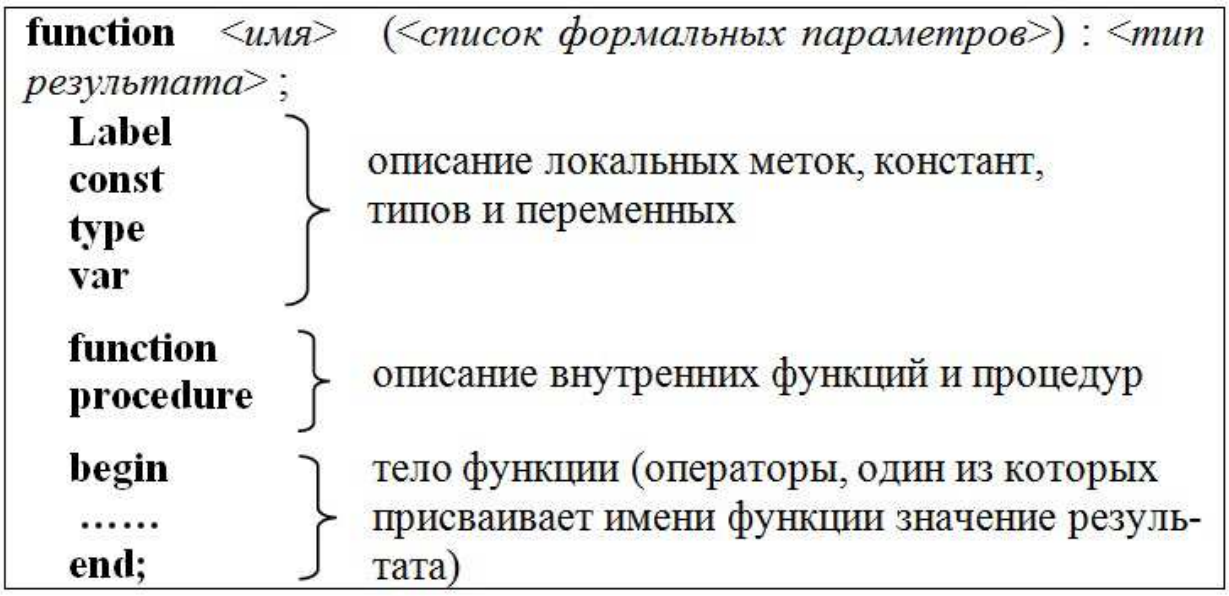


Рис. 5.1. Общая структура функции

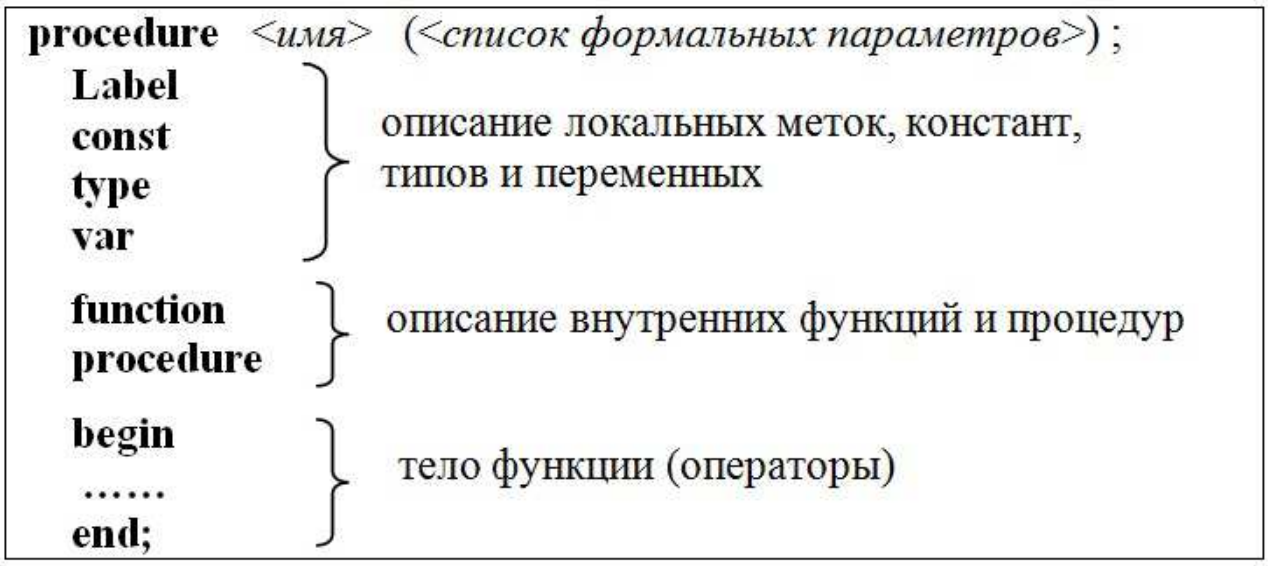


Рис. 5.2. Общая структура процедуры

Список формальных параметров в общем случае включает:

1) *параметры-значения*, которые приводятся без каких-либо ключевых слов:

(…; q1, …, qn: T1; …;p1, …, pm: Tk; …)

2) *параметры-константы*, которые записываются после ключевого слова **const**:

(…; **const** q1, …, qn: T1; …;**const** p1, …, pm: Tk; …)

3) *параметры-переменные*, которые приводятся с ключевым словом **var**:

(…; **var** q1, …, qn: T1; …;**var** p1, …, pm: Tk; …)

Здесь q_i, p_i – формальные параметры; T_i – типы формальных параметров.

Формальные параметры можно указывать в любом порядке, при этом параметры одного типа не обязательно группировать в одном месте. Операторы тела подпрограммы рассматривают их как своеобразное расширение раздела описаний: все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы.

Обращение к функции осуществляется подстановкой ее имени в выражение оператора присваивания вызывающей программы (подпрограммы) с указанием в скобках через запятую фактических параметров. В отличие от функций обращение к процедуре производится как к отдельному оператору с помощью записи имени с фактическими параметрами. При этом фактические параметры должны соответствовать формальным по количеству, типу и месту расположения в заголовке подпрограммы, т.е. фактические параметры следует записывать в той же последовательности, что и формальные.

В качестве фактических параметров-значений и параметров-констант можно применять любые выражения, в простейшем случае – переменные или константы различных типов, а в качестве фактических параметров-переменных – переменные любых типов, кроме констант. Параметры-значения и параметры-константы используются для передачи входных, а параметры-переменные – выходных данных подпрограммы.

Тип формального параметра должен определяться одним идентификатором. Поэтому если параметром подпрограммы является массив, то перед описанием подпрограммы необходимо дать определение типа массива. Например,

```
const N = 10 ;
type matrix = array [1 .. N, 1 .. N] of real ;
. . . . . .
procedure EX (a, b : matrix ; var c : matrix) ;
```

Аналогично если возникает необходимость описать в заголовке подпрограммы как формальный параметр процедуру или функцию, то последние должны быть *типизированы*. Например,

```
type yOFx = function (x : real) : real;
. . . . . .
function Integral (a, b : real ; y : yOFx ; N : word) : real;
```

При этом вызываемая функция оформляется в виде подпрограммы с директивой **far**, которая записывается после ее заголовка, в частности в рассмотренном примере:

```
function F (z : real) : real ; far ;
```

Имена, объявленные в вызывающей программе (подпрограмме), являются *глобальными* по отношению к вызываемой подпрограмме, т.е. они могут использоваться в теле последней и соответственно изменять свои значения. Это, по существу, дополнительный способ передачи результатов из подпрограммы. В предельном варианте подпрограмма может быть организована без параметров, т.е. подпрограмма будет связана с программой посредством глобальных имен.

Имена, объявленные в заголовке подпрограммы и в разделе описаний, являются *локальными*. Они не должны совпадать с именем самой подпрограммы. Доступ к ним из других программ и подпрограмм невозможен. При совпадении локальных и глобальных имен локальные определения в пределах своего действия отменяют действие глобальных (блокируют их).

Примером взаимодействия подпрограмм может служить структура программы, приведенная на рис. 5.3.

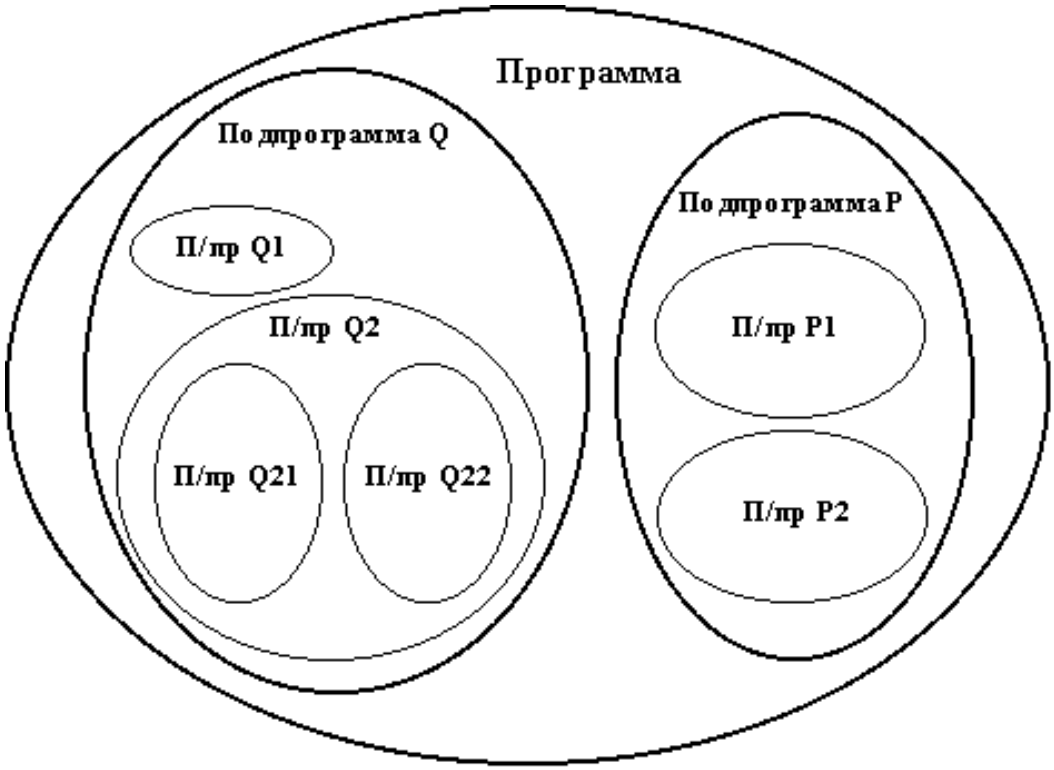


Рис. 5.3. Программа с вложенной структурой подпрограмм

При входе в подпрограмму низшего уровня доступными становятся не только объявленные в ней имена, но и сохраняется доступ ко всем переменным (именам) верхнего уровня. Например, из подпрограммы Q21 можно вызвать подпрограмму P и Q1, использовать имена, объявленные в основной программе и подпрограммах Q и Q2.

Локализация переменных делает подпрограмму независимой от основной программы. Общение между ними, как правило, должно идти через список параметров подпрограммы, что придает последней необходимую гибкость, поэтому в соответствии с требованиями хорошего стиля программирования рекомендуется там, где это возможно, использовать передачу результатов через фактические параметры-переменные.

Для досрочного выхода из подпрограммы предусмотрена стандартная процедура **exit**.

В TP возможен рекурсивный вызов процедур или функций. Это делает программу иногда изящной, но, как правило, ненадежной из-за зацикливания программы и переполнения стека. Поэтому на практике рекурсия заменяется итерацией, что в большинстве случаев выполнимо.

Пример. Составить программу для вычисления значения z: $z = \frac{th^2(a) + th(a-b)}{\sqrt{th(a^2 - b^2)}}$, где $th(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$ при заданных значениях a и b.

Вычисление th(x) можно оформить в виде *функции*:

```
Program Th_1;
   Var z,a,b,t1,t2,t3:real;
    {вычисление th с помощью функции}
    Function th(x:real):real;
     var c:real;
       begin
         c:=exp(2*x);
         th:=(c-1)/(c+1);
       end;
   BEGIN
     writeln('Введите переменные:');
     write('a='); readln(a);
     write('b='); readln(b);
       t1:=sqr(th(a));       {обращения к функции th}
       t2:=th(a-b);
       t3:=sqrt(th(a*a-b*b));
       z:=(t1+t2)/t3;
     writeln('z=',z:12,'  при a=',a:4:1, ' и b=',b:4:1);
   END.
```

При оформлении вычисления th(x) в виде *процедуры* программа будет иметь следующий вид:

```
Program Th_2;
   Var z,a,b,y1,y2,y3:real;
    {вычисление th с помощью процедуры}
```

```
  Procedure th(x:real;var y:real);
   var c:real;
     begin
       c:=exp(2*x);
       y:=(c-1)/(c+1);
     end;
BEGIN
  writeln('Введите переменные:');
  write('a='); readln(a);
  write('b='); readln(b);
    th(a,y1);          {обращения к процедуре th}
    th(a-b,y2);
    th(a*a-b*b,y3);
    z:=(sqr(y1)+y2)/sqrt(y3);
  writeln('z=',z:12,'  при a=',a:4:1, ' и b=',b:4:1);
END.
```

5.2. Программирование с использованием модулей на языке Turbo Pascal

Модули – автономно компилируемые программные единицы. Их использование упрощает модификацию программ, их тестирование и обнаружение ошибок. Кроме того, модули не подвержены влиянию глобальных переменных, поэтому могут использоваться как строительные блоки в других программах.

Структура модулей

Модуль состоит из заголовка и трех частей (разделов): интерфейсной, исполняемой и инициирующей.

Unit <имя модуля>; {unit – зарезервированное слово, начинает заголовок модуля}

{Имя модуля используется при задании связи с основной программой и другими модулями. Эта связь устанавливается путем включения идентификатора модуля в списки **uses**-предложений.

Имя модуля должно совпадать с именем дискового файла, в который помещается исходный текст модуля.

Например, **Unit Global**; размещается в **Global.pas**.}

Interface {начало интерфейсного раздела}

{В этом разделе описывается взаимодействие данного модуля с другими пользовательскими и стандартными модулями, а также с главной программой}

Uses {начинает список *импорта* интерфейсного раздела}

{В этом списке через запятые перечисляются идентификаторы модулей, информация интерфейсных частей которых должна быть доступна в данном модуле. Однако здесь целесообразно описывать идентификаторы только тех модулей, информация из которых используется в описаниях раздела **interface** данного модуля. При этом модули, описанные в **interface**, будут доступны также и в **implementation**}

const
type
var
procedure
function

{Список *экспорта* интерфейсного раздела. Содержит объявления всех глобальных объектов модуля (констант, типов, переменных и заголовки подпрограмм с формальными параметрами), которые можно использовать во всех других модулях и программах, если включить в строку **uses** последних имя данного модуля}

Implementation {начало раздела реализации (исполняемой части)}

{В этом разделе указывается реализационная часть описаний данного модуля, которая недоступна для других модулей и программ.}

Uses {список *импорта* раздела реализации}

{В этом списке через запятые перечисляются идентификаторы модулей, информация интерфейсных частей которых

должна быть доступна в данном модуле. Здесь целесообразно описывать идентификаторы всех необходимых модулей, информация из которых не используется в описаниях раздела **interface** данного модуля и об использовании которых не должен знать ни один другой модуль.

Следует учитывать, что несмотря на то, что информация из модуля, описанного в **interface** данного модуля, используется в **implementation**, указывать его в этом списке недопустимо, так как модуль может быть указан только один раз – в **interface** или **implementation**.}

const
type
var
procedure
function

{Подразделы внутренних для модуля описаний (секции). В них описываются локальные в **implementation** константы, типы, переменные, которые недоступны ни одному другому модулю. Здесь описываются также процедуры и функции, заголовки которых указаны с формальными параметрами в экспортном подразделе **interface**. При этом их заголовки в **implementation** допускается указывать без списка параметров; если списки приводятся, то они должны быть идентичны спискам в **interface**.

procedure
function

В этом же подразделе описываются внутренние процедуры и функции; они недоступны для других модулей. Эти подпрограммы всегда описываются полностью с формальными параметрами.}

Begin {начинает раздел инициализации (инициирующая часть)}

{В этом разделе указываются операторы начальных установок, необходимых для запуска корректной работы модуля, т.е. подготовительные операции. Эти операторы выполняются при начальном запуске программы в порядке их описания в **uses**-предложениях основной программы.

Если операторы инициализации не требуются, то ключевое слово **Begin** может быть опущено}

End.

Пример. Составить программу расчета $y = \frac{1+\mathrm{tg}^2\alpha}{k!}\left(1-\frac{1}{m!}\right)$. При этом вычисление функции **tg x** (**x** – аргумент в градусах) и факториала **n!** объединить в отдельный модуль.

```
unit matem;        {заголовок модуля}
 interface
{список uses отсутствует, так как нет необходимости в использовании других модулей}
{список экспорта, включающий заголовки используемых подпрограмм}
  function tg(x:real):real;
  procedure fact(n:byte;var f:real);
 implementation
{список uses отсутствует, так как нет необходимости в использовании других модулей}
function tg; {реализация функции tg x}
    begin
  x:=x*pi/180;
    tg:=sin(x)/cos(x)
    end;
 procedure fact;     {реализация процедуры вычисления
  var i:byte;        факториала}
    begin f:=1;
      for i:=1 to n do
      f:=f*i;
    end;
{инициирующая часть отсутствует, так как в данном модуле инициализация не требуется}
end.
```

Далее этот модуль подключается к основной программе с условным названием Proba.

```
Program Proba;                {основная программа}
uses Matem;       {список используемых модулей}
var al,y:real; k,m:byte; f1,f2:real;
 begin
   write('al=');readln(al);
   write('k=');readln(k);
   write('m=');readln(m);
  {обращения к процедуре модуля matem}
   fact(k,f1); fact(m,f2);
  {обращения к функции модуля matem}
   y:=((1+tg(al)*tg(al))/f1)*(1-1/f2);
   writeln('y=',y:12);
 end.
```

5.3. Организация функций на языке C++

Любая программа на C++ состоит из функций, по крайней мере, одной функции **main**, называемой **главной** функцией. С нее всегда начинается выполнение программы.

Каждая функция должна быть определена или объявлена до ее использования в программе. **Определение** (описание) функции состоит из заголовка и тела функции и имеет следующую форму:

тип_результата имя_функции *(список_параметров)* //заголовок функции

{тело_функции}

Тип возвращаемого значения (результата) может быть любым, кроме массива и функции. Если функция не должна возвращать значение, то указывается тип **void**, например:

void main ( )

Имя функции желательно подбирать, исходя из содержания решаемой задачи. Это удобно для пользователя, так как имя функции

будет давать представление о ее назначении.

Например: stepen, summa, sort, koren.

Список параметров (аргументов) определяет величины, которые требуется передать в функцию при ее вызове. Их часто называют **формальными** параметрами. Элементы списка разделяются запятыми. Для каждого параметра указывается его тип и имя. При отсутствии аргументов список может быть пустым ( ) или иметь спецификатор **void.**

Тело функции представляет собой блок объявлений и операторов, описывающих определенный алгоритм. Особое место среди них занимает оператор **return.** Он обеспечивает немедленный возврат в вызывающую функцию и может использоваться для передачи вычисленного значения функции.

Пример. Определение функции вычисления $\sqrt[n]{x}$:

```
double koren (double x, int n)  //заголовок функции
{  double y=pow(x,1.0/n);       //1.0 исключает целочисленное деление
   return y;                    //возвращение значения  y
 }
```

Если описание функции следует за использованием функции, то в начале исходного файла (программы) необходимо поместить ее объявление (прототип).

Прототип функции по форме такой же, как заголовок. Однако имеются два существенных отличия: во-первых, прототип всегда заканчивается символом '**;**', во-вторых, в списке параметров имена аргументов функции можно не указывать, так как компилятор их игнорирует. То есть объявление функции указывает тип возвращаемого значения, количество и типы параметров. Например, для выше рассмотренной функции прототип имеет вид

```
double koren (double, int);
```

После того как функция объявлена, ее можно использовать в программе. Если функция описана раньше ее использования, то прототип

не требуется.

Для **вызова** функции нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена **фактических** параметров. Число и типы фактических аргументов должны совпадать с числом и типом формальных параметров функции. Соответствие между формальными и фактическими параметрами устанавливается по порядку их расположения в списках.

Вызов функции может находиться в любом месте программы, но только после объявления функции. Если тип возвращаемого значения не **void**, вызов функции может осуществляться из выражений.

Механизм возврата из функции реализуется оператором

return *выражение*;

Функция может содержать несколько операторов **return** (это определяется потребностями алгоритма). Для функции типа **void** выражение в операторе **return** не указывается; более того, если возврат из нее происходит в конце тела функции перед закрывающей фигурной скобкой, оператор **return** можно опустить. В этом случае компилятор предполагает, что оператор **return** находится в самом конце тела функции и добавляет его сам при компиляции.

Выражение в операторе **return** должно иметь тип, указанный перед именем функции в ее определении, либо иметь тип, допускающий автоматическое преобразование к типу возвращаемого функцией значения.

Пример. Составить программу с функцией вычисления максимума из двух чисел.

```
#include <iostream.h>
float max(int, int);        // прототип
void main(void)             //void в скобках можно опустить
{ int x,y;
cin>>x>>y;                  //Ввод фактических параметров
float z=max(x,y);           //Вызов функции с возвращением значения
cout<<"z="<<z;              //z можно заменить вызовом max(x,y)
return;                     //Оператор return можно опустить
}
//Описание функции max
```

```
float max(int a,int b)      //a и b – формальные параметры
{ return (a>=b)? a:b;     //Автоматическое преобразование int в float
 }
```

Выше были рассмотрены функции со скалярными аргументами. Использование массива в качестве передаваемых в функцию параметров требует несколько иной формы записи заголовка, а именно:

тип имя_функции ***(тип размерность, тип имя_массива[ ])*** **,**

причем размерностей и массивов может быть несколько, а порядок их перечисления – любым. Такие функции либо не возвращают результат (тип void), либо возвращают скалярное значение, например описание функции с двумя массивами одинаковой размерности:

float summ(int n, float a[], float b[])

Прототип соответственно имеет следующий вид:

float summ(int, float [], float []);

Вызов такой функции осуществляется в обычной форме:

summ(m, x, y),

где *m*, *x*, *y* – соответственно фактические размерность и имена массивов.

C++ имеется возможность задавать начальные значения параметров функции, которые иначе называются **аргументами по умолчанию**. Данные параметры должны быть последними в списке аргументов. В качестве значений параметров по умолчанию могут использоваться выражения, константы и глобальные переменные. Параметры по умолчанию должны быть указаны при первом упоминании имени функции – обычно в прототипе.

При вызове функции параметр по умолчанию можно не указывать, если фактический параметр с ним совпадает. При этом должны быть опущены и все аргументы, стоящие за ним. Благодаря такой возможности оператор вызова несколько упрощается.

Примеры. Использование параметров по умолчанию:

а) объявления:

int summ (int a, int b=3);

double f (int, int=50, float=0.26);

void err (int=n); //n – глобальная переменная

б) вызовы функций:

summ (7); //опущен аргумент b=3

summ (7,5); //второй аргумент изменен: b=5

f (3); //опущены последние параметры 50 и 0.26

err( ); //аргумент равен n

В C++ свойство вложенности функций не предусмотрено, то есть нельзя внутри тела одной функции определить другую функцию. Однако можно вызывать одну функцию из другой. Кроме того, функция может вызвать саму себя (рекурсия).

Функции обмениваются информацией тремя способами: с помощью глобальных переменных, через параметры заголовка функции и через возвращаемое функцией значение. Последние два способа были уже рассмотрены.

Глобальные и локальные переменные

Глобальная переменная объявляется вне каких-либо функций, в том числе **main**. Она может использоваться в любом месте программы, начиная от места объявления. Полной противоположностью глобальной переменной является **локальная** переменная, которая описывается внутри функции. Областью ее действия является функция. Между вызовами одной и той же функции значение локальной переменной не сохраняется. Если этого требуется избежать, при объявлении локальных переменных используется модификатор **static**, например:

. . . . . .

static int n=0; n++; //при повторном вызове n=1, далее n=2 и т.д.

. . . . . .

Если локальная и глобальная переменные имеют одно и то же

имя, то локальная переменная "блокирует" в функции действие глобальной переменной, т.е. глобальная переменная в этом случае не видима.

Использование глобальных переменных в функциях не рекомендуется, так как это препятствует их помещению в библиотеку общего пользования. Необходимо всегда стремиться к тому, чтобы функции были максимально независимыми и обменивались информацией через свои заголовки.

Пример. Составить программу вычисления значения

$$y = \operatorname{tg} d \sum_{i=1}^{k} \frac{b_i^2 + d}{(2i)!} + e^d \sum_{i=1}^{m} \frac{c_i^3 + d}{(3i)!},$$

при k=10, m=4, d=2,5; b_i=1…4,6; c_i=1…3,6.

Вычисление суммы и факториала оформить в виде функций.

Программа для решения данной задачи приведена в лист. 6.1.

Листинг 6.1. ***faktorial.cpp***

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
   float d=2.5;                              //Глобальная переменная
 //Прототип функции summ с параметром по умолчанию
   double summ(double,double,int,int=2);
 //Описание основной функции main
void main()                          /*Функция main не имеет параметров
                                       и не возвращает значение*/
{
  int k,m;                             //Локальные переменные
   double b1,bk,c1,ck,y,y1,y2;         //функции main
   //Ввод исходных данных
 cout<<"k=";cin>>k;
 cout<<"m=";cin>>m;
 cout<<"b1=";cin>>b1;                 //Начальное значение b_i
 cout<<"bk=";cin>>bk;                 //Конечное значение b_i
```

```
  cout<<"c1=";cin>>c1;              //Начальное значение ci
  cout<<"ck=";cin>>ck;              //Конечное значение ci
  y1=tan(d)*summ(b1,bk,k);      //Параметр по умолчанию опущен
  y2=exp(d)*summ(c1,ck,m,3);   //Параметр по умолчанию изменен
  y=y1+y2;
  cout<<"\nПолученный результат:";
  cout<<"y="<<setw(8)<<y;
}
    unsigned long fact(int);          //Прототип функции fact

   //Описание функции вычисления суммы sumt
   double summ(double x1,double xk,int p,int q)       //Заголовок
   { double x=x1,s=0,h;                          //Локальные переменные
     h=(xk-x1)/(p-1);                              //Вычисление шага
     for(int i=1;i<=p;i++)
     {
       s+=(pow(x,q)+d)/fact(q*i);                 //Вызов функции fact
       x+=h;
      }
      return s;                                  //Возвращение результата
   }
   //Описание функции вычисления факториала fact
   unsigned long fact(int a)                    //Заголовок функции fact
   {
     unsigned long f=1;                          //Локальная переменная f
     if(a==0) return 1;                        //Первый оператор возврата
     for(int j=1;j<=a;j++)
       f*=j;
     return f;                                 //Второй оператор возврата
   }
```

Результат выполнения программы

```
k=10
m=4
b1=1
bk=4.6
c1=1
ck=3.6
Полученный результат: y= 5.80738
```

!! *Проанализируйте программу. Создав новый файл проекта с именем faktorial.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Перегруженные функции

C++ позволяет определять несколько функций**,** реализующих один и тот же алгоритм, с одним и тем же именем. Эта особенность называется **перегрузкой функции**, а сами функции - **перегруженными функциями**. Применение таких функций делает программы более понятными и легко читаемыми.

Перегруженные функции различаются компилятором с помощью так называемой **сигнатуры –** списка типов их аргументов. Однако, используя в перегруженных функциях параметры по умолчанию, следует быть осторожным, так как функция с пропущенными аргументами может быть вызвана не той перегруженной функцией.

Примером эффективного использования перегрузки функций является задача сортировки массивов различных типов. Ниже приведена программа сортировки методом "пузырька" числовых массивов типа int, long, float и double различной размерности (лист. 6.2).

Листинг 6.2. ***func.cpp***

```
#include <iostream.h>
#include <math.h>
//Прототипы перегруженных функций sort
```

```
void sort(int,int[]);
void sort(int,long[]);
void sort(int,float[]);
void sort(int,double[]);
void main()                          //Основная программа
{
/*Корректируемые параметры до использования
  программы: значение m и тип z */
  const int m=5;                    //Размерность массива чисел
  double z[m];                      //Тип массива чисел z
  //Ввод массива
  cout<<"Введите элементы массива z[]:\n";
   for(int i=0;i<m;i++)
  {
     cout<<"z["<<i<<"]=";cin>>z[i];
  }
  cout<<"\nИсходный массив чисел:\n"<<"z["<<m<<"]={";
  for(int i=0;i<m;i++)
     cout<<z[i]<<"  ";
     cout<<"}"<<endl;
  cout<<"\nОтсортированный массив чисел:\n"<<"z["<<m<<"]={";
  sort(m,z);      //Обращение к функции sort с массивом типа double
  cout<<"}";
}
//Перегруженная функция для массива типа int
void sort(int n,int mass[])       //Заголовок с параметром - массивом
{
 //Сортировка методом "пузырька"
 for(int i=0;i<n-1;i++)
 { int a;
  for(int j=i+1;j<n;j++)
  if (mass[j]<mass[i])
    {a=mass[i];
```

```
      mass[i]=mass[j];
      mass[j]=a;
     }
  }
   //Вывод отсортированного массива
   for(int i=0;i<n;i++)
       cout<<mass[i]<<"  ";
}
//Перегруженная функция для массива типа long
void sort(int n,long mass[])
{
 for(int i=0;i<n-1;i++)
 { long a;
  for(int j=i+1;j<n;j++)
  if (mass[j]<mass[i])
    {a=mass[i];
     mass[i]=mass[j];
     mass[j]=a;
    }
 }
  for(int i=0;i<n;i++)
      cout<<mass[i]<<"  ";
}
//Перегруженная функция для массива типа float
void sort(int n,float mass[])
{
 for(int i=0;i<n-1;i++)
 { float a;
  for(int j=i+1;j<n;j++)
  if (mass[j]<mass[i])
    {a=mass[i];
     mass[i]=mass[j];
     mass[j]=a;
```

```
    }
 }
  for(int i=0;i<n;i++)
   cout<<mass[i]<<"  ";
}
 //Перегруженная функция для массива типа double
void sort(int n,double mass[])
{
 for(int i=0;i<n-1;i++)
 { double a;
  for(int j=i+1;j<n;j++)
  if (mass[j]<mass[i])
    {a=mass[i];
     mass[i]=mass[j];
     mass[j]=a;
    }
 }
  for(int i=0;i<n;i++)
      cout<<mass[i]<<"  ";
}
```

Результат выполнения программы

```
Введите элементы массива z[]:
z[0]= -2.3
z[1]= 5.8
z[2]= 1.2
z[3]= -6.7
z[4]= 0.1
Исходный массив чисел:
z[5]={-2.3   5.8   1.2   -6.7   0.1 }
Отсортированный массив чисел:
 z[5]={-6.7   -2.3   0.1   1.2   5.8 }
```

!! *Проанализируйте программу. Создав новый файл проекта с именем func.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Упражнения

Составить и отладить программу решения задачи согласно приведенным в табл. 5.1 вариантам заданий.

Таблица 5.1

Варианты заданий

	Вычислить $z = f(a,b) + f(a^2,b^2) + f(a^2-1,b) + f(a-b,b) + f(a^2-b^2,b^2-1)$, где $f(u,t) = \begin{cases} u^2+t^2, \text{ если } u>0 \text{ и } t>0, \\ u+t^2, \text{ если } u\le 0 \text{ и } t\le 0, \\ u-t, \text{ если } u>0 \text{ и } t\le 0, \\ u+t, \text{ если } u\le 0 \text{ и } t>0, \end{cases}$ при $\begin{cases} a = 2.5, & b = -7.3; \\ a = -0.5, & b = 4.2; \\ a = -0.2, & b = -0.42; \\ a = 23.7, & b = 41.2 \end{cases}$ с оформлением вычисления f(u,t) в виде функции
	Вычислить $z = f(\sqrt{\lvert x\rvert}, y) + f(a,b) + f(\sqrt{\lvert x\rvert}+1, -y) + f(\lvert x\rvert - \lvert y\rvert, x) + f(x+y, a+b)$, где $f(u,t) = \begin{cases} u+2t, & \text{если } u\ge 0, \\ u+t, & \text{если } u\le -1, \\ u^2-2t+1, & \text{если } -1<u<0, \end{cases}$
	при $\begin{cases} x = 2.31; & y = 42; & a = 3.1, & b = 0.02; \\ x = -4.21; & y = -31.2; & a = 1.2, & b = -3.2; \\ x = 0.34; & y = 17.2; & a = -4.6, & b = -0.44; \\ x = -14.2; & y = 0.32; & a = 7.2, & b = 4.7 \end{cases}$ с оформлением вычисления f(u,t) в виде функции

Вычислить

$$z = f(\sin x + \cos y, x + y) + f(\sin x, \cos y) + f(x - y, x) + f(\sin^2 x - 2, a) + f(a + 3, b + 1),$$

где $f(u,t) = \begin{cases} u + t, & \text{если } u > 1, \\ u - t, & \text{если } 0 \le u \le 1, \\ t - u, & \text{если } u < 0, \end{cases}$

при $\begin{cases} x = 0.785;\ y = 0.41;\ a = 0.1,\ b = -2.1; \\ x = 0.32;\ y = 0.314;\ a = -0.21,\ b = 4.2; \\ x = 19.2;\ y = 0.48;\ a = -4.3,\ b = -6.1; \\ x = 0.628;\ y = 2/3;\ a = 17.1,\ b = 0.2 \end{cases}$

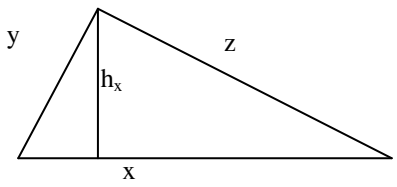
с оформлением вычисления f(u,t) в виде функции

Составить программу расчета значения z

$$z = \frac{\sin y \log_2 x + 0{,}37 \cos x \log_5 y}{b^y \log_{(b+2)}(x + y)}$$

при x = 9.2, y = 7.45, b = 2 с функцией вычисления логарифма

Составить программу вычисления высот треугольника со сторонами a, b, c. Вычисление высоты оформить в виде функции, используя известную тригонометрическую формулу:

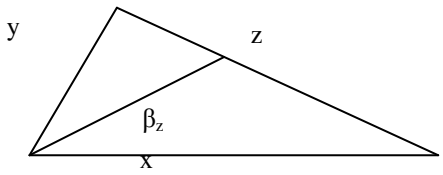


$$h_x = \frac{2}{x}\sqrt{p(p-x)(p-y)(p-z)},$$

где $p = \frac{1}{2}(x + y + z)$

Предусмотреть проверку возможности построения треугольника по заданным сторонам

Окончание табл. 5.1

	Составить программу вычисления биссектрис треугольника по сторонам a, b и c. Вычисление биссектрисы оформить в виде функции, используя известную тригонометрическую формулу: $\beta_z = \frac{2\sqrt{xyp(p-z)}}{x+y}$,  где $p = \frac{1}{2}(x+y+z)$ Предусмотреть проверку возможности построения треугольника по заданным сторонам
	Составить программу вычисления значения y: $y = \frac{\cos b\, th^3(\sqrt{a}) + \sin a\, \mathrm{th}(a-b)}{\mathrm{ctg}\,(a+b)\sqrt[4]{\lvert \mathrm{th}\left(a^3-b^3\right)\rvert}}$, где $\mathrm{th} = \frac{e^{2x}-1}{e^{2x}+1}$ при a= 4,7, b= 2,4. С оформлением вычисления th в виде функции. с оформлением вычисления f(u,t) в виде функции
	Составить программу вычисления значения y: $y = \frac{0{,}75\sum_{i=1}^{n} x_i y_i + 3{,}4\sum_{i=1}^{m} c_i d_i}{7{,}7\sum_{i=1}^{r} x_i c_i}$ при n = 15, m=7, r=11, x_i = 0.2..1.4, y_i = 1..15, c_i = 0.1..2.9, d_i = 1..3.8, оформив определение суммы произведения двух индексных переменных в виде функции

Контрольные вопросы

1. Чем отличаются структурированные программы от монолитных?
2. Какие подпрограммы называются стандартными? Приведите примеры.
3. Какой вид имеет структура описания процедуры?
4. В чем состоит отличие описания функции от процедуры?

5. Что такое область действия идентификаторов?

6. Каковы основные правила определения области действия для идентификаторов процедур и функций?

7. Какие параметры называются формальными и какие – фактическими?

8. По каким признакам различаются параметры подпрограмм?

9. Как осуществляется обращение к функции и процедуре?

10. В каких случаях подпрограмма реализуется как процедура, а в каких – как функция?

11. Как описать массив в заголовке подпрограммы?

12. Как передать в подпрограмму функцию или процедуру как формальный параметр?

13. Для чего предназначена директива **far**?

14. Что такое рекурсивный вызов подпрограмм?

15. Что такое определение функции в C++?

16. Для чего нужен прототип функции?

17. В чем отличие функции **main** от других функций?

18. Какие действия выполняет оператор **return**?

19. Поясните, как формируется список параметров в заголовке функции?

20. Чем отличается объявление функции от ее определения и когда объявление не требуется?

21. Поясните, когда применяется в функциях тип **void**?

22. Что такое параметры по умолчанию и когда их целесообразно использовать?

23. Почему не рекомендуется в функциях использовать глобальные переменные?

24. Каким образом можно сохранить значение локальной переменной между вызовами функции?

6. ОБРАБОТКА СИМВОЛЬНОЙ И СТРОКОВОЙ ИНФОРМАЦИИ

6.1. Обработка символьной информации на языке программирования Turbo Pascal

Turbo Pascal работает с символами, как с формальными кодами, интерпретация которых определяется таблицей кодирования, то есть каждый символ кодируется одним байтом (числом в диапазоне 0..255). В качестве основной таблицы кодирования символов используется американский стандартный код обмена информацией (ASCII).

Символьный тип относится к порядковому типу, что и определяет область применения символьных данных. Как отдельные элементы они могут входить и обрабатываться в составе некоторых структурированных типов данных, таких как строки, диапазоны, множества и др.

Символьный тип данных

Этот тип служит для проведения операций с символами (буквами, цифрами и различными значками). Для представления символов используется формат длиной в один байт.

Идентификатор типа – **Char** (от character – символ).

Задание символов в выражениях и константах осуществляется одним из двух способов:

1) символ приводится в кавычках, например, ’A’, ’1’;

2) указывается знак диеза (#) и номер символа в кодовой таблице ASCII (табл. 6.1); в частности, буква ’A’ может быть обозначена как #65.

Таблица 6.1

Таблица ASCII-кодов печатаемых символов

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	16	32	48 **0**	64 @	80 **P**	96 `	112 **p**	128 **А**	144 **Р**	160 **а**	176 ░	192 └	208 ╨	224 **р**	240 **Ё**
1	1	17	33 **!**	49 **1**	65 **A**	81 **Q**	97 **a**	113 **q**	129 **Б**	145 **С**	161 **б**	177 ▒	193 ┴	209 ╤	225 **с**	241 **ё**
2	2	18	34 "	50 **2**	66 **B**	82 **R**	98 **b**	114 **r**	130 **В**	146 **Т**	162 **в**	178 ▓	194 ┬	210 ╥	226 **т**	242 **Є**
3	1	19	35 **#**	51 **3**	67 **C**	83 **S**	99 **c**	115 **s**	131 **Г**	147 **У**	163 **г**	179 │	195 ├	211 ╙	227 **у**	243 **є**
4	2	20	36 **$**	52 **4**	68 **D**	84 **T**	100 **d**	116 **t**	132 **Д**	148 **Ф**	164 **д**	180 ┤	196 ─	212 ╘	228 **ф**	244 **Ї**
5	5	21	37 **%**	53 **5**	69 **E**	85 **U**	101 **e**	117 **u**	133 **Е**	149 **Х**	165 **е**	181 ╡	197 ┼	213 ╒	229 **х**	245 **ї**
6	6	22	38 **&**	54 **6**	70 **F**	86 **V**	102 **f**	118 **v**	134 **Ж**	150 **Ц**	166 **ж**	182 ╢	198 ╞	214 ╓	230 **ц**	246 **Ў**
7	7	23	39 '	55 **7**	71 **G**	87 **W**	103 **g**	119 **w**	135 **З**	151 **Ч**	167 **з**	183 ╖	199 ╟	215 ╫	231 **ч**	247 **ў**
8	8	24	40 **(**	56 **8**	72 **H**	88 **X**	104 **h**	120 **x**	136 **И**	152 **Ш**	168 **и**	184 ╕	200 ╚	216 ╪	232 **ш**	248 °
9	9	25	41 **)**	57 **9**	73 **I**	89 **Y**	105 **i**	121 **y**	137 **Й**	153 **Щ**	169 **й**	185 ╣	201 ╔	217 ┘	233 **щ**	249 •
A	10	26	42 *	58 **:**	74 **J**	90 **Z**	106 **j**	122 **z**	138 **К**	154 **Ъ**	170 **к**	186 ║	202 ╩	218 ┌	234 **ъ**	250 ·
B	11	27	43 **+**	59 **;**	75 **K**	91 **[**	107 **k**	123 **{**	139 **Л**	155 **Ы**	171 **л**	187 ╗	203 ╦	219 █	235 **ы**	251 ∫
C	12	28	44 **,**	60 **<**	76 **L**	92 ****	108 **l**	124 **|**	140 **М**	156 **Ь**	172 **м**	188 ╝	204 ╠	220 ▄	236 **ь**	252 **№**
D	13	29	45 **-**	61 **=**	77 **M**	93 **]**	109 **m**	125 **}**	141 **Н**	157 **Э**	173 **н**	189 ╜	205 ═	221 ▌	237 э	253 ¤
E	14	30	46 **.**	62 **>**	78 **N**	94 **^**	110 **n**	126 **и**	142 **О**	158 **Ю**	174 **о**	190 ╛	206 ╬	222 ▐	238 **ю**	254 ■
F	15	31	47 **/**	63 **?**	79 **O**	95 **_**	111 **o**	127 ⌂	143 **П**	159 **Я**	175 **п**	191 ┐	207 ╧	223 ▀	239 **я**	255

С символьными переменными можно производить:

1) операции присваивания.

Например: **var** c,y:char;

. . . . . .

c:=’$’; y:= #125;

2) операции отношения (**=, < >, <, >, >=, <=**). При этом сравнение символов производится по порядку номеров кодовой таблицы. Например:

```
var x,y:char;
. . . . . .
  if x<y then writeln(y);
. . . . . .
   while x<>'!' do write(x);
```

Результат операций **'B' > 'A' , 'A' < 'a'** равен True, так как символам 'A', 'B' и 'a' соответствуют номера 65, 66, 97;

К переменным типа Char могут применяться следующие встроенные функции:

1) Function **Ord** (x:char):longint; возвращает порядковый номер символа **x** в таблице ASCII.

Например: z:= ord('9');
write(z); {результат – 57}

2) Function **Chr** (x:byte):char; возвращает символ, соответствующий порядковому номеру **x** в таблице ASCII.

Например: **for** i:=128 **to** 159 **do** write(chr(i));
{результат – АБВГД . . Я}

3) Function **Pred** (x:char):char; возвращает символ, который предшествует символу **x** в таблице ASCII.

Например: y:=pred('j');
write(y); {результат – i}

4) Function **Succ** (x:char):char; возвращает символ, который следует за символом **x** в таблице ASCII.

Например: y:=succ('j');
write(y); {результат – k}

5) Function **UpCase** (x:char):char; применяется для преобразования строчной латинской буквы в прописную (при применении к кириллице возвращает аргумент без обработки).

Например: c:=UpCase('f');
write(c); {результат – F}

Примечание: функция **UpCase** не обрабатывает кириллицу, т.е. возвращает аргумент без обработки.

Так как символьные переменные относятся к порядковому типу, их можно использовать в качестве переменой цикла в операторе **For**.

Например:

For i:='A' **to** 'Я' **do** write (i); {результат – АБВГ . . Я}

или

For i:='z' **downto** 'a' **do** write(upcase(i)); {результат – ZYX . . A}

6.2. Строковый тип данных

Строковый тип (идентификатор типа **String**) используется для обработки строк (цепочки символов).

Описание переменных строкового типа осуществляется двумя эквивалентными способами:

1) с определением типа

Type

<имя типа> = **string**[<максимальная длина строки>];

Var

<имя переменной>:<имя типа>;

без определения типа

Var

<имя переменной>: **string**[<максимальная длина строки>];

Максимальная длина строки должна лежать в диапазоне 1..255. Если максимальная длина строки не указана, то по умолчанию она принимается равной 255 символам.

Например:
```
Type
    L25 = string[25];
    Lmax = string[80];
Var
    screen : Lmax;    {явное описание типа}
    y : L25;
    x : string;       {по умолчанию максимальная
                                длина строки – 255}
    z : string [40];  {неявное описание}
```

Все символы в строке нумеруются от 1 до максимального значения длины строки. К любому символу строки можно обратиться как к элементу одномерного массива.

Например:
```
Var st : string [10];
 . . . . . .
begin
. . . . . .
  for i:=1 to 10 do
     if st[i]='?' then …
. . . . . .
end.
```

В нулевую позицию автоматически записывается текущее значение длины строки – *динамическая длина*.

Значением строковой переменной может быть последовательность символов, взятая в апострофы. Если апостроф входит в состав строки, то он записывается дважды.

Например:
```
st1:='Интегрированная среда';
st2:='Под''езд';
st3:=''  {пустая строка}.
```

К строковым переменным применимы следующие операции:

1). Операция присваивания.

Например: MyStr : =’Turbo Pascal’;

YourStr : = MyStr;

Если при присваивании превышается максимальная длина строки, то лишние символы справа отсекаются.

2). Операции отношения (**=, <>, <, >, >=, <=**).

Стоки сравниваются посимвольно слева направо. При сравнении используются коды соответствующих символов . Строки считаются равными только при одинаковом наборе символов и одинаковой длине. Результат сравнения строк имеет значения False или True.

Например, следующие операции дадут True:

’N’<’n’; ’2’>’123’; ’while’>=’wHile’; ’Паскаль’>’Turbo Pascal’.

3). Операция сцепления (объединения).

Обозначается символом “+”. При выполнении операции к первой строке добавляется вторая строка, ограничивая общую длину в 255 символов или максимально допустимой длиной; "лишние" символы отбрасываются.

Например:

```
c:=’A’+’B’+’CDE’;  writeln(c);   {результат – ABCDE}
x:=’Интегрированная’;   y:=’ среда’;
z:=x+y;  writeln(z);        {результат – Интегрированная среда}
```

Ввод строковых переменных осуществляется только оператором **Readln**. Одним оператором можно ввести только одну строку.

Вывод строки осуществляется с помощью операторов **Write** и **Writeln**. К строковым переменным можно также применять форматированный вывод (как для целых чисел).

Например: A : =’student’;

```
writeln(A : 10);        {результат –      student}
writeln(A);             {результат – student}
writeln(A : 4);  {результат – stud}
```

Для большинства необходимых преобразований строки в TP предусмотрены специальные процедуры и функции:

1)**function Length (s : string) : integer;** возвращает динамическую длину строки.

Например: c:=length(’1230’+’456’);
writeln(c); {результат – 7}

2) **function Concat (s1 [,s2, …,sN] : string) : string;** выполняет конкатенацию (объединение) последовательности строк. Действия этой функции и операции сцепления “+” одинаковы.

Например: st1:=’Студент’;
st2:=’ – отличник’;
st3:=concat(st1,’ Петров’, st2);
{результат – st3=’Студент Петров - отличник’}

3)**function Copy (s : string; Index, Count : integer) : string;** возвращает подстроку из строки **s**, начиная с позиции **Index** длиной **Count** символов.

Например: st1:=’Программа’
st2:=copy(st1, 2, 3); {результат – st2=’рог’}
st3:=copy(st1, 4, 5); {результат – st3=’грамм’}

4)**function Pos (subS, s : string) : byte;** возвращает позицию, начиная с которой в строке **s** располагается подстрока **subS**. Если фрагмент в строке не найден, то **Pos**=0.

Например: S:=’Airplane’;
N1:=pos(’plan’, s); {результат – N1=4}
N2:=pos(’nn’, s); {результат – N2=0}
N3:=pos(’bar’, ’Barbara’); {результат – N3=4}

5)**procedure Delete (var s : string; Index, Count integer);** удаляет **Count** символов из строки **s**, начиная с позиции **Index** со смещением остатка строки на освободившееся место.

Например: а) удаление пробелов из начала строки:
while st[1]=’ ’ **do** delete (st, 1, 1);
б) удаление всех пробелов из строки:

while Pos(’ ’,st)<>0 **do** Delete (st, Pos (’ ’, st), 1)

6)**procedure Insert (subS : string; var S : string; Index : integer);** вставляет подстроку **subS** в строку **s**, начиная с позиции **Index**. Если Index превышает исходную длину строки, то **subS** присоединяется к строке **s** справа.

Например: вставить в строку ’студент – отличник’ свою фамилию:

```
Var st1, fam: string;
begin  writeln (’Введите свою фамилию’);
       readln (fam);
       st1:=’студент  – отличник’;
       insert (fam,st1,9);
       writeln (st1);        {результат: студент Иванов –
end.                          отличник}
```

7)**procedure Str (x[: Size[: Dec]], var s : string);** преобразует числовое значение **x** (целое или вещественное) в его строковое представление. **Size** и **Dec** поля, отводимые соответственно под запись всего числа и его дробной части (как в операторе write).

Например: A:= -6; B:= 8.1E-4;

```
str (A : 4, st1);          {результат – st1=’   -6’}
str (B : 15 : 10, st2);
   {результат – st2=’     0.0008100000’}
```

8)**procedure Val (s : string; var x : real (или integer); code : integer);** преобразует строковое значение **s** в его численное представление **x**. Параметр **code** содержит признак ошибки преобразования (0 – нет ошибки). Если в строке встречается недопустимый символ, то его номер сохраняется в параметре **code** (т.е. если строка не соответствует формату числа, то **code** содержит номер первого ошибочного символа).

Например:

```
st1:=’1234’; val (st1, R1, Ier1);  {результат – R1=1234; Ier1=0}
```

st2:=’7.7E5’; val (st2, R2, Ier2); {результат – R2=770000; Ier2=0}

st3:=’1+1’; val (st3, R3, Ier3); {результат – R3=0; Ier3=2}

При решении задач с символьными данными в качестве начальных значений переменных можно использовать типизированные константы. Задание констант строкового и символьного типов аналогично заданию констант простых числовых типов.

Например:

```
Const
  str : string = 'Turbo Pascal';
  no : char = 'N';
```

6.3. Множества

В TP множество – это конечная неупорядоченная совокупность элементов, принадлежащих некоторому базовому типу данных, который может быть только *порядковым* (кроме word, integer и longint, так как число элементов в них > 256).

Массивы и вещественные числа в множества входить не могут.

Максимально допустимое число элементов (мощность, кардинальное число) множества не может превышать 256, поэтому, порядковые значения базового типа лежат в диапазоне 0..255 включительно.

Описание переменных множественного типа осуществляется двумя эквивалентными способами:

1) с определением типа

Type

<имя типа множества>=**set of** <имя базового типа>;

Var

<имя переменной>:<имя типа множества>;

без определения типа

Var

<имя переменной>: **set of** <имя базового типа>;

Например, **Type**

```
Simbol = set of char;
Week = set of (Sun,Mon,Tue,Wed,Thu,Fri,Sat);
Month = set of 1..31;
   Var
        Letter : simbol;
        Day : Week;
        Date : Month;
        Rus : set of ’А’..’Я’;
```

Переменным множественного типа присваиваются значения в виде подмножеств. Подмножество – это последовательность элементов, разделенных запятыми, взятая в квадратные скобки [ ]. Это задание множества называется *конструктор* множества. Другим способом задания является диапазон. Обе формы конструирования подмножеств могут сочетаться.

Например: Sign:=[’+’,’–’];

```
Digits:=[’0’..’9’];
Letter:=[’А’..’Я’,’а’..’п’,’р’..’я’];
```

Над переменными множественного типа можно производить следующие операции:

1. **In** – проверка на принадлежность множеству. Результат проверки равен True, если первый операнд входит в множество второго операнда.

Например: **if** Temp **In** [0..100] **then** write (’Жидкость’);

if Bukva **In** Letter **then** write (Bukva);

2. Сравнение множеств (операции соотношения). Результат сравнения равен True, если:

а) при операции **=** сравниваемые множества совпадают;

б) при операции **<>** они не совпадают друг с другом;

в) при операции **>=** все элементы второго множества входят в первое;

г) при операции **<=** все элементы первого множества входят во второе;

Например, результаты приведенных сравнений истинны:

[5, 6] = [6, 5]; ['A']<>['B','C']; ['B','U','K','V','A'] >= ['K']; [1..3, 5] <= [0..10];

3. Объединение множеств (знак операции "+"). Результат объединения – множество, включающее элементы обоих множеств.

Например:

```
A:= ['Turbo'];
B:=A+['Pascal'];          {результат – B=['Turbo','Pascal']}
C:= ['Turbo','Pascal']+['Pascal']; {результат –C=['Turbo','Pascal']}
```

4. Разность множеств (знак операции "–"). Результат операции – множество, включающее все элементы первого множества, не входящие во второе множество.

Например:

```
A:= ['a','b','c'] – ['b','c'];      {результат – A=['a']}
B:= [1..10] – [1, 3, 5, 7, 9];      {результат – B=[2,4,6,8,10]}
```

5. Пересечение множеств (знак операции "*"). Результат – множество, включающее элементы первого и второго множеств одновременно.

Например,

```
A:= [1..10] * [8..14];                    {результат – A=[8, 9, 10];}
                          B:= ['A','B','C']*['K','L','M'];      {ре-
                                зультат – B=[ ] – пустое множество}
```

В некоторых случаях исходное множество удобно представить типизированной константой в виде правильного конструктора множества.

Например:

```
type
  Rus = set of 'А' .. 'Я';
const
  s : rus = ['А','Б','В','Г','Д'];
```

```
date : set of 1..31 = [7..9];
```

Пример. Составить программу удаления из строки лишних (нескольких идущих подряд) разделителей (знаков препинания, пробелов и т.п.). Подсчитать число удаленных знаков. Исходную строку и полученные результаты вывести на печать.

```
Program Deleting;
var st,st1:string;          {st – исходная строка,
   razd:set of char;                st1 – обрабатываемая строка}
   x,i:byte;
begin
 writeln('Введите исходную строку:');
 readln(st);
 st1:=st;
   {задание подмножества разделителей}
 razd:=[' ', ',', ';', '.', ':', '-'];
 i:=1;
 while i<length(st1) do        {посимвольный перебор строки}
  begin
   if (st1[i] in razd) and (st1[i]=st1[i+1]) then
   delete(st1,i+1,1)             {удаление лишнего знака}
   else i:=i+1;
  end;
 x:=length(st)-length(st1);
 writeln('Исходная строка:');
 writeln(st);
 writeln('Полученный результат:');
 writeln(st1);
 writeln('Обнаружено ',x,' лишних знаков');
 end.
```

Результат выполнения данной программы:

```
Исходная строка:
СфСамГТУ,,,,   ул..Советская,   д...45....
Полученный результат:
СфСамГТУ, ул.Советская, д.45.
Обнаружено 13 лишних знаков
```

!! *Проанализируйте программу Deleting, введите ее текст в компьютер, осуществите ее компиляцию, запустите на счет и просмотрите результаты.*

Упражнения

1. Составить программу вывода на печать символов таблицы ASCII, в виде 16 колонок, начиная с элемента #32.

2. Составить программу с функцией, в результате применения которой все строчные буквы русского алфавита в заданной строке заменяются на прописные (типа UpCase). Исходную стоку и полученный результат вывести на печать.

Исходная строка: «СфСамГТУ, г. Сызрань, ул. Советская, 45»

3. Составить программу, в которой требуется отформатировать введенную строку, выровняв ее по центру экрана. Исходную строку и полученный результат вывести на печать. При этом необходимо учитывать, что ширина экрана – 80 колонок, а выравнивание осуществляется добавлением (или удалением) в начало строки необходимого количества пробелов.

Исходная строка: «Обработка символьной информации»

4. Составить программу, в которой две введенные строки редактируются следующим образом:

а) определить текущее значение длин обеих строк; если строки равны по длине, то объединить их в одну строку;

б) сравнить строки по величине; большую строку вывести на печать.

Исходные данные и полученные результаты вывести на печать.

Исходные данные:

1) «Сф СамГТУ, », «г. Сызрань.»

2) «Филиал СамГТУ в Сызрани», «Филиал СамГТУ в Бузулуке»

5. Составить программу подсчета слов в введенном тексте (предложении). Примечание: все знаки препинания и пробелы учитывать в качестве разделителей. Исходные данные и полученный результат вывести на печать.

Исходная строка: «Деревня, где скучал Евгений, была прелестный уголок.»

6. Составить программу подсчета процентного содержания гласных и согласных букв и разделителей в заданной строке. Исходную строку и результаты вывести на печать. Результаты оформить в виде диаграммы:

Гласные: ▒▒▒▒▒▒▒▒▒▒ n1%

Согласные: ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ n2%

Разделители: ▒▒▒▒▒ (100 – n1 – n2)%

Для построения диаграммы использовать символы #176, 178, 219

Исходная строка: «Зима. Крестьянин, торжествуя, на дровнях обновляет путь»

Контрольные вопросы

1. Какой объем памяти занимает символьная переменная и в каком виде она представлена?

2. Каким образом можно задать символы в выражениях? Приведите примеры.

3. Какие операции можно производить с символьным типом данных? Приведите примеры.

4. Какие встроенные функции модуля System, предназначенные для символьного типа данных, вы знаете? Приведите примеры.

5. Каким образом можно описать в программе символьные переменные? Приведите примеры.

6. Что такое динамическая длина строки?

7. Где содержится информация о текущей длине строки и с помощью какой функции ее можно определить?

8. Расскажите об особенностях операций сравнения строк?

9. Какие операторы ввода и вывода используются для строкового типа данных? Приведите примеры.

10. Для чего предназначены функции Copy и Pos?

11. Поясните действие функций Delete и Insert.

12. В чем заключается операция сцепления и какая встроенная функция ее заменяет?

13. Каким образом описываются множества?

14. Как осуществляется конструирование множества? Приведите примеры.

15. Поясните с приведением примеров сущность операций сравнения множеств.

16. Каким образом осуществляется проверка на принадлежность переменной заданному множеству?

17. Приведите примеры операций объединения, разности и пересечения множеств.

18. К каким группам типов данных относятся символы, строки и множества?

19. В чем заключается отличие строк от массивов символов?

7. СТРУКТУРЫ ДАННЫХ

7.1. Программирование с использованием записей на языке Turbo Pascal

Для обработки информации, содержащей данные разных типов, в TP используется структурированный тип, который получил название *запись*. Такой тип является основой баз данных.

Запись – это структура данных, состоящая из заранее определенного количества компонент, называемых *полями*.

Общий вид записей:

Type <имя типа> = **record** <список полей> **end**;

или в развернутой форме:

```
Type V = record
           P1 : T1;          {V – имя типа,
           . . . . . .        Pi – имя поля,
           Pn : Tn;           Ti – тип поля}
         end;
```

Например:

```
Type  Birthday = record
                   date : 1..31;
                   month : 1..12;
                   year : word;
                 end;
Var
    a , b : Birthday;
```

Доступ к каждому из полей записи осуществляется с использованием составного имени в виде:

<имя переменной> . <имя поля>.

Например:

```
a .date := 20;
a .month := 11;
a .year := 1982;
```

Поля могут быть вложены друг в друга, тогда поля в составном имени необходимо описывать последовательно, начиная с внутреннего.

Например, предыдущее определение типа можно дополнить следующим описанием:

```
Type  anketa = record
                  name : string;
                  bd : Birthday;
               end;
Var   c : anketa;
```

В результате составное имя будет содержать больше полей.

Например:

```
c.name := ’Иванов’;
c.bd.date := 15;
c.bd.month := 9;
. . . . . .
```

Все значения полей одной переменной типа запись можно присваивать соответствующим полям другой переменной того же типа одним оператором присваивания:

```
b := a;
```

С внутренними полями записи можно выполнять любые операции, применимые к типу этих полей.

Ввод записей осуществляется только по внутренним полям с помощью операторов **Read** и **Readln**.

Например:

```
readln (c.name);
read (c.bd.date);
```

```
read (c.bd.month);
readln (c.bd.year);
```

Вывод значений ведется также по внутренним полям с помощью операторов **Write** и **Writeln**.

Например:

```
Writeln (c.name,' родился  ',c.bd.date,'.',c.bd.month,'.',c.bd.year, '  г.');
```

Для упрощения доступа к полям записи используется оператор присоединения **With**, который применяется, когда идет работа с одной конкретной переменной типа **record** или ее полями.

```
With N  do begin
    P1 := W1;          {N – имя записи,
    . . . . . .                  Pi –поле записи,
    Pn := Wn;          Wi – выражение}
      end;
```

Например:

```
With c.bd  do begin
          date := 15;
         month := 11;
          year := 1982;
        end;
```

Для удобства отладки программ, использующих записи, целесообразно использовать типизированные константы-записи. При этом перед определением типизированной константы-записи следует описать соответствующий тип-запись.

Список значений полей в определении констант-записей имеет следующий вид: имя поля, двоеточие и константа. Элементы списка отделяются друг от друга точкой с запятой.

Например:

```
type
  anketa = record
          fio : string [20];
```

```
            year : 1970..2000;
        end;
const
  first : anketa = (fio : 'Иванов' ; year : 1985);
```

Поля должны указываться в той последовательности, в какой они перечислены в объявлении типа.

Пример. Составить программу, которая позволяет организовывать список в виде массива записей, содержащий информацию о товарах (табл. 7.1), и сортировать этот список:

а) по алфавиту наименования товара;

б) по убыванию цены товара;

в) по убыванию числа единиц товара.

Вывести отсортированный список на печать.

Таблица 7.1

Исходный список товаров

Наименование товара	Цена	Количество
Карандаш	1.50	12
Ручка	3.40	20
Линейка	0.85	10
Ластик	1.00	25
Тетрадь	1.20	30

```
program Zapisi;
uses Printer;
const n=5;
type tovar=record
        t:string[10];  {наименование товара}
        p:real;        {цена товара}
        k:byte;    {количество единиц товара}
        end;
    massiv=array[1..n]of tovar;
var sp:massiv;
   x:tovar;
```

```
   i,j:byte;

          {процедура вывода списка товаров}
Procedure Output(spis:massiv);
 begin
  for i:=1 to n do
  writeln(i,'.',spis[i].t:10,'',spis[i].p:4:2,' ',spis[i].k);
  writeln;
 end;

{процедура сортировки товаров: при ch='t' - по алфавиту,
                                при ch='p' -  по убыванию цены,
                                при ch='k' -  по количеству}
Procedure Sort(ch:char; var spis:massiv);
  var y:tovar;
  begin
    for i:=1 to n-1 do
    for j:=i+1 to n do
    begin
       if (ch='t')and(spis[i].t>spis[j].t) or
         (ch='p')and(spis[i].p<spis[j].p) or
         (ch='k')and(spis[i].k<spis[j].k)
       then
       begin
       y:=spis[i];
       spis[i]:=spis[j];
       spis[j]:=y;
       end;
       end;
  end;
BEGIN
  {Ввод списка товаров}
for i:=1 to n do
```

```
begin
write('товар',i,':');readln(sp[i].t);
write('цена',i,':');readln(sp[i].p);
write('кол-во',i,':');readln(sp[i].k);
end;
  writeln('сортировка по алфавиту');
  Sort('t',sp);            {Сортировка списка}
  Output(sp);      {Вывод отсортированного списка}
  writeln('сортировка по убыванию цены');
  Sort('p',sp);            {Сортировка списка}
  Output(sp);       {Вывод отсортированного списка}
  writeln('сортировка по количеству');
  Sort('k',sp);            {Сортировка списка}
  Output(sp);       {Вывод отсортированного списка}
readln;
END.
```

Результаты:

```
сортировка по алфавиту
1.  Карандаш  1.50 12
2.   Ластик  1.00 25
3.  Линейка  0.85 10
4.    Ручка  3.40 20
5.  Тетрадь  1.20 30

сортировка по убыванию цены
1.    Ручка  3.40 20
2.  Карандаш  1.50 12
3.  Тетрадь  1.20 30
4.   Ластик  1.00 25
5.  Линейка  0.85 10

сортировка по количеству
1.  Тетрадь  1.20 30
```

2. Ластик 1.00 25
3. Ручка 3.40 20
4. Карандаш 1.50 12
5. Линейка 0.85 10

!! *Введите текст программы Zapisi в компьютер, осуществите ее компиляцию, запустите на счет и просмотрите результаты.*

7.2. Программирование с использованием структур на языке С++

Структура – представляет собой совокупность переменных, объединенных общим именем. В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов. Структуры предопределили новый вид данных типа класс, используемый в объектно-ориентированном программировании. На практике чаще всего обработка данных с помощью структур применяется в системах управления базами данных.

Определение структур

Описание структуры в общем случае имеет следующий вид:

```
struct [имя_типа_структуры]
  {тип_1  элемент_1;
тип_2  элемент_2;
. . . . . .
тип_N  элемент_N;
 } [список_имен_структур];
```

где **struct** – служебное (ключевое) слово; **[…] –** необязательный параметр.

Элементы структуры называются ***полями структуры*** и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если ***список_имен_структур*** отсутствует, то

имя_типа_структуры опускать нельзя и наоборот. После закрывающей фигурной скобки структуры обязательно ставится точка с запятой.

Пример. Структура библиографической карточки.

```
struct card              //card – тип структуры
 {char *author;          //author – Ф.И.О. автора
  char *title;           //title – заголовок книги
  char *city;            //city – место издания
  char *firm;            //firm – издательство
  int  year;             //year – год издания
  int  pages;            //pages – количество страниц
 };                      //список_имен_структур опущен
```

Далее тип структуры можно использовать для определения конкретных объектов.

Например:

```
card rec1, rec2, rec3;    //reci – имена структур типа card
```

Если структура используется однократно, то объекты определяются без имени типа.

Например:

```
struct  {char N[12];
int value;
} XX, XY, EE[8], *pst;
```

Организация программ с использованием структур

Для **обращения** к полям структур используются расширенные (составные, уточненные) имена:

имя_структуры . имя_элемента_структуры

При этом с полями структур можно выполнять любые операции, применимые к типу этих полей.

Например:

```
XY . N[0]=37;          //присваивание поля N[0] структуры XY
```

cin >> EE[3] . value; //ввод значения поля value структуры EE[3]

При определении структур возможна **инициализация**, т.е. задание начальных значений их элементов.

Например:

```
card dictionary=
 {"Подбельский В.В.", "Язык Си ++", "М.",
   "Финансы и статистика", 1999, 560};
```

При инициализации массивов структур каждый элемент массива заключается в фигурные скобки.

Например:

```
struct complex {float real, im;} comple [2] [3]=
{{{1,1}, {1,1},  {1,1}}, {{2,2}, {2,2}, {2,2}}};
```

Для структур одного и того же типа определена операция **присваивания**. При этом происходит поэлементное копирование полей структур.

Например:

```
XY=EE[3]; //предполагается, что структура EE[3] уже
//инициализирована
XX=XY;
```

Структуру можно передавать в функцию и возвращать в качестве значения функции по правилам, определенным для работы с функциями.

Пример. Составить программу, которая позволяет организовывать список в виде массива структур, содержащий информацию о товарах (табл. 7.1), и сортировать этот список:

а) по алфавиту наименования товара;

б) по убыванию цены товара;

в) по убыванию числа единиц товара.

Вывести отсортированный список на печать (лист. 8.1).

Листинг 8.1. ***struktur.cpp***

```
#include <iostream.h>
#include <string.h>   //Подключает функцию сравнения строк strcmp
#include <iomanip.h>
  //Определение структуры
  struct tovar                    //tovar - глобальный тип структуры
  {
   char *name;                    //Наименование товара
   float price;                   //Цена товара
   short kol;                     //Количество товара
  }
//Инициализация массива структур sp[]
sp[]={{"Карандаш",1.50,12},
        {"Ручка",3.40,20},
        {"Линейка",0.85,10},
        {"Ластик",1.00,25},
        {"Тетрадь",1.20,30}};
void out_sp(int,tovar*);      //Прототип функции вывода на печать
void sort(char,int,tovar*);       //Прототип функции сортировки
void main()
  {
    int n=5;
    cout<<"Исходный список товаров:"<<endl;
    out_sp(n,sp);             //Вызов функции вывода
    sort('t',n,sp);           //Сортировка по наименованию товара
    cout<<"Сортировка по наименованию товара (по алфавиту):"<<endl;
    out_sp(n,sp);
    sort('p',n,sp);           //Сортировка по убыванию цены товара
    cout<<"Сортировка по убыванию цены товара:"<<endl;
    out_sp(n,sp);
    sort('k',n,sp);      //Сортировка по убыванию количества товара
    cout<<"Сортировка по убыванию количества товара:"<<endl;
    out_sp(n,sp);
  }
```

```
//Функция вывода списка на печать out_sp
 void out_sp(int m,tovar *tabl)
 {
  for(int i=0;i<m;i++)
  cout<<(i+1)<<". "
      <<setw(8)<<tabl[i].name
      <<setw(6)<<tabl[i].price
      <<setw(4)<<tabl[i].kol<<endl;
 }
 /*Функция сортировки sort:
    при ch='t' - по наименованию товара (по алфавиту)
    при ch='p' - по цене товара
    при ch='k' - по количеству товара*/
 void sort(char ch,int m,tovar *lst)
 { tovar x;
  for(int i=0;i<m-1;i++)
  for(int j=i+1;j<m;j++)
   if((ch=='t' && strcmp(lst[i].name,lst[j].name)>0)||
      (ch=='p' && lst[i].price<lst[j].price)||
      (ch=='k' && lst[i].kol<lst[j].kol))
        {x=lst[i];lst[i]=lst[j];lst[j]=x;
        }
}
```

Результат выполнения программы

Исходный список товаров:

```
1. Карандаш     1.5    12
2.       Ручка     3.4    20
3.    Линейка    0.85   10
4.      Ластик         1   25
5.     Тетрадь     1.2   30
```

Сортировка по наименованию товара (по алфавиту):

```
1. Карандаш      1.5   12
```

```
2.     Ластик        1   25
3.    Линейка     0.85   10
4.      Ручка      3.4   20
5.    Тетрадь      1.2   30
Сортировка по убыванию цены товара:
1.      Ручка      3.4   20
2. Карандаш        1.5   12
3.    Тетрадь      1.2   30
4.     Ластик        1   25
5.    Линейка     0.85   10
Сортировка по убыванию количества товара:
1.    Тетрадь      1.2   30
2.     Ластик        1   25
3.      Ручка      3.4   20
4. Карандаш        1.5   12
5.    Линейка     0.85   10
```

!! *Проанализируйте программу. Создав новый файл проекта с именем struktur.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Упражнения

Составить и отладить программу, которая позволяет организовать список в виде массива структур, содержащий заданную по вариантам в табл.7.2 информацию и обработать этот список заданным образом. Вывести полученные результаты на печать.

Таблица 7.2

Варианты заданий

<table>
<tr><td>1</td><td>Список, содержащий информацию о результатах сдачи экзаменов по математике, физике и химии студентами группы. Обрабатывать этот список следующим образом:
а) сортировка по алфавиту;
б) сортировка по успеваемости;
в) вывести на печать список отличников;
г) вывести на печать список хорошистов;
д) вывести на печать список троечников</td></tr>
<tr><td>2</td><td>Список, содержащий информацию о результатах сдачи экзаменов по математике, физике и химии студентами группы. Обрабатывать этот список следующим образом:
а) сортировка по алфавиту;
б) сортировка по успеваемости;
в) подсчитать средний балл успеваемости группы;
г) вывести на печать список студентов, имеющих средний балл выше среднего балла группы;
д) вывести на печать список студентов, имеющих средний балл ниже среднего балла группы</td></tr>
<tr><td>3</td><td>Список, содержащий информацию о файлах (имя, расширение, размер). Обрабатывать этот список следующим образом:
а) сортировка по имени;
б) сортировка по расширению;
в) сортировка по размеру
Исходные данные
<table><tr><th>№</th><th>Имя</th><th>Расширение</th><th>Размер</th></tr><tr><td>1</td><td>graph</td><td>tpu</td><td>33440</td></tr><tr><td>2</td><td>pascal</td><td>bat</td><td>50</td></tr><tr><td>3</td><td>trip</td><td>clr</td><td>16677</td></tr><tr><td>4</td><td>turbo</td><td>exe</td><td>402474</td></tr><tr><td>5</td><td>turbo</td><td>tph</td><td>933384</td></tr></table></td></tr>
</table>

Продолжение табл. 7.2

4

Список в виде массива структур, содержащий сведения о багаже (название багажа, число вещей, общий вес). Обрабатывать этот список следующим образом:

а) сортировка по алфавиту;

б) сортировка по числу вещей;

в) сортировка по весу

Исходные данные

№	Название багажа	Число вещей	Общий вес, кг
1	Чемодан	10	7,3
2	Сумка	23	15,8
3	Коробка	15	9,45
4	Чемодан	28	20,1
5	Мешок	7	17

5

Список, содержащий информацию о расписании движения поездов (номер поезда, направление, время отправления). Обрабатывать этот список следующим образом:

а) сортировка по номерам поездов; вывести на печать четные и нечетные номера поездов;

б) сортировка по направлению в алфавитном порядке;

в) сортировка по времени отправления

Исходные данные

Номер поезда	Направление		Отправление
	Откуда	Куда	
9	Самара	Москва	17.45
58	Адлер	Новосибирск	14.23
14	Москва	Челябинск	21.30
213	Ульяновск	Самара	10.15
55	Новосибирск	Воронеж	12.00

6

Список, содержащий сведения о книгах (автор, название, место издательства, издательство, год, количество страниц), и обрабатывать этот список следующим образом:

а) сортировка по алфавиту фамилий авторов;

б) сортировка по алфавиту названий произведении;

в) сортировка по алфавиту названий издательств;

г) сортировка по возрастанию количества страниц.

Контрольные вопросы

1.Что такое запись, и в каких случаях применяют данный тип данных?

2.Каким образом описываются в программе переменные типа запись? Приведите примеры.

3.Что такое поле записи? Приведите примеры.

4.Как осуществляется доступ к полям записи?

5.Что такое вложенное поле? Приведите примеры обращения к вложенным полям.

6.Расскажите об особенностях ввода переменных типа запись. Приведите примеры.

7.Какие операции можно применять к переменным типа запись и к их полям? Приведите примеры.

8.Как осуществляется вывод значений переменных типа запись? Приведите примеры.

9.В каких случаях целесообразно использовать оператор присоединения **With**? Приведите примеры.

10. Каков общий вид оператора присоединения? Поясните на примерах механизм действия этого оператора.

11. Что такое структуры и в каких случаях применяют этот тип данных?

12. Какого типа могут быть элементы структуры?

13. Каким образом описываются в программе переменные типа структура? Приведите примеры.

14. Что такое поле структуры? Приведите примеры.

15. Каким образом осуществляется обращение к полям структуры? Приведите пример.

16. Что такое вложенное поле? Приведите примеры обращения к вложенным полям.

17. В каком случае и почему можно опустить в описании структуры имя типа?

18. Какие операции применимы к элементам структур?

19. В каком случае при работе со структурами можно использо-

вать операцию присваивания?

20. Как осуществляется инициализация структур? Приведите пример.

21. Как осуществить изменение элементов массива структуры с использованием функций?

22. Каким образом осуществляется возвращение структуры в качестве значения функции?

8. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ НА ЯЗЫКЕ TURBO PASCAL

Связь с внешними источниками, приемниками и носителями информации в системе TP осуществляется только с помощью файлов.

Традиционно под файлом понимается: либо именованная область внешней памяти (как правило, магнитных дисков), либо устройство, являющееся по своему назначению источником или приемником информации (дисплей, клавиатура, принтер), чаще называемое логическим устройством.

Любой файл имеет имя, которое представляет собой спецификацию по MS DOS, например C:\Files\Baza.dat.

Логическая структура файла в принципе похожа на структуру массива. Различия заключаются в следующем:

1) у массива количество элементов фиксируется в момент распределения памяти, и он целиком располагается в оперативной памяти;

2) у файла количество элементов в процессе работы может изменяться, и он располагается на внешних носителях информации; в конце файла находится специальный символ **Eof** – "конец файла" (это управляющий символ #26).

В TP различают три типа файла: типизированные, которые содержат компоненты одного типа; текстовые; нетипизированные. К элементам всех файлов можно обращаться последовательно, а для типизированных файлов – выборочно (прямой доступ).

При последовательном доступе ввод или считывание компонент ведутся последовательно в порядке возрастания их номеров; прямой доступ предполагает выборочное обращение к конкретным компонентам, которые задаются их номерами. При этом считается, что первой компоненте соответствует нулевой номер.

В программах файлы представляются так называемой файловой переменной, которая в интегрированной среде TP по существу играет роль нового имени файла.

Файловая переменная может быть описана следующими способами.

1.С явным объявлением файлового типа:

```
Type
  <имя_типа_1> = file of <тип компонент>;
{типизированные файлы}
  <имя_типа_2> = file;                    {нетипизированные файлы}
  <имя_типа_3> = text;                          {текстовые файлы}

Var
 <имя файловой переменной> : <имя типа>;
```

2.С неявным объявлением файлового типа:

```
Var
<имя_файловой_переменой_1> : file of <тип компонент>;
<имя_файловой_переменой_2> : file;
<имя_файловой_переменой_3> : text;
```

Например:

```
 Type fib = file of integer;
              student = record name : string [12];
                               age : word;
                        end;
var f : fib;
            gruppa : file of student;
            instr : text;
            myfile : file;
```

С переменной файлового типа связано понятие текущего указателя файла. Его можно понимать как скрытую переменную (т.е. неявно описанную вместе с файловой переменной), которая указывает на некоторый конкретный элемент файла. Как правило, все действия с файлом производятся поэлементно, причем в этих действиях участву-

ет тот элемент файла, который обозначается текущим указателем. При выполнении операции ввода/вывода указатель автоматически перемещается в следующую позицию.

Наибольшее применение находят типизированные файлы, так как они обеспечивают возможность как прямого, так и последовательного доступа к элементам файла и поэтому используются для хранения баз данных.

Работа с типизированными файлами

Работа с файлами заключается в записи и считывании их элементов. Чтобы проводить эти операции, необходимо организовать доступ к файлу на диске. Для этого необходимо *связать* файловую переменную с именем физического файла (по DOS). Это происходит с помощью процедуры **Assign**:

Procedure Assign (var F; name : string);

где **F** – файловая переменная, **name** – полное имя файла на диске в DOS или имя логического устройства (выражение строкового типа).

Например, на диске **a:** в каталоге **MYFILE** имеется файл **data.dat** (a:\MYDIR\data.dat):

```
Var f : file of integer;
  begin
     . . . . . .
     Assign (f,'a:\MYFILE\data.dat');
     . . . . . .
  end.
```

После проведения процедуры связывания требуется произвести подготовку файла к выполнению операций, т.е. *открыть* файл. Это осуществляется с помощью одной из двух процедур модуля System: **Reset**, **Rewrite**.

1. **Procedure Reset (var F);** открывает уже существующий файл, который связан с файловой переменной F. При этом указатель текущей позиции файла устанавливается в начало файла (на компонент с

номером 0), т.е. типизированный файл будет открыт как для чтения, так и для записи.

Если связанный файл не существует, то программа останавливается и на экран выдается сообщение об ошибке. Чтобы избежать аварийной остановки при отсутствии требуемого файла, можно отключить режим автоматического контроля ошибок, задав перед **Reset** директиву компилятора **{$I-}**. В этом случае можно выяснить наличие файла, воспользовавшись стандартной функцией **IOResult** типа word. Если операция завершилась успешно, то **IOResult** = 0, иначе – <>0. После этого необходимо включить режим контроля ошибок, задав директиву компилятора **{$I+}**.

Например:

```
assign(old,S);{связывание файловой переменной old с именем физического файла}
{$I-}                    {отключение контроля ошибок}
reset(old);              {открытие файла}
{$I+}                    {включение контроля ошибок}
if IOResult<>0 then begin   {проверка результата открытия файла}
writeln('Ошибка открытия файла!');
```

2. **Procedure Rewrite(var F);** создает новый физический файл, имя которого уже связано с файловой переменной процедурой **Assign**. Если файл с таким именем уже существует, то содержимое его удаляется, и на его месте создается новый пустой файл. Указатель текущей позиции устанавливается на 0. В результате файл готов для записи информации.

После завершения работы с каждым файлом их рекомендуется закрывать (для надежности сохранения информации). Это осуществляется применением стандартной процедуры **Close**:

Procedure Close(var F);

При этом связь файловой переменной с именем файла, установленная ранее процедурой **Assign** сохраняется. Далее файл можно открывать снова.

Все практические действия с файлами основаны на наборе стандартных процедур и функций модуля **System** и **DOS**. Это не исключает использование собственных подпрограмм.

Для работы с типизированными файлами используются следующие основные встроенные процедуры и функции:

1. **Procedure Read(F, V1 [,V2, …,Vn]);**

где Vi – переменная того же типа, что и тип файла.

Используется для считывания информации из файла, при этом файл должен быть открыт процедурой **Reset** (открывать файл для чтения с помощью процедуры **Rewrite** нельзя, т.к. содержимое файла уничтожится).

2. **Procedure Write(F, V1 [,V2, …,Vn]);** используется для записи переменной в файл.

Особенностью процедуры **Write** является возможность ее применения к типизированным файлам, открытым не только для записи процедурой **Rewrite**, но и для считывания посредством процедуры **Reset**. Это позволяет модифицировать содержимое файла, не закрывая его, попеременно обращаясь к **Read** и **Write**.

3. **Procedure Seek(var F, N:Longint);** перемещает текущую позицию (указатель) к заданному элементу с номером N (при счете от 0).

4. **Function FileSize(var F):Longint;** возвращает текущий размер файла. Если файл пуст, то возвращает 0.

Эту функцию можно использовать для перемещения указателя в конец файла (на EOF) с помощью процедуры **Seek**:

Seek(F, FileSize(F));

5. **Function FilePos(Var F):Longint;** возвращает текущую позицию в файле, т.е. определяет номер текущей компоненты (указателя), считая от нуля.

6. **Procedure Truncate(Var F);** усекает размер файла до текущей позиции, все элементы после нее удаляются, и текущая позиция становится концом файла.

7. **Procedure Rename(var F; NewName:string);** переименовывает внешний файл любого типа. Процедура применяется к уже связанным файловым переменным F, но для которых еще не выполнена процедура открытия.

Например: Assign (NewFile, 'a:\MyFile\Data.dat');
Rename (NewFile, 'a:\MyFilt\Mem.txt');

8. **Function EOF(var F):Boolean;** возвращает для файла F признак конца файла, т.е. **EOF = true**, если указатель стоит за последним элементом файла или файл не содержит никаких элементов.

Функция **EOF** часто используется в циклах.

Например:

```
. . . . While not EOF(FileName) do
        begin  Read(FileName, X);
. . . . . . . . . . . . . .
        end; . . . . . . . . . .
```

9. **Procedure Erase(var F);** удаляет внешний файл, связанный с переменной F. Используется только для закрытых файлов.

Типизированные константы не могут принимать значения файловых типов; также нельзя объявить типизированную константу-запись, если хотя бы одно из ее полей является полем файлового типа.

Пример. Составить программу, которая обеспечивает:

1. Создание внешнего файла **C:\baza.dat**, состоящего из записей типа **Student**, в которых содержится информация о студентах: фамилия и инициалы, дата рождения, адрес, номер группы, оценки по физике и математике, полученные на вступительных экзаменах.

2. Сортировку списка по алфавиту.

3. Выбор из списка всех студентов, родившихся раньше 1985 года и занесение их в новый файл типа Student (файл на диске назвать своей фамилией).

4. Вывод на печать всех данных о студентах из нового файла.

```
Program files;
uses printer;
type
  birthday = record              {дата рождения}
              day: 1..31;
              month: 1..12;
              year: 1975..1985;
             end;
  adress = record                {домашний адрес студента}
            gor:string[20];      {город}
            ul:string[20];       {улица}
            dom:integer;         {№ дома}
            kv:integer;          {№ квартиры}
           end;
  ball = record                  {Вступительные оценки}
          mat: real;             {по математике}
          phis: real;            {по физике}
         end;
  student = record
             FIO:string[20];       {фамилия и инициалы}
             date:birthday;        {дата рождения}
             adr:adress;           {адрес}
             grup:string[6];       {группа}
             exam:ball;            {оценки}
            end;
var
  baz,new:file of student;   {baz-базовый файл; new-свой файл}
  x,y:student; i,j,n,m:integer;
```

```
  c:char; S:string[40];
begin
writeln('Введите полное имя базового файла');
readln(S);                    {ввод полного имени файла}
assign(baz,S);    {связывание файловой переменной baz с именем физического файла}
rewrite(baz);                 {открытие файла для записи}
writeln('Введите новые данные');
{ввод новых данных в оперативную память}
repeat
 writeln('Введите данные о студенте');
  write('Фамилия, имя, отчество: ');readln(x.fio);
  write('Число: ');readln(x.date.day);
  write('Месяц: ');readln(x.date.month);
  write('Год: ');readln(x.date.year);
  write('Город: ');readln(x.adr.gor);
  write('Улица: ');readln(x.adr.ul);
  write('№ дома: ');readln(x.adr.dom);
  write('№ квартиры: ');readln(x.adr.kv);
  write('Группа: ');readln(x.grup);
  write('Математика: ');readln(x.exam.mat);
  write('Физика: ');readln(x.exam.phis);
  write(baz,x);     {запись переменной x в файл baz}
 writeln('Продолжить ввод? Да(Д)/Нет(Н)');readln(c);
 while not (c in ['д','Д','н','Н']) do
        begin writeln('Введите Д или Н');
            readln(c);
        end
 until (c='Н')or(c='н');
{сортировка по алфавиту содержимого файла методом "пузырька"}
L2: for i:=filesize(baz)-1 downto 1 do
   for j:=0 to i-1 do
```

```
    begin
      seek(baz,j);       {установка указателя на элемент с номером j}
      read(baz,x,y);       {считывание из файла двух элементов}
      if x.fio>y.fio then
      begin
        seek(baz,j);
        write(baz,y,x);              {запись в файл считанных элементов
                                     в обратном порядке}
      end;
    end;
    seek(baz,0);           {установка указателя в начало файла}

    {связывание файловой переменной new с вновь создаваемым
файлом}
    assign(new,'c:\Files\Ivanov');
    rewrite(new);              {открытие нового файла для записи}
    while not eof(baz) do
    begin read(baz,x);    {считывание переменной из базового файла}
      if x.date.year<1986 then write(new,x);{выбор студентов
                              и запись данных о них в новый файл}
      end;
    close(baz);                  {закрытие базового файла}
    seek(new,0);           {установка указателя в начало файла new}

    {считывание данных из своего файла и вывод их на экран}
    i:=1;
    while not eof(new) do
    begin
      read(new,x);
    writeln(i,'. ',x.fio:16,'  ',x.date.day:2,'.',x.date.month:2,'.',
          x.date.year,        ' ',x.adr.gor:10,' ',x.adr.ul,
         ' ',x.adr.dom,'-',x.adr.kv);
    writeln('    ',x.grup,' Математика: ',x.exam.mat:4:1,
```

```
            '  Физика: ',x.exam.phis:4:1);
    writeln;
    i:=i+1;
    end;
    close(new);
    end.
```

!! *Проанализируйте программу files, введите ее текст в компьютер, откомпилируйте ее, запустите на счет, введите запрашиваемые данные и просмотрите результаты.*

Упражнения

1. Составить программу, которая обеспечивает:

- пополнение базы данных, находящихся во внешнем файле **C:\baza.dat**, созданном при выполнении программы **files**;
- сортировку списка в алфавитном порядке по названию улицы;
- выбор из списка всех студентов, живущих в домах с нечетным номером, и занесение их в свой файл;
- вывод на печать их фамилий и адресов.

2. Составить программу, которая обеспечивает:

- пополнение базы данных, находящихся во внешнем файле **C:\baza.dat**, созданном при выполнении программы **files**;
- сортировку списка по алфавиту;
- выбор из списка всех студентов, родившихся зимой и занесение их в один файл, а всех студентов, родившихся осенью – в другой файл;
- вывод на печать их фамилий и дат рождения сначала "осенних" студентов, а затем – "зимних".

3. Составить программу, которая обеспечивает:

- пополнение базы данных, находящихся во внешнем файле **C:\baza.dat**, созданном при выполнении программы **files**;
- сортировку списка по успеваемости (по убыванию);

- занесение отсортированного списка в свой файл, затем удаление из списка всех данных о студентах, имеющих суммарный балл ниже проходного (усечение файла);

- вывод на печать их фамилий, номеров групп и результатов вступительных экзаменов.

Контрольные вопросы

1. Что понимается под файлом в TP и каковы характерные особенности файлов?

2. В чем заключается сходство и различие файлов и массивов?

3. Каким образом описываются в программе файловые переменные? Приведите примеры.

4. В чем разница между файловой переменной и именем физического файла?

5. Расскажите о назначении процедуры Assign. Приведите примеры.

6. С помощью каких процедур осуществляется подготовка файлов к работе?

7. В каких случаях применяется процедура Reset? Приведите примеры.

8. Расскажите об особенностях применения стандартной функции IOResult.

9. Для чего предназначена процедура Rewrite? Приведите примеры.

10. Почему рекомендуется закрывать файл после работы с ним? Какая процедура обеспечивает закрытие файла? Приведите примеры.

11. Какие стандартные подпрограммы работы с типизированными файлами вам известны? Расскажите кратко о них.

12. Что такое текущий указатель файла? Поясните назначение функции EOF.

13. В чем отличие типизированных файлов от текстовых?

14. Как осуществляется считывание информации из файла? Приведите примеры.

15. Каким образом информация заносится в файл? Приведите примеры.

16. Поясните на примерах действие процедур Seek и Truncate.

17. Приведите примеры использования функций FileSize и FilePos.

9. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МОДУЛЯ CRT НА ЯЗЫКЕ TURBO PASCAL

Использование модуля CRT увеличивает возможности текстового ввода-вывода. В нем собраны функции и процедуры, которые позволяют управлять:

1) клавиатурой;
2) экраном;
3) звуковым динамиком.

Подпрограммы модуля CRT дополняют основные возможности процедур **Write** и **Read** из модуля **System**.

Основные отличия модуля CRT от модуля System:

1) отслеживание нажатия специальных клавиш и их комбинаций с другими клавишами;
2) возможность ввода символьной информации без эхо-повтора на экране монитора;
3) управление цветом фона и символов;
4) возможность организации окон;
5) вывод информации в произвольную позицию окна.

Для использования функций и процедур модуля CRT его необходимо указать в **uses**-предложении основной программы.

9.1. Работа с клавиатурой

Все клавиши клавиатуры можно разделить на три группы:

1. Клавиши и комбинации клавиш, нажатие которых посылает в буфер клавиатуры код ASCII (алфавитно-цифровые и специальные символы в обоих регистрах (с Shift и без него, или CapsLock), Tab, BackSpace, Enter, Esc). Часто эти клавиши называют *основным набором*. Коды печатаемых символов представлены в табл. 9.1 (см. гл. 9 "Обработка символьной информации"), коды непечатаемых символов приведены в табл. 12.1.

2. Клавиши и комбинации клавиш, нажатие которых посылает в буфер клавиатуры *расширенный код*:

а) функциональные клавиши **F1÷F12**, нажатые как самостоятельно, так и с **Shift**, **Alt**, **Ctrl**;

б) буквенно-цифровые в комбинации с **Alt**;

в) клавиши редактирования и управления курсором.

3. Клавиши и комбинации клавиш, нажатие которых не посылает в буфер клавиатуры никаких кодов. К таким клавишам относятся клавиши регистров **Shift**, **Ctrl**, **Alt**, **CapsLock**, **NumLock**, **ScrollLock**.

Таблица 9.1

Непечатаемые символы таблицы ASCII

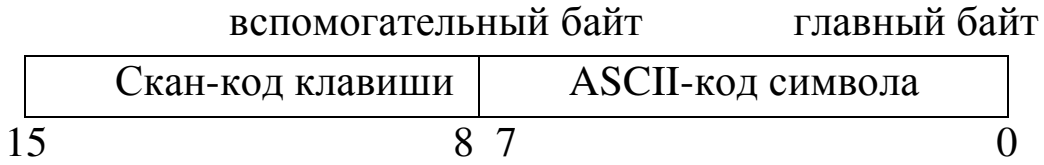
16-ый код	Комбинация клавиш (клавиша)	16-ый код	Комбинация клавиш (клавиша)	16-ый код	Комбинация клавиш (клавиша)
00	Ctrl + 2	0B	Ctrl + K	16	Ctrl + V
01	Ctrl + A	0C	Ctrl + L	17	Ctrl + W
02	Ctrl + B	0D	Ctrl + M (Enter)	18	Ctrl + X
03	Ctrl + C	0E	Ctrl + N	19	Ctrl + Y
04	Ctrl + D	0F	Ctrl + O	1A	Ctrl + Z
05	Ctrl + E	10	Ctrl + P	1B	Ctrl + [(Esc)
06	Ctrl + F	11	Ctrl + Q	1C	Ctrl + \
07	Ctrl + G	12	Ctrl + R	1D	Ctrl +]
08	Ctrl + H (BackSpace)	13	Ctrl + S	1E	Ctrl + 6
09	Ctrl + I (Tab)	14	Ctrl + T	1F	Ctrl + -
0A	Ctrl + J	15	Ctrl + U		

Нажатие любой клавиши первоначально обрабатывается встроенным процессором самой клавиатуры, который в результате сканирования клавиатуры помещает код клавиши (скан-код) в собственный буфер. По существу скан-код – это порядковый номер клавиши в со-

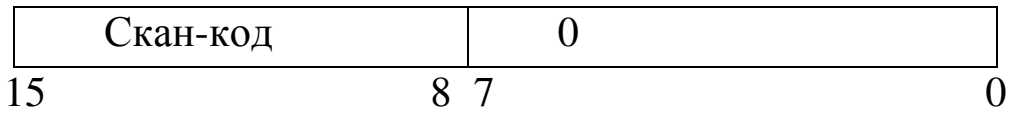
ответствии с ее геометрическим расположением на панели клавиатуры (табл. 9.2).

Программа обработки прерывания выбирает скан-коды из аппаратного буфера клавиатуры, преобразует их в значения основного или расширенного набора и помещает их в буфер BIOS (программный буфер клавиатуры или просто буфер) в виде двухбайтового кода:

а) для основного набора



б) для расширенного набора



Например:

2	33	восклицательный знак **!**
71	0	клавиша Home

Таблица 9.2

Скан-коды клавиатуры

Клавиша	Скан-код в 16-й СС	Клавиша	Скан-код в 16-й СС	Клавиша	Скан-код в 16-ой СС	Клавиша	Скан-код в 16-й СС
Esc	01	U	16	| \	2B	F6	40
! 1	02	I	17	Z	2C	F7	41
@ 2	03	O	18	X	2D	F8	42
# 3	04	P	19	C	2E	F9	43
$ 4	05	{ [	1A	V	2F	F10	44
% 5	06	}]	1B	B	30	Num-Lock	45
^ 6	07	Enter	1C	N	31	F11	D9
& 7	08	Ctrl	1D	M	32	F12	DA

Окончание табл. 9.2

Клави-ша	Скан-код в 16-й СС	Клави-ша	Скан-код в 16-й СС	Клавиша	Скан-код в 16-ой СС	Клавиша	Скан-код в 16-й СС
* 8	09	A	1E	< ,	33	Home	47
(9	0A	S	1F	> .	34	↑	48
) 0	0B	D	20	? /	35	PageUp	49
_ -	0C	F	21	правый Shift	36	серый -	4A
+ =	0D	G	22	Print-Screen	37	←	4B
Back-Space	0E	H	23	Alt	38	5	4C
Tab	0F	J	24	Пробел	39	→	4D
Q	10	K	25	CapsLock	3A	серый +	4E
W	11	L	26	F1	3B	End	4F
E	12	: ;	27	F2	3C	↓	50
R	13	" '	28	F3	3D	PageDown	51
T	14	~ `	29	F4	3E	Insert	52
Y	15	левый Shift	2A	F5	3F	Delete	53

В расширенных кодах первый байт (служебный байт) всегда равен нулю, а второй байт (информационный байт) представляет собой скан-код (табл. 9.3).

Для опроса содержимого буфера в модуле CRT применяются функции **KeyPressed** и **ReadKey**. Первая функция имеет описание следующего вида:

Function KeyPressed : Boolean;

Она возвращает True, если в буфере содержится хотя бы один символ, и False, если буфер пуст.

Таблица 9.3

Информационные байты расширенных кодов клавиатуры

Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код
Alt + Q	10	F1	3B	Shift + F7	5A	Ctrl + →	74
Alt + W	11	F2	3C	Shift + F8	5B	Ctrl + End	75
Alt + E	12	F3	3D	Shift + F9	5C	Ctrl + PageUp	76
Alt + R	13	F4	3E	Shift + F10	5D	Ctrl + Home	77
Alt + T	14	F5	3F	Ctrl + F1	5E	Alt + 1	78
Alt + Y	15	F6	40	Ctrl + F2	5F	Alt + 2	79
Alt + U	16	F7	41	Ctrl + F3	60	Alt + 3	7A
Alt + I	17	F8	42	Ctrl + F4	61	Alt + 4	7B
Alt + O	18	F9	43	Ctrl + F5	62	Alt + 5	7C
Alt + P	19	F10	44	Ctrl + F6	63	Alt + 6	7D
Alt + A	1E	Home	47	Ctrl + F7	64	Alt + 7	7E
Alt + S	1F	↑	48	Ctrl + F8	65	Alt + 8	7F
Alt + D	20	PageUp	49	Ctrl + F9	66	Alt + 9	80
Alt + F	21	←	4A	Ctrl + F10	67	Alt + 0	81
Alt + G	22	→	4D	Alt + F1	68	Alt + -	82
Alt + H	23	End	4F	Alt + F2	69	Alt + =	83
Alt + J	24	↓	50	Alt + F3	6A	Ctrl + PageDown	84
Alt + K	25	PageDown	51	Alt + F4	6B	F11	85
Alt + L	26	Insert	52	Alt + F5	6C	F12	86
Alt + Z	2C	Delete	53	Alt + F6	6D	Shift + F11	87
Alt + X	2D	Shift + F1	54	Alt + F7	6E	Shift + F12	88
Alt + C	2E	Shift + F2	55	Alt + F8	6F	Ctrl + F11	89

Окончание табл. 9.3

Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код	Клавиша, комбинация клавиш	Код
Alt + V	2F	Shift + F3	56	Alt + F9	70	Ctrl + F12	8A
Alt + B	30	Shift + F4	57	Alt + F10	71	Alt + F11	8B
Alt + N	31	Shift + F5	58	Ctrl + Prt-Scr	72	Alt + F12	8C
Alt + M	32	Shift + F6	59	Ctrl + ←	73		

Наиболее часто функция **KeyPressed** используется в циклах ожидания нажатия на любую клавишу, в частности в виде конструкций:

а) **Repeat until KeyPressed;**

б) **While not Keypressed do;**

В обоих случаях процессор зацикливается, выполняя пустой оператор до нажатия любой клавиши. Если буфер содержит хотя бы один код (была нажата клавиша, т.е. KeyPressed = True), то цикл завершается и управление передается следующему за ним оператору.

Для корректного использования цикла ожидания буфер клавиатуры необходимо предварительно очищать от кодов случайно или ранее нажатых клавиш. Для этого используется функция **ReadKey**. Она описывается следующим образом:

Function ReadKey : char;

Эта функция извлекает код символа из буфера и возвращает его в программу без эхо-повтора символа на экран. Буфер организован в виде очереди по принципу "первым пришел - первым ушел" и рассчитан на хранение до 15 кодов.

Если буфер пуст, то функция **ReadKey** приостанавливает выполнение программы до нажатия на любую клавишу кроме Shift, Ctrl, Alt, CapsLock, NumLock, ScrollLock, т.е. позволяет отслеживать нажатие более широкого множества клавиш по сравнению с **Read/ReadLn**.

Применение **ReadKey** позволяет очистить буфер и в результате цикл ожидания нажатия клавиши становится надежным. Например, предыдущий цикл ожидания нажатия на любую клавишу при использовании **ReadKey** можно записать в следующем виде:

```
Var ch : char;
  Begin . . . . . .
 while KeyPressed do ch := ReadKey;   {очистка буфера}
 repeat until KeyPressed;             {ожидание нажатия}
            . . . . . .
End.
```

Каждое обращение к функции **ReadKey** извлекает один содержащийся там код. Если в двухбайтовом коде главный байт совпадает с ASCII, **ReadKey** возвращает в программу только этот код, а скан-код отбрасывает. Если главный байт равен нулю, то после его считывания необходимо еще раз обратиться к **ReadKey** для считывания вспомогательного байта расширенного кода. Например:

```
while KeyPressed do ch := ReadKey;    {очистка буфера}

ch := ReadKey;                  {ожидание ввода и считывание
                                кода при нажатии клавиши}
if ch =# 0 then begin ch := ReadKey;   {прием расширенного кода}
                  . . . . . .
          end;
```

Далее ch анализируется обычно с помощью условного оператора **if** или оператора **case of**.

9.2. Работа с экраном

Основным текстовым режимом работы является CO80 (размер экрана 80×25), который устанавливается по умолчанию.

Координаты экрана имеют вид, представленный на рис. 9.1.

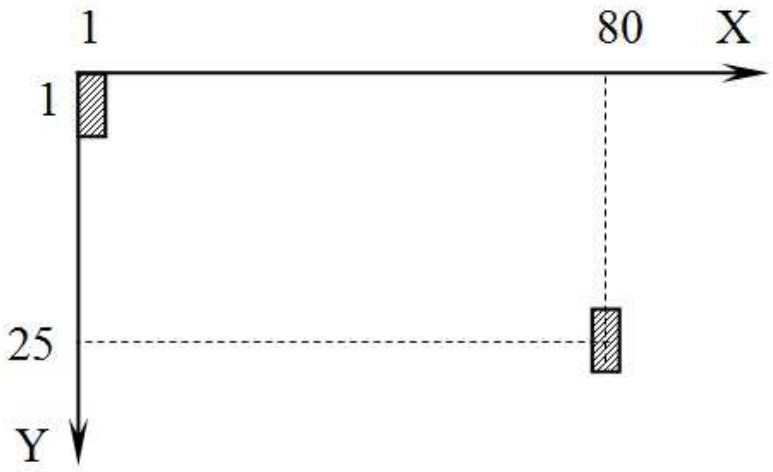


Рис. 9.1. Координаты экрана
в текстовом режиме

Началом координат является точка (1,1). Очередной вывод символа на экран начинается с текущей позиции курсора.

Координаты курсора можно определить с помощью функций

Function WhereX : byte; - координату X;

Function WhereY : byte; - координату Y.

Курсор можно перемещать в любую позицию экрана с помощью процедуры **GoToXY**:

Procedure GoToXY (X, Y : byte);

Если задаются недопустимые координаты (X, Y), то обращение к процедуре игнорируется.

Модуль CRT позволяет создавать окна, которые, в частности, используются для отображения текстовой информации и формирования меню.

Основной подпрограммой организации окна является процедура

Procedure Window (X1, Y1, X2, Y2 : byte);

где (X1,Y1) и (X2,Y2) - соответственно координаты верхнего левого и нижнего правого углов окна.

После выполнения процедуры **Window** курсор первоначально устанавливается в позицию (1,1) созданного окна. Все последующие выводы информации процедурами **Write/WriteLn** осуществляются в пределах этого окна.

Текущее окно отменяется введением нового окна, которое становится текущим. Например, для того чтобы вернуться к работе в пределах всего экрана, необходимо подать команду **Window (1,1,80,25)**. Вновь созданное окно может перекрывать ранее созданные окна.

При необходимости можно задать цвет фона с помощью процедуры

Procedure TextBackGround (Color : byte);

а цвет символов - применением процедуры

Procedure TextColor (Color : byte);

где Color - константа цвета фона.

Новые установки цвета не влияют на ранее выведенный текст.

В текстовых режимах цвет символов и фонов определяется константами, значения которых представлены в табл. 9.4.

Таблица 9.4

Константы цвета

Константа	Значение	Цвет	Константа	Значение	Цвет
Black	0	Черный	DarkGray	8	Темно-серый
Blue	1	Синий	LightBlue	9	Голубой
Green	2	Зеленый	LightGreen	10	Светло-зеленый
Cyan	3	Бирюзовый	LightCyan	11	Светло-бирюзовый
Red	4	Красный	LightRed	12	Светло-красный
Magenta	5	Малиновый	LightMagenta	13	Светло-малиновый
Brown	6	Коричневый	Yellow	14	Желтый
LightGray	7	Светло-серый	White	15	Белый

При этом цвет фона ограничен значениями Color = 0..7.

Если к константе цвета в процедуре **TextColor** прибавить константу Blink = 128, то выводимые символы будут мерцать.

Для очистки окна (экрана) используется процедура **ClrScr** (от англ. Clear Screen):

Procedure ClrScr;

Она очищает окно и помещает курсор в точку (1,1).

Для работы со строками в CRT применяются процедуры **ClrEol**, **InsLine** и **DelLine**:

а) **Procedure ClrEol;** - стирает все символы, начиная от позиции курсора до конца строки;

б) **Procedure DelLine;** - удаляет строку, на которой находится курсор; при этом все строки, расположенные ниже, перемещаются на одну строку вверх, а внизу экрана появляется стока, закрашенная цветом фона;

в) **Procedure InsLine;** - вставляет пустую строку в позицию курсора; при этом все строки смещаются вниз на одну строку, а самая нижняя исчезает с экрана.

9.3. Управление звуком динамика

В CRT имеются две процедуры, предназначенные для управления встроенным динамиком:

1) **Procedure Sound (Herz : word);** - включает динамик с частотой звука Herz (в Гц);

2) **Procedure NoSound;** - выключает динамик.

Выключение звука - обязательная часть программы, так как звучание будет продолжаться и после окончания выполнения программы или ее остановки.

Для задания длительности звучания используется процедура **Delay**:

Procedure Delay (Msec : word); - останавливает выполнение программы на время **Msec** (в миллисекундах).

Эта процедура может также применяться в программах для задержки вывода информации на экран.

Пример. Составить программу, которая обеспечивает:

- расчет табулированных значений функции $z = \frac{xy^4}{4} + \frac{yx^4}{4}$

  при x = 1..1.4, h_x= 0.2, y = 2..2.2, h_y= 0.1.
- вывод исходных данных и полученных результатов в разные окна:
  - а) окно ввода – в левую верхнюю часть экрана, цвет фона – зеленый, цвет символов – черный;
  - б) окно вывода – в правую нижнюю часть экрана, цвет фона – малиновый, цвет символов – светло-серый;
- два останова:
  - а) перед выводом результатов с диалоговым предложением в дополнительном окне внизу экрана: **"Для вывода результатов нажмите любую клавишу"**;
  - б) после вывода результатов с диалоговым предложением в дополнительном окне: **"Для выхода из программы нажмите любую клавишу"**.

Текст программы

```
Program Tab_Crt;
uses CRT;
var X0,Xn,Y0,Yn,Hx,Hy:real;
   X,Y:array[1..20]of real;
   Z:array[1..20,1..20]of real;
   i,j,Nx,Ny:byte;
   ch:char;
Begin
{Восстановление начальных параметров окна}
window(1,1,80,80);Textbackground(0); textcolor(7);clrscr;
{Организация окна ввода}
```

```
window(2,2,33,9);Textbackground(5); textcolor(7);clrscr;
{Ввод исходных данных}
 writeln('Исходные данные');
 write('1. Начальное значение x: X0=');readln(X0);
 write('2. Конечное значение x: Xn='); readln(Xn);
 write('3. Шаг изменения x: Hx='); readln(Hx);
 write('4. Начальное значение y: Y0=');readln(Y0);
 write('5. Конечное значение y: Yn=');readln(Yn);
 write('6. Шаг изменения y: Hy=');readln(Hy);
{Заполнение массивов x и y значениями}
Nx:=Round((Xn-X0)/Hx)+1;
Ny:=Round((Yn-Y0)/Hy)+1;
x[1]:=X0;
For i:=2 to Nx do x[i]:=x[i-1]+Hx;
y[1]:=Y0;
for j:=2 to Ny do y[j]:=y[j-1]+Hy;
{Организация первого окна останова}
window(4,17,35,18);Textbackground(4);
textcolor(15+128);clrscr;
writeln('       Для продолжения');
write('     нажмите любую клавишу!');
  while keypressed do ch:=ReadKey;
  Repeat until KeyPressed;
{Организация окна вывода}
window(48,11,78,21);Textbackground(7); textcolor(4);clrscr;
{Вывод полученных результатов}
writeln('Полученные результаты:');
For i:=1 to Nx do
for j:=1 to Ny do
begin
z[i,j]:=x[i]*exp(4*ln(y[j]))/4+y[j]*exp(4*ln(x[i]))/4;
writeln('X=',x[i]:5:2,'    Y=',y[j]:5:2,'    Z=', z[i,j]:6:2);
end;
```

```
{Организация второго окна останова}
window(4,17,35,18);Textbackground(4); textcolor(15+128);clrscr;
writeln('    Для выхода из программы');
 write('     нажмите любую клавишу!');
  while keypressed do ch:=ReadKey;
  Repeat until KeyPressed;
end.
```

!! *Проанализируйте программу Tub_Crt, введите ее текст в компьютер, откомпилируйте ее, запустите на счет, введите запрашиваемые данные и просмотрите результаты.*

Упражнения

Составить программу, которая обеспечивает расчет табулированных значений функции z(x,y) и вывод исходных данных и полученных результатов в разные окна. Предусмотреть в программе два ***останова***:

а) перед выводом результатов с диалоговым предложением в дополнительном окне внизу экрана: **"Для вывода результатов нажмите любую клавишу"**;

б) после вывода результатов с диалоговым предложением в дополнительном окне: **"Для выхода нажмите любую клавишу"**.

Задания представлены в виде таблиц. Здесь приведены варианты заданий (табл. 9.5), исходных данных (табл. 9.6), требования к расположению окон (табл. 9.7), цвету фона и символов (табл. 9.8).

Таблица 9.5

Варианты заданий

Вариант	Функция	Окна (табл. 9.4)		Цвета (табл. 9.5)	
		ввода	вывода	окно ввода	окно вывода
1	$z = \ln \frac{\sqrt{x^2 + y^2} - b}{\sqrt{x} + \sin y + b}$	а	г	3	2
2	$z = \frac{e^{xy} + e^{-xy}}{xy}$	б	г	4	5
3	$z = \frac{xy^4}{x} + \frac{yx^4}{y}$	в	г	5	1
4	$z = \left(y + \frac{1}{x} \right)^y$	а	в	2	4
5	$z = \frac{x}{1 + y^x}$	б	д	3	1

Таблица 9.6

Исходные данные

№	X_0	X_n	H_x	Y_0	Y_m	H_y
1	1	1,4	0,2	2	2,2	0,1
2	1,2	1,5	0,1	1,8	2	0,2
3	0,3	1,2	0,3	2,15	2,65	0,25
4	2,5	2,8	0,15	1	1,9	0,3
5	1	4	1	2	3	0,5

Таблица 9.7

Расположение окон ввода и вывода

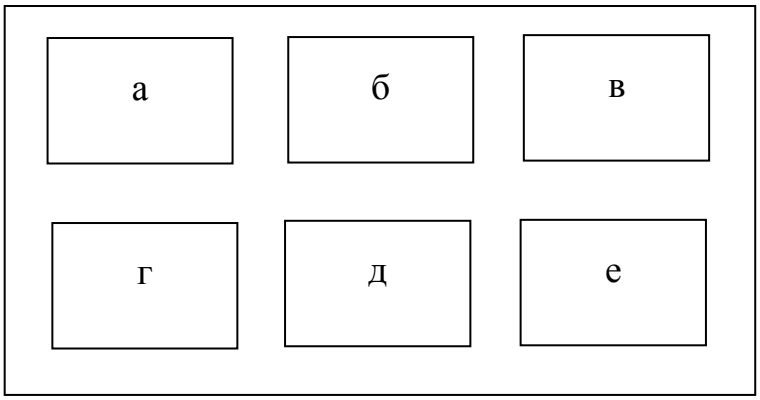


Таблица 9.8

Цвета фона и символов окон

№	Цвет фона	Цвет символа
1	Малиновый	Голубой
2	Зеленый	Темно-серый
3	Бирюзовый	Синий
4	Коричневый	Светло-зеленый
5	Светло-серый	Красный

Контрольные вопросы

1. На какие группы можно разделить клавиши клавиатуры?
2. Какой вид имеет цикл ожидания нажатия любой клавиши?
3. Какой вид имеет цикл очистки буфера клавиатуры?
4. Какие особенности обработки расширенных кодов клавиатуры?
5. Какие процедуры используются для создания звуковых эффектов?
6. Какой текстовый режим устанавливается по умолчанию?
7. Каким образом можно установить мерцание выводимых на экран символов?
8. С помощью какой процедуры осуществляется перемещение курсора?
9. С помощью каких функций определяются координаты курсора?
10. Каким образом можно вернуться из окна к работе в пределах всего экрана?
11. С помощью каких процедур устанавливается цвет фона и выводимых символов?
12. Какие процедуры используются для работы со строками для программного редактирования информации, выводимой на экран?

13. Какие координаты имеют угловые точки экрана в текстовом режиме?

14. Какая процедура предназначена для очистки экрана в текстовом режиме?

15. Какие значения могут принимать константы цвета для символов и для фона в текстовом режиме?

16. Как установить окно для вывода информации в текстовом режиме?

17. Какой размер окна устанавливается по умолчанию?

18. Какое действие выполняет процедура Delay?

10. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МОДУЛЯ GRAPH НА ЯЗЫКЕ TURBO PASCAL

Модуль **Graph** предназначен для создания графических изображений. Он представляет собой библиотеку функций и процедур и подключается к основной программе стандартным образом, т.е. через uses-строку: **Uses** Graph;

10.1. Инициализация и завершение графического режима

Для переключения ПК в графический режим используется процедура **InitGraph**:

Procedure InitGraph (var GrDriver : integer; var Mode : integer; Path : string);

где **GrDriver** – переменная, задающая графический драйвер;

Mode – переменная, задающая графический режим;

Path – переменная, задающая путь к каталогу, где находится используемый драйвер адаптера монитора.

Наиболее простой способ выбора графического драйвера – автоматический. Для этого перед обращением к **InitGraph** полагают **GrDriver := Detect**, где **Detect** – константа. При использовании самого распространенного адаптера типа SVGA **Detect = 0**. Значение переменной **Mode** в автоматическом режиме игнорируется. Драйвер этого адаптера **egavga.bgi** целесообразно помещать в рабочий каталог (например в MYFILE), тогда путь можно задать просто в виде пустой строки **Path := ’’**. Следует отметить, что обе переменные **GrDriver** и **Mode** требуется обязательно описывать.

Для выхода из графического режима используется процедура **CloseGraph**:

Procedure CloseGraph;

Она закрывает графический режим и автоматически возвращает режим, установленный до **InitGraph**.

10.2. Установка цвета

Драйвер **egavga**.bgi позволяет использовать 16 цветов, стандартный набор которых закодирован константами, используемыми также в текстовом режиме (см. табл. 9.1).

Цвет выводимых на экран линий и символов можно задать процедурой **SetColor**:

Procedure SetColor (Color : word);

где **Color** - константа цвета.

Если SetColor не вызвана, то используется белый цвет.

Цвет фона для всего экрана устанавливается процедурой **SetBkColor**:

Procedure SetBkColor (Color : word);

где **Color** - константа цвета фона.

В отличие от текстового видеорежима в графическом режиме цвет фона может быть ярким, т.е. Color = 0..15. Установка нового фона немедленно изменяет цвет графического экрана. Если процедура установки цвета фона после инициализации графического режима не вызвана, то экран будет черным.

Примечание. Цвет символов и линий с номером Color = 0 отождествляется с цветом фона, поэтому обычно он не используется, так как получается невидимое изображение.

10.3. Стиль заполнения

Для того, чтобы выводимое изображение было более красочным и эффектным, используют специальные шаблоны заливки, или стиль заполнения. По существу это комбинация узора и цвета.

Для установки стиля заполнения используется процедура:

Procedure SetFillStyle (Pattern, Color : word);

где **Color** – цвет узора;
Pattern – один из 12 шаблонов в соответствии с рис. 10.1.

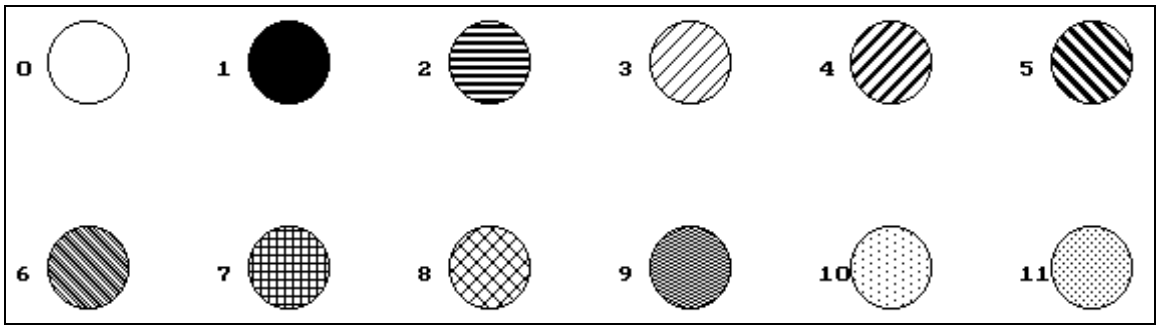


Рис. 10.1. Шаблоны узоров

Закраска областей осуществляется процедурой **FloodFill**:

Procedure FloodFill (X,Y : integer; Border : word);

где (**X,Y**) – координаты точки внутри или вне замкнутой фигуры, т.е. заполняемой цветом области;

Border – цвет граничной линии закрашиваемой области.

В результате заполнение области осуществится стилем, заданным процедурой **SetFillStyle**. Если фигура незамкнута, то заполнение “разольется” по всему экрану.

10.4. Экран, окно, графический указатель

Координаты экрана имеют вид, представленный на рис. 10.2.

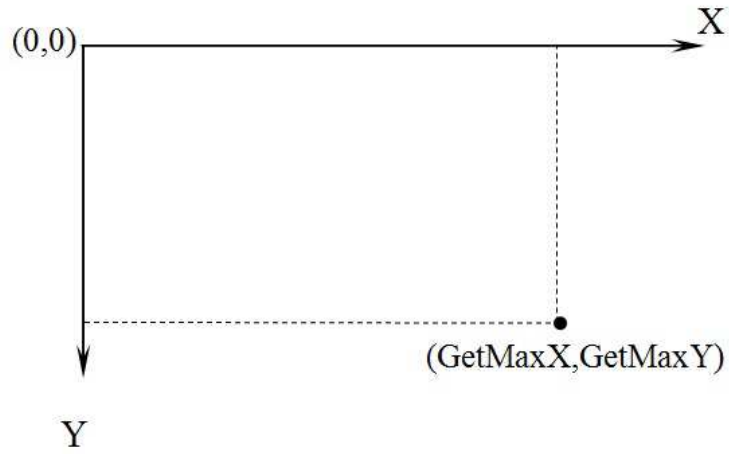


Рис. 10.2. Координаты экрана

Началом координат является точка (0,0).

Максимальные значения координат X и Y определяются соответственно с помощью **GetMaxX** и **GetMaxY** .

Поэтому координаты средней части экрана можно задать точкой **(GetMaxX div 2 , GetMaxY div 2)**

Роль курсора в графическом режиме выполняет *указатель* текущей позиции на экране. В отличие от текстового курсора он невидим.

Для перемещения указателя (без вывода изображения на экран) используются две процедуры:

1) **Procedure MoveTo (X,Y : integer); –** перемещает указатель в точку (X,Y);

2) **Procedure MoveRel (Dx,Dy : integer); –** перемещает указатель в точку, определяемую приращением Dx по координате X и Dy - по координате Y;

Часто бывает полезно работать не со всем экраном, а с отдельным его окном. Окно в графическом режиме – это прямоугольная область для вывода изображения, т.е. визуальный порт. Установка окна осуществляется следующей процедурой:

Procedure SetViewPort (X1,Y1,X2,Y2 : word; Clip : boolean);

Если **Clip = true**, то изображение за границами окна обрезается; если **Clip = false**, то изображение выводится и за указанными границами окна.

При обращении к процедуре **SetViewPort** удобно использовать константы **ClipOn =** true и **ClipOff =** false, которые подставляют в процедуру вместо **Clip**.

Процедуры отмены текущего окна не предусмотрено. Если требуется вернуться к работе с полным экраном, то вызывается процедура **SetViewPort** с соответствующими параметрами:

SetViewPort (0,0,GetMaxX,GetMaxY,ClipOn);

Для очистки графического окна используется процедура **ClearViewPotr**:

Procedure ClearViewPort;

В результате изображение в окне стирается, цвет фона остается без изменения, указатель перемещается в левый верхний угол окна, в позицию (0,0).

Для очистки графического экрана применяется процедура **ClearDevice**:

Procedure ClearDevice;

при этом указатель перемещается в левый верхний угол экрана.

10.5. Отображение точки и линии на экране

Все изображения на экране построены из точек. Для отображения точки используется процедура **PutPixel**:

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

где X,Y – координаты точки, Color – константа цвета точки.

Процедура вывода отрезка прямой (в текущем стиле и цвете) определена следующим образом:

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

где (X1,Y1) и (X2,Y2) – координаты начальной и конечной точек линии.

Для построения отрезков применяются еще две процедуры:, которые чертят линию от текущего указателя:

а) **Procedure LineTo (X, Y : Integer);** - чертит линию от текущей позиции указателя до точки с координатами (X,Y);

б) **Procedure LineRel (Dx, Dy : Integer); -** проводит линию из позиции текущего указателя до точки, определяемой приращениями Dx и Dy.

Модуль Graph позволяет вычерчивать линии различных стилей. Установка стиля производится процедурой:

Procedure SetLineStyle (LineType : Word; Pattern : Word; Thick : Word);

где **LineType** – тип линии (рис. 10.3), который задается константами:

SolidLn = 0; DotteLn = 1; CenterLn = 2; DashedLn = 3;

UserBitLn = 4 (тип линии, определяемый пользователем)

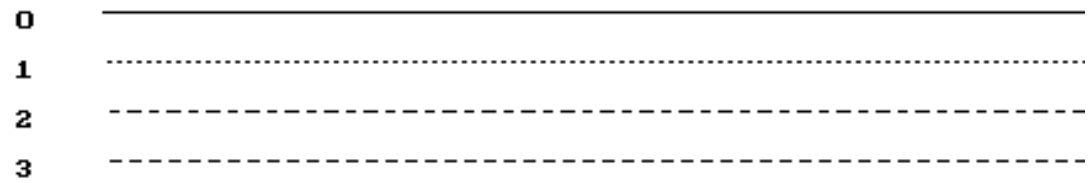


Рис. 10.3. Типы линий с константами 0..3

Pattern – шаблон (образец), учитывается только для линий, тип которых был определен пользователем;

Thick – толщина линии, которая задается с помощью двух констант:

NormWidth = 1 - толщина в один пиксел;

ThickWidth = 3 - жирная линия толщиной в три пиксела.

10.6. Вывод некоторых геометрических фигур

В модуле Graph предусмотрены процедуры для вывода ряда геометрических фигур. Наиболее часто используются следующие процедуры:

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

вычерчивает прямоугольник с использованием текущих цвета и стиля линии. Область внутри прямоугольника не закрашивается и совпадает по цвету с фоном;

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

рисует прямоугольник, внутренняя область которого залита по текущему шаблону;

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

вычерчивает окружность радиуса **R** с центром в точке (**X,Y**).

10.7. Вывод текста в графическом режиме

Вывод текста на экран выполняется с помощью двух процедур:

1) **Procedure OutText (Txt : string);** - выводит строку **Txt**, начиная с текущей позиции указателя;

2) **Procedure OutTextXY (X,Y : integer; Txt : string);** - выводит строку, начиная с позиции, задаваемой координатами **X,Y**.

Примечание. Для вывода численных данных необходимо предварительно перевести численную константу в строковую с помощью процедуры **Str**, например:

```
Var f,a:real;
   Fst:string; {Fst - строковое представление
……                    числа f}
f:=sin(a)+cos(a);
str(f,Fst);
```

```
OutTextXY(250,100,'f= '+Fst);
……….
```

Размещение выводимого текста относительно графического указателя – ***юстировка*** – определяется с помощью процедуры

Procedure SetTextJustify (Horiz, Vert : word);

где **Horiz** – метод выравнивания по горизонтали, задаваемый тремя константами:

LeftText = 0 – выравнивание по левому краю (указатель находится слева от текста);

CenterText = 1 – выравнивание по центру;

RightText = 2 – выравнивание по правому краю (указатель – справа от текста);

Vert – метод выравнивания по вертикали, который также можно задать тремя константами:

BottomText = 0 – выравнивание по нижнему краю (указатель – внизу текста);

CenterText = 1 – выравнивание по центру;

TopText = 2 – выравнивание по верхнему краю (указатель – сверху текста).

По умолчанию действуют установки Horiz = LeftText и

Vert = TopText.

Для установки стиля текстового вывода используется процедура:

Procedure SetTextStyle (Font, Direct, Size : word);

где **Font** - код (номер) шрифта.

По умолчанию после инициализации устанавливается матричный шрифт 8x8 (единственный шрифт для вывода кириллицы). Этому шрифту соответствует константа DefaultFont = 0.

Direct - код направления (HorizDir = 0 - горизонтальное, VertDir = 1 - вертикальное);

Size - коэффициент увеличения размера букв (до 32).

Пример. Составить программу расчета и вывода графика функции **a = tg b** в области $-1{,}5 \le b \le 1{,}5$; $-15 \le a \le 15$. Цвета фона, выводимых символов, линий и точек задать самостоятельно.

При этом в программе необходимо осуществить масштабирование координат **a** и **b** по соотношениям

$$M_a = \frac{\Delta a}{\Delta y}, \quad M_b = \frac{\Delta b}{\Delta x},$$

где Δa, Δb – цены делений реальных координат;
ΔY, ΔX – число пиксел в одном делении ординаты Y и абсциссы X;
M_a, M_b – масштабные коэффициенты, определяющие цену одного пиксела.

В результате координаты a и b можно записать в виде $a = M_a \cdot Y$, $b = M_b \cdot X$, а функцию $a = f(b)$ представить следующим образом:

$$Y = \frac{1}{M_a} f(M_b \cdot X),$$

где Y, X – координаты a и b, выраженные в пикселах.

Текст программы

```
Program Tangens;
Uses Graph,crt,printer;
Var Gd, Gm,X0,Y0,Xmax,Xmin,X1,Y1,Lx,Ly,
   x,y,i,Dx,Dy:Integer;
   Bmin,Bmax,Amin,Amax,Mx,My,Da,Db,f:real;
Begin
writeln('Введите исходные данные');
write('Минимальное значение аргумента Bmin:'); readln(Bmin);
write('Максимальное значение аргумента Bmax:');
readln(Bmax);
write('Минимальное значение функции Amin:'); readln(Amin);
write('Максимальное значение функции Amax:'); readln(Amax);
{Задание масштаба по координатным осям X и Y}
Da:=5; Db:=Pi/4; {Выбор цены деления реальных координат}
```

```
    Dx:=50; Dy:=60;
    {Dx,Dy - число пиксел, содержащихся в одном делении}
    Mx:=Db/Dx; My:=Da/Dy; {Mx-масштаб по оси X, My-по оси Y}

          {Инициализация графического режима}
     Gd:=Detect;
     InitGraph(Gd, Gm, 'c:\pascal\turbo\bgi');
    { Путь к BGI     драйверам }
    If GraphResult <> grOk Then Halt(1);

     SetBkColor(7);      {Установка цвета фона экрана - серый}
     SetFillStyle(1,15); {Установка стиля заливки - сплошной белый}

     X1:=200; Y1:=80;    {Задание координат верхнего левого угла
          координатной сетки}
     Lx:=Dx*4; Ly:=Dy*6;
    {Задание длины координатной сетки по осям X и Y}
     bar(X1,Y1,X1+Lx,Y1+Ly);
    {Формирование прямоугольника  координатной сетки}
     X0:=X1+2*Dx; Y0:=Y1+3*Dy;       {задание начала координат}

    {формирование координатной сетки}
    SetLineStyle(3,0,1); {Задание стиля для внутренних линий - пунк-
тир}
    SetColor(7);         {Задание цвета для внутренних линий}
    For i:=0 to 6 do     {Формирование горизонтальных линий}
       Line(X1,Y1+i*Dy,X1+Lx,Y1+i*Dy);
    For i:=0 to 4 do     {Формирование вертикальных линий}
       Line(X1+i*Dx,Y1,X1+i*Dx,Y1+Ly);
    SetLineStyle(0,0,1); {Смена стиля линий на сплошной}
    Line(X0,Y1,X0,Y1+Ly);   {Формирование линий, проходящих
    Line(X1,Y0,X1+Lx,Y0);         через начало координат}
```

```
    SetColor(1);       {Установка синего цвета выводимых элементов}
    RectAngle(X1,Y1,X1+Lx,Y1+Ly);   {Формирование внешних гра-
ниц  коорд. сетки}

                  {Формирование надписей}
    {Вдоль оси Y}  SetTextJustify(2,1);        {Установка выравни-
          OutTextXY(X1-5,Y1+Dy,'10');   вания текста}
          OutTextXY(X1-5,Y1+2*Dy,'5');
          OutTextXY(X1-5,Y1+3*Dy,'0');
       OutTextXY(X1-5,Y1+4*Dy,'-5');
          OutTextXY(X1-5,Y1+5*Dy,'-10');
          OutTextXY(X1-5,Y1,'a');
    {Вдоль оси X} SetTextJustify(1,2);
          OutTextXY(X1,Y1+Ly+6,'-Pi/2');
          OutTextXY(X1+Dx,Y1+Ly+6,'-Pi/4');
          OutTextXY(X1+2*Dx,Y1+Ly+6,'0');
          OutTextXY(X1+3*Dx,Y1+Ly+6,'Pi/4');
          OutTextXY(X1+4*Dx,Y1+Ly+6,'b');
    {Заголовок}   SetTextJustify(1,2);
          SetColor(11);
          SetTextStyle(0,0,2);
       OutTextXY(X1+2*Dx,Y1-40,'График функции a=tg(b)');

    {Вывод графика функции}
    Xmin:=Round(Bmin/Mx);  {определение минимального и макси-
маль-
    Xmax:=Round(Bmax/Mx); { ного значений аргумента в пикселах}

    SetColor(4);
    SetLineStyle(0,0,1);
    For x:=Xmin to Xmax do
    begin
    f:=sin(Mx*x)/cos(Mx*x)/My;          {f - значение, принимаемое
```

```
y:=round(f);                                    { функцией,в пикселах}
if (y>=-3*Dy)and(y<=3*Dy) then
PutPixel(X0+x,Y0-y,4); {Вывод пиксела с учетом переноса начала
                                                координат}
delay(1000);
end;
ReadLn;
CloseGraph;     {Завершение работы с графическим режимом}
End.
```

Результат выполнения программы представлен на рис. 10.4.

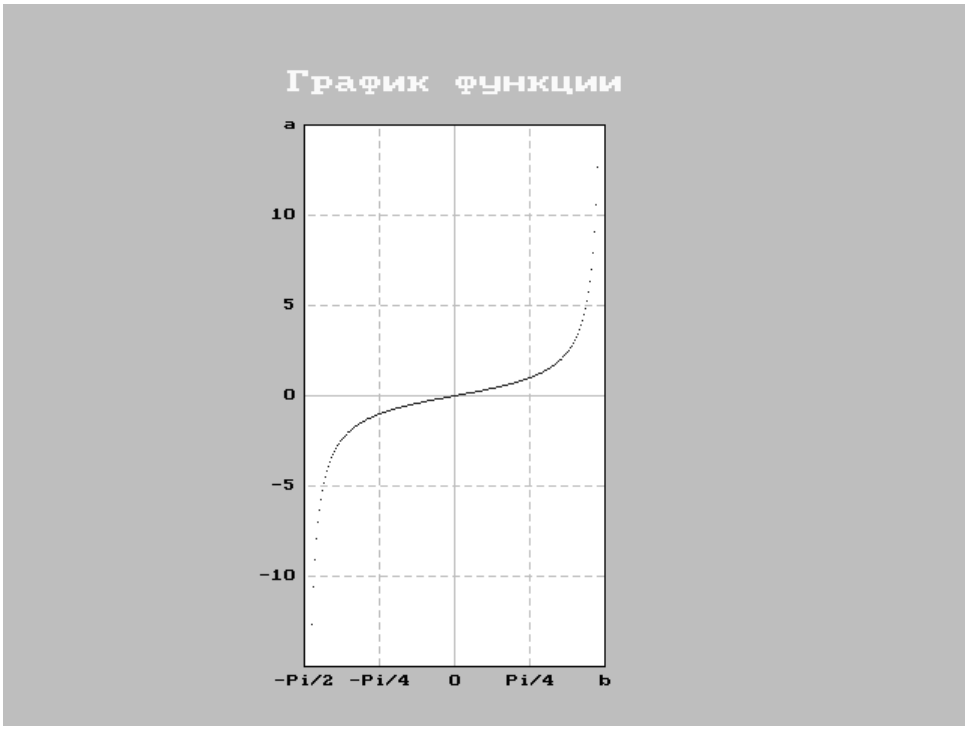


Рис. 10.4. Результат выполнения программы Tangens

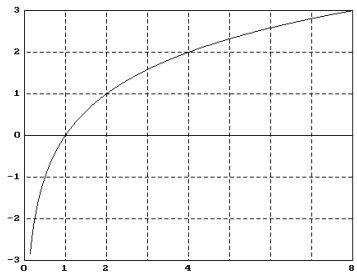
!! *Проанализируйте программу tangens, введите ее текст в компьютер, откомпилируйте ее, запустите на счет, введите запрашиваемые данные и просмотрите результаты.*

Упражнения

Составить программу расчета и вывода графика функции **a = f(b)**. Цвета фона, выводимых символов, линий и точек задать самостоятельно. Варианты заданий представлены в табл.10.1.

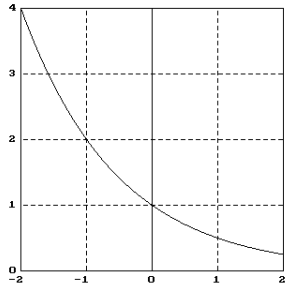
Таблица 10.1

Варианты заданий



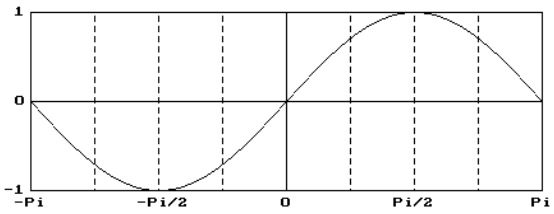
1. $a = \log_2 b$

при $0{,}01 \le b \le 8$
$-3 \le a \le 3$

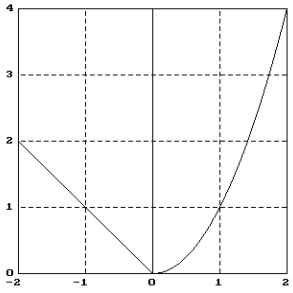


2. $a = \left(\frac{1}{2}\right)^b$

при $-2 \le b \le 2$
$0{,}25 \le a \le 4$



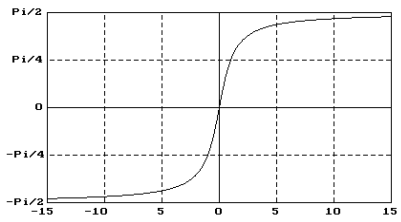
3. $a = \sin b$ при $-3{,}14 \le b \le 3{,}14$
$-1 \le a \le 1$



4. $a = \begin{cases} -b, \text{ если } b < 0; \\ b^2, \text{ если } b \ge 0 \end{cases}$

при $-2 \le b \le 2$
$0 \le a \le 4$

Окончание таблицы 10.1

	5. $a = \text{arctg}\, b$ при $-15 \le b \le 15$ $-1{,}5 \le a \le 1{,}5$

Контрольные вопросы

1. Какие процедуры используются для задания цвета фона, линий и символов?
2. Что такое стиль заполнения и какой процедурой он устанавливается?
3. С помощью каких процедур перемещается текущий указатель?
4. Как установить текущий указатель в центр экрана?
5. Дайте характеристику понятия "Окно" в графическом режиме и какой процедурой оно устанавливается?
6. Какие процедуры используются для очистки окна и экрана?
7. Каким образом можно провести линию?
8. Как осуществляется установка стиля вычерчивания линий?
9. Приведите основные процедуры задания прямоугольников и параллелепипеда.
10. Как начертить круг, сектор?
11. В чем заключается отличие процедур **OutText** и **OutTextXY**?
12. Как в графическом режиме выводятся численные данные?
13. Какой по умолчанию стиль вывода текста используется в модуле Graph?
14. Какие значения могут принимать константы цвета для символов и для фона?
15. Какой вид имеет система координат в графическом режиме?
16. Какая процедура предназначена для инициализации графического режима и какие параметры она имеет?
17. Какая функция позволяет выполнять обработку ошибок графического режима?

11. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ УКАЗАТЕЛЕЙ НА ЯЗЫКЕ С++

Указатель – это именованная переменная, представляющая собой символический адрес, по которому располагается объект определенного типа. С помощью этой переменной организуется косвенный способ адресации.

На практике указатели чаще всего используются при работе с **динамической памятью**. Это свободная память, в которой можно выделять во время работы программы место для временных (промежуточных) массивов данных переменной размерности. Участки выделенной динамической памяти называют **динамическими переменными** и обращаются к ним через указатели.

Общие сведения об указателях

Объявление указателя имеет следующий вид:

type *ptr;

где **ptr –** указатель на значение типа **type**;

* - символ унарной операции косвенной адресации.

Например:

```
int *z;      //z – указатель на целое число типа int
float *a;   //a – указатель на вещественное число типа float
void *p;   //p – указатель на тип void (бестиповый указатель)
```

Указатель **инициализируется** либо непосредственно при объявлении (для надежности), либо после объявления с помощью оператора присваивания. При этом ему присваивается конкретный или нулевой адрес.

Например:

```
int x, y=5;
int *yptr, *count=NULL; //указателю count присваивается нулевой адрес
int *xptr=&x;      //указателю xptr присваивается адрес переменной x
```

. . . . . .

yptr=&y; //указателю yptr присваивается адрес переменной y

Здесь **&** - символ унарной операции взятия адреса; **NULL** – символическая константа, равная нулевому адресу, по которому никогда не располагаются данные (этот адрес зарезервирован операционной системой для своих нужд).

Для того чтобы использовать значение, на которое указывает указатель, необходимо применить к указателю унарную операцию **разыменования** * (обращения по адресу).

Например:

x=*yptr; //присваивание числа y=5, находящегося по адресу yptr

Этот же результат можно получить, записав

x=y; **//**обычное присваивание без применения указателя

или

*xptr=*yptr; //по адресу xptr помещается значение,

//расположенное по адресу yptr

Операцию * нельзя применять к бестиповому указателю, так как неизвестно какой размер памяти нужно разыменовать. Прежде чем использовать такой указатель, ему необходимо присвоить значение указателя на нужный тип данных.

Например:

float *d, c=2; void *q;

q=d; //указатель q и d - синонимы указывают на тип float

*q=c; **//**по адресу q помещается значение **c** типа float

Указатель может изменять свое значение, т.е. ему можно присвоить адрес другой переменной того же типа.

Например:

yptr=&x; //yptr будет указывать на x

Подобно другим переменным указатель имеет собственный ад-

рес в памяти. Он может быть получен путем применения операции **&** к имени указателя.

Например:

cout<<&yptr; //печать адреса указателя yptr

Можно так же определить указатель на указатель и так далее сколько нужно раз.

Например:

int z=37;

int *pz=&z; //pz – указатель на z

int **ppz=&pz; //ppz – указатель на указатель pz

В данном случае при вызове **ppz последовательно обеспечивается доступ к участку памяти *ppz= =pz, затем *pz= =z, т.е. **ppz дает значение 37.

С указателями можно выполнять не только операции присваивания и обращения по адресу, но и ряд других операций:

- унарные операции **++**,--;
- аддитивные и составные операции с целыми числами **+, -, +=, -=;**
- вычитание указателей (сложение указателей запрещено);
- операции отношения.

Последние имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например с массивами. Исключение составляют операции **==** и !=, которые используются для любых типизированных указателей.

Массивы и указатели в C++ тесно связаны и могут быть использованы практически эквивалентно. Имя массива можно понимать как **константный указатель** на первый элемент массива. Его отличие от обычного указателя в том, что его нельзя модифицировать. Однако константный указатель можно присваивать переменной типа указатель.

Например:

```
int b[5], *pt;
pt=b;                  //эта запись эквивалентна pt=&b[0]
```

После инициализации указателя его можно использовать для ссылок на элементы массива **b[i]** с помощью записи ***(pt+i).** Указатели можно индексировать точно так же, как и массивы. Например, выражение **pt[i]** ссылается на элемент массива **b[i].** Учитывая, что b=&b[0], в целом можно считать, что b[i] – индексированный указатель. Эта взаимосвязь отражена графически на рис. 11.1.

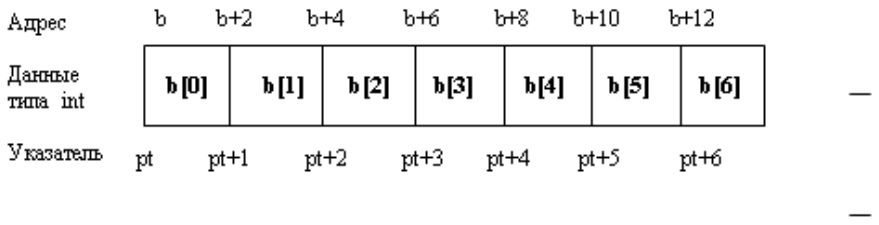


Рис. 11.1. Взаимосвязь указателя с одномерным массивом

Двумерные массивы заданной размерности располагаются в памяти подобно одномерным массивам, занимая построчно смежные ячейки. Поэтому для работы с таким массивом достаточно одного указателя.

Например:

```
float mas [4] [2], *ptr;
ptr=mas;          //или ptr=& mas[0] [0]
```

На рис. 11.2 представлена схема расположения массива **mas** в памяти компьютера.

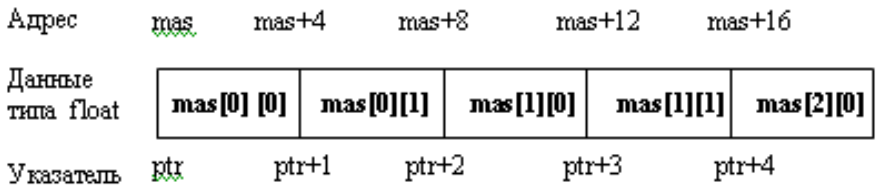


Рис. 11.2. Взаимосвязь указателя и двумерного массива

Применение указателей затрудняет составление и анализ про-

грамм, поэтому их используют в тех случаях, где без них невозможно обойтись или когда они позволяют создать более оптимальный по ряду характеристик программный код.

Возможности и преимущества указателей проявляются при работе с динамическими массивами, функциями, строками и структурированными данными.

Динамические массивы

Динамическим называется массив, размерность которого становится известной в процессе выполнения программы. Такие массивы делают последнюю более общей.

Формирование динамических массивов осуществляется с помощью указателей и средств распределения памяти. Память выделяется в специальной области оперативной памяти – **heap**, которую часто называют свободной памятью.

Для выделения памяти в C++ используется операция **new**, а для для ее освобождения – операция **delete.**

Ниже приводится общепринятый способ динамического распределения памяти под одномерный массив.

```
int n, *mas;          //объявление указателя
. . . . . .
cin>>n;               //ввод размерности массива
mas=new int[n];       //выделение памяти
. . . . . .
delete [] mas;        //освобождение памяти
mas=NULL;             //обнуление указателя
```

В данном примере переменная **mas** является указателем на массив из **n** элементов (и его именем одновременно). Ему присваивается адрес выделенной области памяти в соответствии с заданным типом объекта. Иногда объявление указателя и операцию **new** совмещают.

```
int *mas = new int[n];
```

Если объем любого свободного участка памяти недостаточен для

размещения массива, операция **new** возвращает нулевой указатель (адрес 0), индицируя тем самым безуспешную попытку выделения памяти. Поэтому результат операции **new** обычно проверяют следующим образом:

```
float *mas = new float[n];
if (mas == NULL)
{ cout<<"Свободной памяти нет"<<endl;
  exit (1);              //аварийное завершение программы
}
```

Пример. Формирование и сортировка массива с n элементами (лист. 11.1).

Листинг 11.1. ***ukaz_1.cpp***

```
#include <iostream.h>
#include <stdlib.h>                    //Для функции exit
void main()
{ int n;
  double a,*mas;          //mas - указатель на динамический массив
  //Ввод размерности массива
 cout<<"Введите размерность массива:\n";
 cout<<"n=",cin>>n;
 mas=new double[n];       //Выделение памяти под массив
 if(mas==NULL)
        {
  cout<<"Свободной памяти нет"<<endl;
  exit(1);
 }
  //Ввод массива
 cout<<"Введите элементы массива:\n";
 for(int i=0;i<n;i++)
 {
   cout<<"mas["<<i<<"]=";cin>>mas[i];
 }
```

```
        cout<<"\n Исходный массив чисел:\n"<<"mas["<<n<<"]={";
        for(int i=0;i<n;i++)
          cout<<mas[i]<<"  ";
          cout<<"}"<<endl;
         //Сортировка массива методом "пузырька"
       for(int i=0;i<n-1;i++)
       for(int j=i+1;j<n;j++)
        if (mas[j]<mas[i])
          {a=mas[i];
           mas[i]=mas[j];
           mas[j]=a;
          }
         //Вывод отсортированного массива
        cout<<"\nОтсортированный массив чисел:\n"<<"mas["<<n
<<"]={";
          for(int i=0;i<n;i++)
            cout<<mas[i]<<"  ";
        cout<<"}";
        delete [] mas;              //Освобождение памяти
        mas=NULL;                   //Инициализация указателя
      }
```

Результат выполнения программы

```
 Исходный массив чисел:
mas[6]={3.6  -1  4  3.4  -0.5  0  }
Отсортированный массив чисел:
mas[6]={-1  -0.5  0  3.4  3.6  4  }
```

!! *Проанализируйте программу. Создав новый файл проекта с именем ukaz_1.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

При работе с матрицами различной размерности возникает необходимость создания **двумерных** динамических массивов. В этом случае сначала с помощью операции **new** выделяется память под **n** ука-

зателей, где **n** – число строк матрицы. После этого каждому указателю присваивается с помощью **new** адрес выделенной области памяти с размером **m,** где **m** – число столбцов матрицы (рис. 11.3).

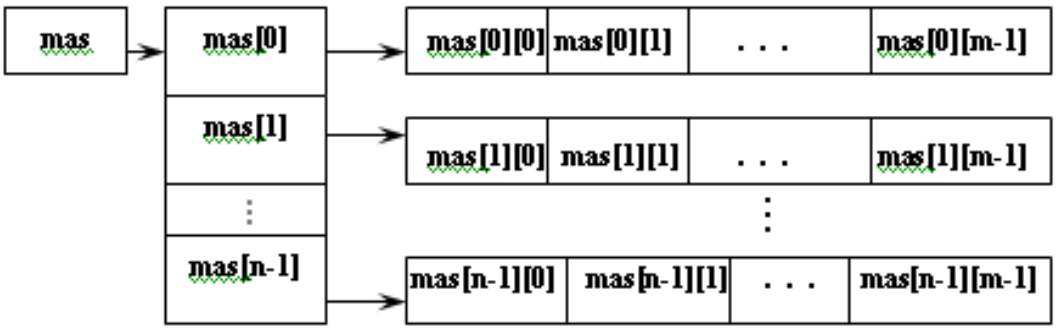


Рис. 11.3. Размещение динамического двумерного массива в памяти

Такой динамический массив по расположению элементов в памяти существенно отличается от статического двумерного массива, для которого память выделяется последовательно для всех строк. Для первого память выделяется последовательно только для каждой строки матрицы, а строки могут располагаться в разных местах памяти.

Пример. Формирование двумерного целочисленного массива размерностью n x m с вводом элементов матрицы с клавиатуры и последующим вычислением нормы матрицы по формуле

$$\|A\| = \max_i \sum_{j=1}^{m} |a_{ij}| \ .$$

Программа для решения данной задачи приведена в листинге 11.2.

Листинг 11.2. ***ukaz_2.cpp***

```
#include <iostream.h>
#include <stdlib.h>              //Для функции exit
#include <iomanip.h>             //Для функции setw
void main()
{
  int n,m,**mas;                 //mas - указатель на указатель
  //Ввод размерностей массива
```

```
cout<<"Введите размерность матрицы:\n";
cout<<"число строк n=",cin>>n;
cout<<"число столбцов m=",cin>>m;
mas=new int*[n];              //Выделение памяти под n указателей
if(mas==NULL)                 //Проверка выделения памяти
        {
 cout<<"Свободной памяти нет"<<endl;
 exit(1);
 }
 for(int i=0;i<n;i++)
 { mas[i]=new int[m];         //Выделение памяти для i строки
  if(mas[i]==NULL)            //Проверка выделения памяти
   {
    cout<<"Свободной памяти нет"<<endl;
    exit(1);
   }
 }
 //Ввод массива
cout<<"Введите элементы массива:\n";
 for(int i=0;i<n;i++)
  for(int j=0;j<m;j++)
{
   cout<<"mas["<<i<<"]"<<"["<<j<<"]=";
   cin>>mas[i][j];
}
//Вывод исходной целочисленной матрицы
cout<<"\nИсходный массив чисел:"
    <<"\nmas["<<n<<"]"<<"["<<m<<"]=\n";
for(int i=0;i<n;i++)
for(int j=0;j<m;j++)
 { cout<<setw(6)<<mas[i][j];
   if(j==m-1)cout<<endl;
 }
```

```
    cout<<endl;
   //Вычисление нормы матрицы
     int max=0;
    for(int i=0;i<n;i++)
    { int s=0;
     for(int j=0;j<m;j++)
     s+=abs(mas[i][j]);
     if (s>max) max=s;
    }
    cout<<"Норма матрицы="<<max;
    for(int i=0;i<n;i++)          //Освобождение памяти, отведенную
    delete mas[i];                 // под матрицу
    delete [] mas;               //Освобождение памяти под указатели
 }
```

Результат выполнения программы

Исходный массив чисел:

```
mas[3][4]=
    5    8    4    2
    4    9    6    4
    1    5    6    8
Норма матрицы=23
```

!! *Проанализируйте программу. Создав новый файл проекта с именем ukaz_2.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Указатели и динамическое распределение памяти – мощное средство C++. Однако возможности, которые получает программист при работе с указателями, накладывают на него большую ответственность, так как наибольшее количество ошибок вносится в программу именно при работе с указателями. К характерным ошибкам можно отнести следующие "промахи" неопытных программистов.

1. Использование неверного адреса в операции delete. В результа-

те внутренняя структура памяти будет испорчена, что может привести к порче нужной информации.

2. Пропущено освобождение памяти, т.е. отсутствует операция delete. Такие ошибки называют "утечками памяти". В результате многократных утечек памяти ее в итоге может не хватить, и произойдет сбой программы.

3. Запись по неверному адресу. В результате, скорее всего, будут испорчены какие-либо данные, получен неверный результат, сбой программы и т.п. Ошибку определить в этом случае очень сложно, так как она обычно проявляется не сразу.

Указатели и функции

Функция может возвращать одно скалярное результирующее значение. Для расчета нескольких значений используется другой подход – изменение значений параметров функции. Это можно сделать, если в качестве аргументов взять адреса переменных. В результате обеспечивается доступ к данным, расположенным по указанным адресам.

В списке параметров такой функции перед именем переменной указывается символ косвенной адресации '*', который свидетельствует о том, что в функцию передается указатель на переменную. В теле функции тоже перед именем параметра ставится символ операции разыменования '*'.

При вызове функции в нее в качестве аргумента должна передаваться не сама переменная, а ее адрес, получаемый с помощью операции **&**. Такая функция вызывается обычно как подпрограмма, т.е. в виде отдельного оператора, после которого ставится точка с запятой.

Пример. Обмен значениями двух переменных.

```
# include <iostream.h>
void chage (float*, float*);      // прототип функции
void main ( )
 { float a,b;
   cin>>a>>b;                     //ввод значений переменных
```

```
  chage (&a, &b);                 //вызов функции
  cout<<"a="<<a<<"\tb="<<b;
 }
 void chage (float *x, float *y)      //x, y - указатели
  { float z;
    z=*x; *x=*y; *y=z;              //обмен значениями
   }
```

В языке С++ имя массива представляет собой константный указатель адреса элемента массива с нулевым индексом. Отсюда следует, что при использовании имени массива в качестве параметра функции допускается изменять значения элементов массива в теле функции. При этом для одномерных массивов можно использовать равноправные описания перечня параметров в заголовках функций (сигнатуры):

(int n, float a[], float b[]) //традиционный способ

(int n, float *a, float *b) //альтернативный способ

Для динамических массивов применяется второй вариант определения функции.

Если используются динамические двумерные массивы, то в заголовке функции записывается указатель на массив указателей на строки матрицы:

(int n, float **a, float **b)

Динамический двумерный массив, сформированный непосредственно в вызываемой функции, невозможно вернуть в вызывающую функцию в качестве результата. Однако возвращаемым значением может быть указатель на одномерный массив указателей на одномерные массивы с элементами заданного типа.

Например:

double **matr (int n, int m) //n, m – размерности массива

Таким образом динамические массивы создаются в функции, а доступ к ним выполняется в основной программе. В этой же про-

грамме освобождается память, выделенная под динамические массивы.

Пример. Перемножение матриц A и B соответственно с размерностями (n x m) и (m x r) (лист. 11.3).

Листинг 11.3. ***ukaz_3.cpp***

```
/*Программа перемножения матриц C=A*B
соответственно с порядками [n*m],[m*r]*/
 #include <iostream.h>
#include <iomanip.h>
void mult_matr(int,int,int,int**,int**,int**);
void cin_matr( int,int,char,int**);
void cout_matr(int,int,char,int**);
int **RAM(int,int);
void del_RAM(int,int**);
void main()
{
  int n,m,r,**a,**b,**c;
  //Ввод размерностей матриц
 cout<<"Введите размерности матриц:\n";
 cout<<" n=",cin>>n;
 cout<<" m=",cin>>m;
 cout<<" r=",cin>>r;
    a=RAM(n,m);          //Выделение памяти под матрицу A
    b=RAM(m,r);          //Выделение памяти под матрицу B
    c=RAM(n,r);          //Выделение памяти под матрицу C
   //Вызовы функций
  cin_matr(n,m,'A',a);        //Ввод матрицы A
  cin_matr(m,r,'B',b);        //Ввод матрицы B
  mult_matr(n,m,r,a,b,c);     //Вычисление матрицы C
  cout_matr(n,m,'A',a);       //Вывод матрицы A
  cout_matr(m,r,'B',b);       //Вывод матрицы B
  cout_matr(n,r,'C',c);       //Вывод матрицы C
   //Освобождение выделенной памяти
```

```
    del_RAM(n,a);            //под массив A
    del_RAM(m,b);            //под массив B
    del_RAM(n,c);            //под массив C
}
  //Функция выделения памяти
 int **RAM(int p,int q)
 {int **s;
  s=new int*[p];
  for(int i=0;i<p;i++)
   s[i]=new int[q];
   return s;
 }
  //Функция освобождения памяти
  void del_RAM(int k,int**z)
  { for(int i=0;i<k;i++)
    delete z[i];
    delete []z;
  }
   //Функция ввода  матрицы
 void cin_matr(int p,int q,char Y,int**X)
 {cout<<"\nВведите элементы матрицы "<<Y<<endl;
  for(int i=0;i<p;i++)
   for(int j=0;j<q;j++)
 {
   cout<<Y<<"["<<i<<"]"<<"["<<j<<"]=";
   cin>>X[i][j];
 }
 }
    //Функция вывода матрицы
  void cout_matr(int p,int q,char Y,int**X)
 {
 cout<<"\n Матрица "
    <<Y<<"["<<p<<"]"<<"["<<q<<"]=\n";
```

```
 for(int i=0;i<p;i++)
 for(int j=0;j<q;j++)
  { cout<<setw(6)<<X[i][j];
    if(j==q-1)cout<<endl;
  }
 }
  //Функция перемножения матриц C=A*B
 void mult_matr(int N,int M,int R,
                int **A,int **B,int **C)
 {   for(int i=0;i<N;i++)
     for(int j=0;j<R;j++)
  {
    C[i][j]=0;
    for(int k=0;k<M;k++)
    C[i][j]+=A[i][k]*B[k][j];
  }
 }
```

Результат выполнения программы

```
Матрица A[3][4]=
    1     6     8     3
    5     7     8     3
    6     8     3     2
Матрица B[4][2]=
    1     4
    6     8
    4    23
    0     8
Матрица C[3][2]=
   69   260
   79   284
   66   173
```

!! *Проанализируйте программу. Создав новый файл проекта с именем ukaz_3.ide, наберите в нем текст данной программы, откомпилируйте ее и произведите запуск программы на выполнение.*

Упражнения

Составить и отладить программу решения задачи согласно приведенным в табл.11.1 вариантам заданий.

Дополнительные требования. В программе должны быть оформлены в виде функций следующие процессы:

1) выделение памяти под динамические массивы;

2) заполнение массива A значениями, которые вычисляются алгоритмически на основе исходных данных (табл. 11.1);

3) действия над массивами, необходимые для решения поставленной задачи согласно выданному заданию (формирование нового массива и т.д.)

4) вывод результатов вычислений (результирующего и исходного массивов) на дисплей;

5) освобождение памяти, выделенной под динамические массивы.

Таблица 11.1

Варианты заданий

1	а) Сформировать матрицу A с размерами n x m и присвоить ее элементам значения $a_{ij}=F(x_i, y_j)$, где $F(x_i,y_j)=\sin y_j - \cos x_i$, $x_{min}\le x_i\le x_{max}$, $i=1\div n$, $y_{min}\le y_j\le y_{max}$, $j=1\div m$. Вводимые данные: x_{min}, $x_{max,}$, n, m, y_{min}, y_{max}. б) Получить из матрицы A матрицу B, элементами которой являются средние арифметические значения элементов матрицы A без элемента, соответствующего формируемому. Затем сложить матрицы: C=A+B. Вывести на печать матрицы A, B, C

Продолжение табл. 11.1

2	а) Сформировать матрицу A с размерами n x m и присвоить ее элементам значения $a_{ij}=F(x_i, y_j)$, где

	$F(x_i, y_i) = x_i y_i^{\,2} + 0{,}5x_i^2 - 0{,}04 y_j \ldots x_{\min} \le x_i \le x_{\max}\ i = 1 \div n,\ y_{\min} \le y_j \le y_{\max},\ j = 1 \div m$ Вводимые данные: x_{min}, $x_{max,}$, n, m, y_{min}, y_{max}. б) Получить из матрицы A транспонированную матрицу B = A^T по правилу $b_{ij} = a_{ji}$. Затем получить матрицу C=B*A. Вывести на печать матрицы A, B, C
	а) Сформировать матрицу A с размерами n x m и присвоить ее элементам значения $a_{ij}=F(x_i, y_j)$, где $F(x_i,y_j)=\sqrt{2-(x_i-0{,}5)^2-0{,}9\,y_j^2}+0{,}5x_i y_j$, $x_{min} \le x_i \le x_{max}$, $i=1 \div n$, $y_{min} \le y_j \le y_{max}$, $j=1 \div m$. Вводимые данные: x_{min}, $x_{max,}$, n, m, y_{min}, y_{max}. б) Получить из матрицы A матрицу B, элементы которой получаются как среднее квадратичное значение элементов матрицы A без элемента, соответствующего формируемому. Затем получить матрицу C=A-B. Вывести на печать матрицы A, B, C
4	а) Сформировать матрицу A с размерами n x m и присвоить ее элементам значения $a_{ij}=F(x_{i,}\, y_j)$, где $F(x_i,y_j)=(5x_i+7y_j-25)\,e^{-\left(x_i^2+x_i y_j+y_j^2\right)}$, $x_{min} \le x_i \le x_{max}$, $i=1 \div n$, $y_{min} \le y_j \le y_{max}$, $j=1 \div m$. Вводимые данные: x_{min}, $x_{max,}$, n, m, y_{min}, y_{max}. б) На основании матрицы A сформировать матрицу B путем циклического сдвига строк матрицы A на k элементов вверх ($k \le n$). Затем получить D=A + B и $C=D^T$ по правилу $c_{ij}=d_{ji}$. Вывести на печать матрицы A, B, C

Окончание табл. 11.1

	а) Сформировать матрицу A = $\{a_{ij}\}$ с размерами n x m, из предварительно сформированного одномерного массива, элементы которого определяются по соотношению $x_i = (ax_{i-1} + b) \bmod c$, где i=1,2..., c=256, a=17, b=11, x_0 = 172, где mod – операция определения остатка от деления нацело. б) На основе матрицы A сформировать матрицу B, элементами которой являются средние арифметические значения элементов матрицы A без элемента, соответствующего формируемому. Затем получить D=A - B и $C=D^T$ по правилу $c_{ij}=d_{ji}$. Вывести на печать матрицы A, B, C
	а) Ввести размеры n и m матрицы A = $\{a_{ij}\}$ и определить значения

	ее элементов по следующему правилу: $a_{ij} = \frac{m}{i}\exp\left\{\frac{-i^2}{n \cdot m}(j-n)^2\right\}$, где i=1÷n, j=1÷m. б) На основании матрицы A сформировать матрицу B путем циклического сдвига строк матрицы A на k элементов вниз (k ≤ n). Затем получить D=A + B и C=D^T по правилу $c_{ij}=d_{ji}$. Вывести на печать матрицы A, B, C
	а) Ввести размеры n и m матрицы A = {a_{ij}} и определить значения ее элементов по следующему правилу: $a_{ij} = \exp\{-\frac{b(i-1)+d(n+1-i)}{n} \cdot \frac{j-1}{m} \cdot c\}$, где b=1.5, c=3.0, d=0.5. б) На основании матрицы A сформировать матрицу B путем циклического сдвига столбцов матрицы A на k элементов вправо (k ≤ m). Затем получить D=A - B и C=D^T по правилу $c_{ij}=d_{ji}$. Вывести на печать матрицы A, B, C

Контрольные вопросы

1. Поясните, что такое указатель.
2. Каким образом объявляется и инициализируется указатель?
3. Что такое операция разыменования? Приведите пример.
4. Каким образом осуществляется инициализация бестипового указателя? Приведите пример.
5. Как определить указатель на указатель? Приведите пример.
6. Какие операции можно выполнить с указателями?
7. Какой элемент выполняет функцию константного указателя в массиве и на что он указывает?
8. В чем заключается эквивалентность указателей и массивов? Поясните их взаимосвязь на примере.
9. Как располагаются в памяти двумерные массивы с известной размерностью на этапе компиляции?
10. При работе с какими типами данных проявляются возможности и преимущества указателей?

11. Что такое динамический массив?

12. Каким образом выделяется память под одномерные динамические массивы?

13. Для чего необходимо освобождать память с помощью delete?

14. Как выделяется и освобождается память под двумерный динамический массив?

15. Для чего необходимо осуществлять проверку результата операции new?

16. Какие основные ошибки при работе с указателями допускают неопытные программисты?

17. Каким образом обеспечивается изменение значений переменных с помощью функций?

18. Как осуществляется изменение элементов одномерного динамического массива с использованием функций?

19. Каким образом передается в функцию двумерный динамический массив с целью его изменения?

20. Как осуществляется возврат сформированного в функции динамического массива?

ЗАКЛЮЧЕНИЕ

Развитие современной IT-индустрии идет очень быстрыми темпами. С каждым годом появляются новые аппаратные и программные средства вычислительной техники. Все большую потребность испытывает рынок программных средств в создании качественно новых продуктов, использующих аппаратные возможности ЭВМ. Активно развивается разработка мобильных приложений.

Огромный объем информации по программированию, представленный в различных источниках не позволяет сориентироваться начинающему программисту. В этой ситуации особое значение имеет освоение основных принципов и подходов к написанию программ.

Дальнейшее развитие программирования связано с совершенно новыми подходами к разработке программных продуктов, основанными на использовании языков четвертого поколения (4GL). Решение

современных задач требует от программиста качественно новых знаний, так как идет речь о создании таких программных комплексов, аналогов которых до настоящего времени не существовало.

Особую актуальность приобретают облачные вычисления (cloud computing) – технология обработки данных, где аппаратные и программные ресурсы предоставляются пользователю в качестве Интернет-сервисов. Пользователь имеет доступ к собственным данным, но не может управлять и не должен заботиться об инфраструктуре, операционной системе и собственно программном обеспечении, с которым он работает. Облачные вычисления реализуют клиент-серверную технологию, при которой используются ресурсы группы серверов в сети (процессорное время, оперативная память, дисковое пространство, сетевые каналы; специализированные контроллеры, программное обеспечение и т.п.). Таким образом, для пользователя все выглядит как предоставление единого виртуального сервиса.

Авторы надеются, что предлагаемое учебное пособие поможет студентам разобраться в базовых приемах и подходах к написанию программ, а также подготовит их к дальнейшему освоению новых знаний в области программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Березин Б.И.,Березин С.Б. Начальный курс С и С++.-М.:ДИАЛОГ-ИФИ,2003.-288с.

2. Будин В.И., Крайнова Е.А. Программирование в интегрированной среде TurboPascal: Учеб. пособ. – Самар. гос. техн. ун-т. Самара, 2004. 148 с.

3. В.И. Будин, С.Н. Майорова Информатика. Основы программирования на С++: Учеб. пособ./ В.И. Будин, С.Н. Майорова ;– Самар. гос. техн. ун-т. Самара, 2005. 124 с.

4. Городняя Л.В. Основы функционального программирования: Курс лекций для вузов.-М.: ИНТУИТ. РУ,2004.-280с.

5. Демидович Е.М. Основы алгоритмизации и программирования. Язык Си : учеб.пособие для вузов.-СПб.:БХВ-Петербург,2006.-440с.

6. Иванова Г.С. Основы программирования: учебник для вузов.-4-е изд.,испр.-М.: Изд-во МГТУ им. Н.Э.Баумана, 2007.-416с. : ил.

7. Немнюгин С.А. Turbo Pascal.Программирование на языке высокого уровня: учебник для вузов.-2-е изд.-СПб.:Питер,2005.-544с. :ил.

8. Подбельский В.В. Язык Си++ : учеб. пособие для вузов.-5-е изд.-М.:Финансы и статистика, 2007.-560с.:ил.

9. Холзнер С. Visual С++6: учебный курс[пер. с англ.].-СПб.:Питер, 2006.-570с.: ил.

ОГЛАВЛЕНИЕ

Учебное пособие

БУДИН Владимир Иванович
КРАЙНОВА Екатерина Анатольевна
МАЙОРОВА Светлана Николаевна
ТАРАКАНОВ Алексей Валерьевич

Программирование

Редакторы:
Е.С. Захарова
И. А. Назарова

Подписано в печать 30.12.2013г.
Формат 60x84 1/16. Бумага офсетная
Усл. п. л. 12,09 Уч.-изд. л. 10
Тираж 100 экз. Рег. № 11/13sf

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Самарский государственный технический университет»
443100, г. Самара, ул. Молодогвардейская, 244. Главный корпус

Отпечатано в типографии
Самарского государственного технического университета
Филиал в г. Сызрани, 446001, г. Сызрань, ул. Советская 45