**ГОСУДАРСТВЕННАЯ АКАДЕМИЯ НАУК**

**РОССИЙСКАЯ АКАДЕМИЯ ОБРАЗОВАНИЯ**

**ИНСТИТУТ ИНФОРМАТИЗАЦИИ ОБРАЗОВАНИЯ**

**Касторнов А.Ф., Касторнова В.А.**

# ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ ПАСКАЛЬ

***Учебно-методическое пособие***

**Москва, 2011**

УДК

ББК

М

**Касторнов А.Ф., Касторнова В.А. Практикум по программированию на языке паскаль. Учебно-методическое пособие. – М.: ИИО РАО, 2011. – 84 с.**

Практикум посвящен реализации учебно-методической поддержки содержательной линии «Алгоритмизация и программирование» предмета «Информатика» для старшего звена средней образовательной школы на основе изучения программирования на языке Паскаль. В пособии рассматриваются основные понятия языков структурного программирования на примере языка Паскаль, описываются различные типы данных, набор операторов для реализации базовых алгоритмических структур, создание процедур и функций, программирование графики, приведены примеры разработки программ, имеются задания для проведения лабораторных работ.

**Kastornov A.F., Kastornova V.A. The algorithmic programming language PASCAL practical work**

This manual is dedicated to the creation of a subject “Informatics” educational methodical “Algorithmization & programming” substantial line supporting for the secondary school higher forms by the use of PASCAL programming studying. The main algorithmic programming languages concepts on an example of PASCAL are considered, different data types, operators sets for basic algorithmic structures, procedures and functions creating, graphics programming are described, programs examples and laboratory tests are given in this manual.

© Касторнов А.Ф., 2011

© Касторнова В.А., 2011

© ИИО РАО, 2011

*ОГЛАВЛЕНИЕ*

[Предисловие 5](#_Toc312328602)

[Общие указания к выполнению лабораторных работ 6](#_Toc312328603)

[1. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ 7](#_Toc312328604)

[1.1. Оператор присваивания и выражения 7](#_Toc312328605)

[1.2. Операторы вызова процедур. Ввод/вывод данных 8](#_Toc312328606)

[1.2.1. Процедуры ввода READ и READLN 9](#_Toc312328607)

[1.2.2. Процедуры вывода WRITE и WRITELN 10](#_Toc312328608)

[Лабораторная работа № 1 11](#_Toc312328609)

[2. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ 14](#_Toc312328610)

[2.1. Составной и пустой операторы 14](#_Toc312328611)

[2.2. Организация ветвлений. Операторы выбора 15](#_Toc312328612)

[2.2.1. Оператор ветвления IF 15](#_Toc312328613)

[2.2.2. Оператор варианта CASE 17](#_Toc312328614)

[Лабораторная работа №2 19](#_Toc312328615)

[3. ОРГАНИЗАЦИЯ ЦИКЛОВ 23](#_Toc312328616)

[3.1. Оператор WHILE 23](#_Toc312328617)

[3.2. Оператор REPEAT 24](#_Toc312328618)

[3.3. Оператор FOR 25](#_Toc312328619)

[Лабораторная работа № 3 27](#_Toc312328620)

[4. ОРГАНИЗАЦИЯ ПОДПРОГРАММ. ПРОЦЕДУРЫ И ФУНКЦИИ. ПРОЦЕДУРЫ И ФУНКЦИИ 29](#_Toc312328621)

[4.1. Процедуры и их типизация 30](#_Toc312328622)

[4.1.1. Встроенные процедуры 31](#_Toc312328623)

[4.1.2. Процедуры пользователя 32](#_Toc312328624)

[4.1.3. Процедуры без параметров 32](#_Toc312328625)

[4.1.4. Процедуры с параметрами-значениями 33](#_Toc312328626)

[4.1.5. Процедуры с параметрами-переменными 35](#_Toc312328627)

[4.1.6. Комбинированные процедуры 36](#_Toc312328628)

[4.2. Функции пользователя. Рекурсивные функции 38](#_Toc312328629)

[4.2.1. Определение функции 38](#_Toc312328630)

[4.2.2. Функции пользователя 39](#_Toc312328631)

[4.2.3. Рекурсивные функции 40](#_Toc312328632)

[Лабораторная работа №4 42](#_Toc312328633)

[5. ОБРАБОТКА МАССИВОВ 46](#_Toc312328634)

[5.1. Одномерные массивы 46](#_Toc312328635)

[5.2. Многомерные массивы 48](#_Toc312328636)

[5.3. Примеры работы с массивами 49](#_Toc312328637)

[Лабораторная работа №5 50](#_Toc312328638)

[6. ОБРАБОТКА СТРОКОВЫХ ВЕЛИЧИН 55](#_Toc312328639)

[6.1. Тип данных CHAR 55](#_Toc312328640)

[6.2. Массивы литер 57](#_Toc312328641)

[6.3. Тип данных STRING 58](#_Toc312328642)

[6.4. Строковые функции и процедуры 59](#_Toc312328643)

[Лабораторная работа №6 62](#_Toc312328644)

[7. ПРОГРАММИРОВАНИЕ ГРАФИКИ 65](#_Toc312328645)

[7.1. Инициализация графического режима 65](#_Toc312328646)

[7.2. Простейшие графические операторы (процедуры) языка TurboPascal 67](#_Toc312328647)

[7.3. Основные приемы работы с графикой 70](#_Toc312328648)

[7.3.1. Работа с цветом 71](#_Toc312328649)

[7.3.2. Заполнение (закрашивание) произвольной замкнутой фигуры 72](#_Toc312328650)

[7.3.3. Построение простейших геометрических фигур 73](#_Toc312328651)

[Лабораторная работа №7 79](#_Toc312328652)

[Литература 84](#_Toc312328653)

## Предисловие

Настоящий практикум предназначен для профильного курса старшего звена средней образовательной школы по теме «Алгоритмизация и программирование», предусматривающего более глубокое изучение технологии процедурного программирования на основе языка Паскаль.

Пособие включает в себя выполнение 7 лабораторных работ, включающих в себя три работы на освоение построения линейных, разветвляющихся и циклических программ. В четвертой работе учащийся познакомится с подпрограммами, механизмом передачи параметров между подпрограммами, а также с рекурсивными функциями. Работа 5 посвящена изучению алгоритмов описания и обработки массивов. Здесь же приведены примеры реализации рассмотренных алгоритмов в Паскале. Обработка строковых величин рассмотрена в работе 6, где предусмотрено освоение работы со строкой как с единой последовательностью символов, так и со строкой, состоящей из слов. Способы обработки графических изображений средствами языка Паскаль (в среде TurboPascal) представлены в работе 7.

Выполнение лабораторных работ опирается на знание теории, вот почему каждая из объявленных выше тем курса предваряется необходимым теоретическим материалом, где помимо собственно теории имеются примеры ее использования при решении практических задач.

В каждой из семи тем практикума после изложения теории идут контрольные вопросы, ответы на которые могут служить неким тестом по готовности учащегося к выполнению практического задания. Само задание начинается с формулировки цели выполнения работы, а затем рассматриваются типичные примеры решения задач по этой теме. Конечно, при проведении лабораторных работ каждый ученик должен иметь свое собственное задание, однако мы ограничились только четырьмя вариантами, рассчитывая на то, что учитель, ориентируясь на приведенные в пособии примеры, можем сам «сгенерировать» недостающие варианты заданий.

Заметим, что пособие содержит довольно обширный материал по языку Паскаль и учитель может сам выбрать для изучения только те темы, которые интересны учащимся и самому учителю.

## Общие указания к выполнению лабораторных работ

Основная цель при выполнении лабораторных работ - практическое освоение всех этапов разработки программ для решения задач, начиная от анализа условия задачи и заканчивая сдачей отчета по написанной программе.

Задание каждой лабораторной работы включает не менее двух задач и предусматривает следующие виды работ:

1. Анализ условия задачи и выработка подхода к ее решению.
2. Пошаговая разработка алгоритма решения и его описание.
3. Составление блок-схемы алгоритма.
4. Разработка программы на языке Паскаль.
5. Отладка программы и демонстрация правильной ее работы на выбранном наборе данных.

Подготовка к выполнению задания осуществляется во внеурочное время. Выполнив лабораторную работу, учащийся оформляет отчет, который должен включать:

Тема и цель работы.

Условия задания.

Схема алгоритма решения задачи:

- математическая модель задачи;

- блок схема алгоритма.

Анализ алгоритма.

Текст программы и размещение исходных данных при вводе.

Результаты выполнения программы.

Обоснование правильности разработанной программы.

Выводы.

При защите отчета проверяется правильность его выполнения, учащийся должен быть готов отвечать на контрольные вопросы и уметь пояснять как работу программы в целом, так и ее отдельных частей.

# ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

Всякая Паскаль-программа есть последовательность операторов, которые подразделяются на простые и структурные. Каждый оператор имеет определенную структуру и записывается с использованием служебных слов и символов языка. Говорят, что оператор характеризуется своим синтаксисом и семантикой.

Синтаксис оператора есть правило его описания, которое может быть задано либо в виде общей формы записи оператора, либо в виде синтаксической диаграммы. Синтаксическая диаграмма помимо синтаксиса задает и семантику оператора, т.е. определяет те действия, которые заложены в этом операторе, и порядок выполнения этих действий. Для некоторых сложных операторов помимо синтаксической диаграммы необходимо давать дополнительные пояснения по их семантике.

Различают простые и структурные операторы. *Простым* оператором является оператор, не содержащий в себе других операторов. В простом операторе определяется, как правило, одно элементарное действие. В Паскале имеются три простых оператора: присваивания, вызова процедуры и перехода. *Структурные* операторы подразделяются, в свою очередь, на составные, условные, цикла и операторы над записями. Структурный оператор включает в себя другие операторы (как простые, так и составные). Существует несколько способов формирования структурных операторов, о которых речь пойдет в разделе «Структурные операторы».

## 

## 1.1. Оператор присваивания и выражения

Оператор присваивания относится к простым операторам. По этому оператору переменной присваивается значение выражения. Несмотря на кажущуюся простоту оператора присваивания, при его выполнении осуществляется целый набор элементарных действий:

* переменные, находящиеся в выражении, получают свои значения;
* вычисляется значение выражения;
* переменной слева от знака присваивания «:=» присваивается полученное значение.

В простейшем случае, когда выражение задано константой или другой переменной, вычислений не производится и переменная сразу получает свое значение, например:

RAZN := A – 3.5;

N := 25; C := D; Y := 'программа';

L := true; P := X > 10.

В языке Паскаль существует несколько типов выражений: арифметические, литерные, логические (булевские). В этом пункте мы рассмотрим только арифметические выражения.

Арифметические выражения складываются из констант, переменных, стандартных функций с использованием скобок и знаков операций. В Паскале определены следующие операции над числами: \* , / , + , – , DIV, MOD, где DIV – деление нацело; MOD – вычисление остатка от деления. Приоритеты:

\*, /, DIV, MOD – высший;

+, – – низший.

Например:

A := 13 DIV 5; (результат: A = 2),

B := 13 MOD 5; (результат: B = 3).

Каждое арифметическое выражение может иметь типы INTEGER и REAL. Тип константы определяется самим видом константы, тип переменной задается в ее объявлении.

Тип арифметического выражения определяется по следующему правилу:

1. Для операций «\*, +, -» результат имеет тип REAL, если хотя бы один из операндов имеет тип REAL. Если оба операнда типа INTEGER, то и результат имеет тип INTEGER.
2. Для «/» результат всегда имеет тип REAL.
3. Для «DIV, MOD» операнды и результат имеют тип INTEGER.

Значение переменной интервального типа, образованной на основе INTEGER, всегда имеет тип INTEGER. При использовании оператора присваивания нужно соблюдать типизацию объектов слева и справа от знака «:=». Смешение типов недопустимо за исключением случая, когда слева от знака «:=» стоит тип REAL, а справа – тип INTEGER.

## 1.2. Операторы вызова процедур. Ввод/вывод данных

Оператор вызова процедуры определяет активизацию процедуры, обозначенную с помощью идентификатора (имени) процедуры. Другими словами, с помощью операторов этого типа осуществляется вызов процедур с указанием в них входных и выходных параметров (подробнее об этом будет сказано в разделе «Процедуры»). Мы начнем знакомство с операторами-процедурами на базе организации ввода/вывода данных в языке Паскаль.

Для организации ввода и вывода данных используются следующие встроенные (машинные) процедуры: READ, WRITE, READLN, WRITELN.

### 1.2.1. Процедуры ввода READ и READLN

Процедура READ вызывается с помощью соответствующего оператора процедуры.

**Общая форма записи оператора**

READ (X,Y, ... , Z), где X,Y, ... , Z – переменные, называемые

списком ввода.

При выполнении процедуры READ работа программы приостанавливается, ЭВМ ждет ввода данных. Пользователь должен с клавиатуры ввести значения переменных, указанных в списке, отделяя их одним пробелом. Ввод завершается нажатием клавиши ENTER. Можно нажимать клавишу ввода и после набора каждого элемента ввода. В этом случае каждое нажатие клавиши ENTER осуществляет присваивание очередной переменной списка ввода ее значения, набранного с клавиатуры. По завершении ввода программа возобновляет свою работу.

Для лучшего понимания работы данной процедуры и ее умелого использования при задании значений нескольких переменных необходимо знать, что при вводе значений переменных (констант) с клавиатуры они сначала идут в буфер клавиатуры, а потом считываются в ячейки оперативной памяти, отведенные компилятором этим переменным. При считывании буфер очищается по принципу очереди (первым зашел – первым вышел). Это означает, что при вводе сразу нескольких констант и при последующем нажатии клавиши ENTER из буфера клавиатуры будет считано столько констант, сколько переменных в операторе READ, а остальные останутся в буфере. Если же в буфере клавиатуры после очередного считывания останутся еще константы, то при следующем операторе READ остановки работы программы не будет, и его переменные получат свои значения из буфера (если только в нем достаточно констант для всех переменных).

Например, пусть имеется фрагмент программы, включающий в себя два оператора READ:

......................

READ (A, B, C);

…………….

READ (D, E);

…………….

и пусть по первому оператору READ на клавиатуре набрано 5 констант. Тогда при работе второго READ останова работы программы не будет и переменные C и D получат значения последних двух ранее введенных констант. Если же ввести 4 константы, то второй оператор READ затребует еще одну константу с клавиатуры.

Вызов процедуры READLN имеет тот же синтаксис, что и оператор READ, однако ее работа отличается от работы первой процедуры. При однократном вводе констант отличий нет, а при одноразовом вводе нескольких констант происходит очистка буфера клавиатуры. Так, если в нашем примере заменить первый READ на READLN и тоже ввести сразу 5 констант, то второй оператор READ произведет остановку работы программы и затребует повторного ввода последних двух значений для переменных D и E. Заметим также, что оператор READLN используется преимущественно при вводе текстовых констант (READLN – read line – читать текст).

### 1.2.2. Процедуры вывода WRITE и WRITELN

Процедуры вывода WRITE и WRITELN служат для вывода на экран констант (как числовых, так и текстовых), значений переменных и выражений. Они вызываются с помощью одноименных операторов вызова процедур, например:

WRITE ('программа', X, Y – Z \* 3).

По этому оператору на экран будет выведено в одной строке слово «программа» и далее без пробелов значения переменной X и выражения Y – Z \* 3. Например, если имеем X = -3, Y = -5, Z = 12, то на экран будет выведено: программа-3-41.

Чтобы отделить элементы вывода друг от друга, используется прием форматирования вывода. Так, WRITE (А:20) – одиночное форматирование – показывает, что значению переменной А отводится 20 позиций на экране монитора. Если в значение переменной А входит менее 20 символов, то они сдвигаются вправо, а слева строка заполняется пробелами.

Двойное форматирование используется только для вывода вещественных значений. Например, WRITE (C:17:7) означает, что для вывода значения переменной C отведено всего 17 позиций, из них 7 позиций предназначены для представления дробной части. Если формат не указан, то вещественные константы выводятся на экран в экспоненциальной форме. Заметим также, что форматировать в операторах WRITE можно не только переменные, но и выражения, например:

WRITE (cos (x + 4) : 5 : 2);

Работа оператора WRITE отличается от работы оператора WRITELN тем, что по завершении вывода у WRITE курсор остается в конце списка вывода, а у WRITELN он переходит на следующую строку. Часто используют оператор WRITELN без списка вывода для вывода на экран пустой строки.

Проиллюстрируем работу этих операторов на следующем примере:

program AVERAGE;

var FIRST, SECOND, TROIS, SUM: integer;

begin

writeln ('Введите 3 числа ');

readln (FIRST, SECOND, TROIS);

SUM := FIRST + SECOND + TROIS;

writeln ('Среднее значение ', FIRST:4,',',SECOND:4,',');

write (TROIS:4,' равно ';(SUM div 3):3)

end.

На экран будет выведено:

|  |
| --- |
| Введите 3 числа  2 12 9  Среднее значение 3, 12,  9 равно 8 |

## Лабораторная работа № 1

**Цель работы:** выработать практические навыки использования системы программирования Borland Pascal или Delphi, научиться создавать, вводить, отправлять на выполнение и исправлять простейшие программы на языке Паскаль, познакомиться с диагностическими сообщениями компилятора об ошибках на примере программ, реализующих линейные алгоритмы.

**Общие сведения**

Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых значениях исходных данных. Операторы программы выполняются последовательно, один за другим, в соответствии с их расположением в программе.

Перед выполнением работы необходимо ознакомиться с теоретическим материалом по темам «Описание языка Паскаль», «Простые операторы. Ввод/вывод данных».

**Пример.** Определить расстояние на плоскости между двумя точками с заданными координатами M1(x1,y1) и M2(x2,y2).

***Решение задачи.***

В этом примере проведем полный разбор решения задачи.

Математическая модель: расстояние на плоскости между двумя точками M1(x1,y1) и M2(x2,y2) высчитывается по формуле:



Составим блок-схему алгоритма, а затем уточним содержимое блоков "Вычисление расстояния" и "Вывод расстояния" (см. рис.1):



Рис. 1. Блок-схема алгоритма

Дальнейшая детализация не требуется. Переводим блок-схему на язык Паскаль, доработав программу, чтобы улучшить ее интерфейс:

program example1;

var x1, x2, y1, y2: Integer;

d:Real;

begin

[Writeln](http://valera.asf.ru/delphi/help/name.php?name=writeln)('Эта программа вычисляет расстояние между двумя точками на плоскости');

[Writeln](http://valera.asf.ru/delphi/help/name.php?name=writeln)('Введите координаты двух точек:');

[Write](http://valera.asf.ru/delphi/help/name.php?name=write)('x1= '); [Readln](http://valera.asf.ru/delphi/help/name.php?name=readln)(x1);

[Write](http://valera.asf.ru/delphi/help/name.php?name=write)('y1= '); [Readln](http://valera.asf.ru/delphi/help/name.php?name=readln)(y1);

[Write](http://valera.asf.ru/delphi/help/name.php?name=write)('x2= '); [Readln](http://valera.asf.ru/delphi/help/name.php?name=readln)(x2);

[Write](http://valera.asf.ru/delphi/help/name.php?name=write)('y2 ='); [Readln](http://valera.asf.ru/delphi/help/name.php?name=readln)(y2);

d:=[sqrt](http://valera.asf.ru/delphi/help/name.php?name=sqrt)([sqr](http://valera.asf.ru/delphi/help/name.php?name=sqr)(x2-x1)+[sqr](http://valera.asf.ru/delphi/help/name.php?name=sqr)(y2-y1));

[Writeln](http://valera.asf.ru/delphi/help/name.php?name=writeln)('d= ',d:6:2);

[Writeln](http://valera.asf.ru/delphi/help/name.php?name=writeln)('нажмите Enter для завершения работы программы');

[Readln](http://valera.asf.ru/delphi/help/name.php?name=readln);

end.

**Варианты заданий**

**Задание 1.** Даны *x*, *y*. Составить программу вычисления значения выражения:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a) |  | b) |  | c) |  | d) |  |

**Задание 2.** Составить программу для решения следующей задачи:

1. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
2. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
3. Вычислить высоту треугольника, опущенную на сторону *а*, по известным значениям длин его сторон *a, b, c*.
4. По данным сторонам прямоугольника вычислить его периметр, площадь и длину диагонали.

**Задание 3.** Вывести значение *true*, если приведенное высказывание для предложенных исходных данных является истинным, и значение *false* в противном случае (все числа, для которых не указано иное, являются действительными):

1. данное число *x* принадлежит отрезку [*-a, a*];
2. данное число *x* не принадлежит интервалу (*a, b*);
3. данное целое число *x* является нечетным;
4. данное число *x* является корнем уравнения: ax2+bx+c=0;

**Дополнительные задания**

1. Ученик начал решать задачи данного урока программирования, когда электронные часы показывали *h1* часов и *min1* минут, а закончил, когда было *h2* часов и *min2* минут. Составьте программу, позволяющую определить, сколько времени (в часах и минутах) ученик решал эти задачи.
2. Дано действительное число а. Не пользуясь никакими другими операциями, кроме умножения, получить: а) а4 за две операции; б) а6 за три операции; в) а7 за четыре операции; г) а8 за три операции.

**Контрольные вопросы**

1. Каковы назначение и возможности системы программирования?
2. Как запустить программу на трансляцию и выполнение?
3. Как обозначается начало и конец программы?
4. Из каких разделов состоит программа на языке Паскаль?
5. Как в языке Паскаль осуществляется вывод на экран?
6. Для чего предназначен оператор присваивания?
7. Как вывести на экран значение переменной?

# 2. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Структурные операторы строятся из других операторов по определенным правилам. Операторы, входящие в структурный оператор, выполняются последовательно – в составных операторах и операторах над записями, альтернативно – в условных операторах, многократно – в операторах цикла.

## 

## 2.1. Составной и пустой операторы

При формировании структурных операторов существуют некоторые ограничения на число входящих в него операторов. В частности, в операторе выбора IF (в школьном алгоритмическом языке команда «если») после служебного слова THEN (аналог – «то») может стоять только один оператор. Поэтому в Паскале возникла необходимость группирования операторов в единое целое – в один составной оператор.

Любая группа операторов, размещенных между словами BEGIN и END (иначе, операторные скобки), рассматривается как один – составной оператор. При выполнении составного оператора все его компоненты (операторы) выполняются в порядке их написания (линейно).

Составные операторы обрабатываются как единое целое, как один оператор, что имеет значение там, где синтаксис языка допускает использование только одного оператора. Это имеет место практически во всех структурных операторах. Примеры использования составных операторов будут рассмотрены в соответствующих разделах данного пособия.

Наряду с понятием «составной оператор» в языке существует специфическое понятие «пустой оператор». *Пустой оператор* – это оператор, который не предусматривает выполнения никаких действий.Зачем он нужен? Действительно, если оператор не выполняет никаких действий, то стоит ли его писать? Однако практика показывает, что иногда полезно иметь такое средство, например, при выполнении искусственной задержки выполнения программы:

FOR I := 1 TO 10 000 DO;

При выполнении данного оператора машина переменной I последовательно присвоит значения от 1 до 10 000. В теле цикла нет операторов, значит, кроме увеличения значений переменной на 1 ничего не будет выполнено, однако время на это затрачивается, и, следовательно, некоторое время программа «висит» на данном операторе.

Существуют и другие примеры использования пустого оператора, когда по синтаксису оператор формально необходим, но никаких действий внутри него не производится.

## 

## 2.2. Организация ветвлений. Операторы выбора

В языке Паскаль алгоритмическая базовая конструкция выбора может быть реализована с помощью двух структурных операторов – IF и CASE, называемых *операторами выбора*. С их помощью можно выбрать для выполнения один из составных операторов (или ни одного оператора).

### 2.2.1. Оператор ветвления IF

Оператор IF можно представить в общей форме записи как

IF <Условие> THEN <Оператор 1> ELSE <Оператор 2>,

где конструкция «Условие» есть логическое выражение, которое принимает два значения типа BOOLEAN: TRUE, FALSE (истинно или ложно).

Само логическое выражение складывается из операций сравнения: >, >=, <, <=, =, <>. Результат сравнения может быть TRUE или FALSE.

Логические выражения могут формироваться также и с помощью трех логических операций: NOT, AND, OR. Приоритеты всех используемых в Паскале операций таковы:

**Высший**: ( )

NOT \*, /, DIV, MOD

AND

OR +, -

**Низший**: >, =, < , >=, <>, <=

В качестве условия может быть использована и логическая переменная, например:

I and J or K ---> (I and J) or K;

not X and Y ---> (not X) and Y, где I, J, K, X, Y – переменные типа BOOLEAN;

(A < B) or (B = 0), где A, B – переменные простого типа.

В операторе IF всегда за словами THEN и ELSE должен следовать один оператор. Этим оператором может быть не только оператор присваивания, но и любой другой. Если хотя бы один из них является снова оператором IF, то полученную конструкцию называют вложением.

П р и м е р :

IF <условие1> THEN <ветвь 1>

ELSE IF <условие2> THEN <ветвь 2>

ELSE <ветвь 3>;

Такое вложение используется для уменьшения числа необходимых проверок. Этот метод часто обеспечивает большую эффективность, чем составное условие, однако одновременно он уменьшает надежность программы. Не рекомендуется использовать более двух-трех уровней вложения IF. Вложения могут идти и после слова THEN.

Первый способ предпочтительнее, чем второй, так как конструкция THEN-IF менее удобна, чем ELSE-IF. С помощью конструкции ELSE-IF чаще всего осуществляется выбор одного из нескольких альтернативных вариантов. Заметим, однако, что иногда такое вложение можно заменить на последовательность операторов короткой формы IF-THEN. Это видно на следующем примере:

program QUARD;

var A, B, C : real; D : real;

begin

read (A, B, C); D := sqr (B) – 4 \* A - C;

|  |  |  |
| --- | --- | --- |
| 1-й вариант | 2-й вариант | |
| if D < 0 then write ('Не имеет корней'); | if D < 0 then write ('Нет корней')  else if D = 0 then write ('Один корень') | |
| if D = 0 then write ('Один корень');  if D > 0 then write ('Два корня'); | else write ('Два корня'); | |
|  |  |

end.

Рис. 2. Пример программы двух разных вложений

Однако в данном примере 2-й вариант более эффективен, так как имеет на одно сравнение меньше, и в случае D < 0 сразу же дает ответ, не делая последующих проверок.

В Паскале существует специальный оператор, позволяющий сделать выбор из нескольких альтернатив – множественное ветвление, или оператор варианта.

### 2.2.2. Оператор варианта CASE

Оператор варианта состоит из выражения и списка операторов, каждому из которых предшествует одна или более констант, называемых *константами выбора.*

**Общая форма записи**

CASE <выражение> OF

константы: оператор;

....................

константы: оператор

ELSE <оператор>

END;

Выражение, стоящее между CASE и OF, называется *селектором*. Константы (значения выражения), предшествующие двоеточию, называются *метками случаев*. Порядок работы оператора: сначала вычисляется значение селектора, затем выполняется оператор, одна из меток которого совпадает со значением селектора. Все остальные операторы не выполняются, и управление передается следующему после END оператору. В случае короткой формы оператора CASE при несовпадении значения селектора (ключа) ни с одной из констант из списка никакой оператор не подлежит исполнению. Если же в операторе есть строка ELSE, то при несовпадении значения селектора ни с одной константой выполняется оператор, следующий за ELSE.

Выражение «селектор» может относиться к любому скалярному типу, кроме REAL. Метки случаев должны принадлежать тому же типу, что и селектор. Нежелательно, чтобы одна и та же метка появлялась более одного раза в операторе CASE. Если же это произойдет (компилятор не проверяет повторяемость меток), то выполнится тот оператор, который соответствует первому вхождению метки в список констант.

Оператор CASE особенно удобно использовать, во-первых, когда характер действий определяется значениями, которые не попадают в последовательно расположенные интервалы, во-вторых, когда нескольким дискретным значениям селектора соответствует одно и то же действие.

П р и м е р 1. Печать названия десятичных цифр.

program DICITS;

var DIGIT: integer;

begin

writeln ('Введите цифру');

readln (DIGIT);

case DIGIT of

0: writeln ('нуль');

1: writeln ('один');

..................

9: writeln ('девять');

else writeln ('это не цифра');

end;

end.

П р и м е р 2. Печать номера квартала года.

program NUMKVART;

var MESIATZ: 1..12;

begin

write ('Введите номер месяца года – ');

read (MESIATZ);

case MESIATZ of

1, 2, 3: writeln ('Первый квартал');

4, 5, 6: writeln ('Второй квартал');

7, 8, 9: writeln ('Третий квартал');

10, 11, 12: writeln ('Четвертый квартал');

end;

end.

*Примечание*. В операторе CASE формально нет условий как таковых, однако проверка условий осуществляется в неявном виде на предмет совпадения константы со значением селектора.

## Лабораторная работа №2

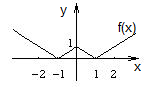
**Цель работы:** научиться решать задачи на разветвляющиеся алгоритмы; научиться использовать в программах условный оператор *if* и оператор выбора *case*.

**Общие сведения**

Алгоритм называется разветвляющимся, если он содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход вычислительного процесса на ту или иную ветвь алгоритма определяется исходными данными задачи.

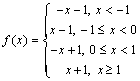
Перед выполнением работы необходимо ознакомиться с правилами записи логических выражений, операций сравнения, операторов *if, case*.

**Пример.** Дано действительное *x*. Для функции *f*, график которой представлен на рисунке, вычислить *f(x)*.



***Решение задачи.***

Математическая модель: функция вычисляется по следующей формуле:



Составим схему алгоритма, детализировав все блоки (рис. 2).

Дальнейшая детализация не требуется. Переводим алгоритм на язык Паскаль.

Program example1;

var x, f:Real;

begin

[Write](http://valera.asf.ru/delphi/help/name.php?name=write)('Введите x: '); [Readln](http://valera.asf.ru/delphi/help/name.php?name=readln)(x);

if x<-1 then f:= -x-1 else

if (x>=-1) and (x<0) then f:= x-1 else

if (x>=0) and (x<1) then f:= -x+1 else f:= x+1;

[Writeln](http://valera.asf.ru/delphi/help/name.php?name=writeln)('F= ',f:6:2);

Readln;

end.



Рис. 3. Блок-схема ветвления в 4-х направлениях

**Варианты заданий**

**Задание 1**. Используя оператор *if,* вычислить заданное выражение для данных типа *Integer:*

а) b)

c)  d)

**Задание 2.** Найти алгоритм решения задачи и реализовать его с помощью оператора (операторов) if-then-else:

1. Составить программу, реализующую эпизод сказки: машина спрашивает, куда пойдет герой, и в зависимости от ответа (налево – (-1), прямо – 0, направо – 1), печатает, что произойдет с героем.
2. Морской бой. Машина задумывает два числа от 0 до 9. Игрок пытается их угадать, вводя свои два числа. Если они совпали (в любом сочетании), то игрок выиграл.
3. В Атлантическом океане терпит бедствие пассажирский теплоход «Посудина». Все пассажиры будут спасены, если на помощь успеют два судна. Судно продержится на плаву *t* часов. Скорость судов-спасателей 40 узлов. Составить программу, определяющую спасутся ли пассажиры. Известны расстояния от трех судов-спасателей до тонущего судна.
4. Через старый мост движется поток автомашин. Одновременно на мосту могут находиться 3 машины. Если на мост въедут 3 легковых или 2 легковых и грузовик – мост выдержит. Если 2 грузовика и легковая или 3 грузовика – рухнет.

**Задание 3.** Используя оператор выбора, составить программы решения следующих задач.

1. По номеру дня недели вывести на печать рабочий это день или выходной, считая выходными субботу и воскресенье.
2. По номеру месяца указать, к какому времени года он относится.
3. По номеру месяца вывести на печать количество дней в нем.
4. Единицы массы пронумерованы следующим образом: 1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна. Дан номер единицы массы и масса тела *M* в этих единицах (*M* - вещественное число). Вывести массу данного тела в килограммах.

**Дополнительные задания**

1. Даны действительные числа *a, b, c, x, y*. Выяснить, пройдет ли кирпич с ребрами *a,b,c* в прямоугольное отверстие со сторонами *x* и *y*. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.
2. Сможет ли шар радиуса *R* пройти в ромбообразное отверстие со стороной *P* и острым углом *Q*?

**Контрольные вопросы**

1. Какие операторы используются для программирования развилок?
2. Как выполняются операторы условного перехода?
3. Какую из функций: *Sin(x), Abs(x), Trunc(x)* можно заменить условным оператором *if x<0 then x := -x*?
4. Если выбор вариантов осуществляется из конечного числа элементов выбора, то лучше взять для этого оператор *if* или *case*?
5. Как заменить оператор case операторами *if*?
6. В чем преимущество оператора *case* от последовательности «коротких» операторов *if*?
7. Какой тип переменной можно использовать в качестве ключа оператора *case*?

# 3. ОРГАНИЗАЦИЯ ЦИКЛОВ

Оператор цикла задает повторное выполнение определенных операторов. Для реализации циклов в Паскале предусмотрены три различных структурных оператора: WHILE, REPEAT, FOR. Первые два используются, если число повторений (итераций) заранее не определено, но известно условие завершения цикла. Оператор FOR применяется тогда, когда число повторений тела цикла известно заранее.

## 3.1. Оператор WHILE

Этот оператор является наиболее мощным из всех трех, реализующих циклы. Два других оператора можно выразить с его помощью. Общий этого оператора:

**WHILE <Условие> DO <Тело цикла>;**

Логическое выражение, стоящее после WHILE, называется *условием возобновления цикла* и должно иметь булевский тип. Оператор, следующий за DO, является *телом цикла*. Он повторяется до тех пор, пока истинно условие возобновления цикла. Как только условие возобновления цикла становится ложным, управление переходит к оператору, стоящему за WHILE. Если условие возобновления не удовлетворяется до начала выполнения цикла, то тело цикла пропускается.

Из указанного описания видно, что оператор WHILE реализует базовую структуру «цикл-пока», так как здесь проверка условия идет до тела цикла. Поэтому оператор WHILE называют *оператором цикла с предусловием*.

Оператор

П р и м е р . Даны числа A, B (A > 1). Получить все степени числа A, меньшие числа B.

program STEPENI;

var A, B, C: real;

begin

readln (A, B); C := A;

while C < B do

begin

writeln (C);

C := C\*A;

end;

end.

*Примечание*. Грамотное использование оператора WHILE предполагает умение правильно написать условие возобновления цикла. Здесь надо иметь в виду следующие рекомендации:

1. В условии, как в логическом выражении, должны обязательно фигурировать переменные, изменяющие свои значения в теле цикла.
2. Во избежание зацикливания лучше сначала написать условие прекращения цикла и взять потом в операторе его отрицание.
3. Переменные логического выражения должны получить свои исходные значения до входа в оператор WHILE.

## 3.2. Оператор REPEAT

Оператор REPEAT называют оператором цикла с постусловием, так как здесь выражение, управляющее повторным выполнением последовательности операторов, помещается после тела цикла:

**REPEAT <Тело цикла> UNTIL <Условие>;**

Из общего вида оператора видно, что в этом операторе не обязательно использовать для тела цикла операторные скобки. Здесь ключевые слова REPEAT и UNTIL сами играют роль этих скобок.

В этом операторе тело цикла выполняется до тех пор, пока ложно условие, стоящее после UNTIL. Условием выхода из цикла является истинность выражения. Мы видим, что это есть форма «цикла-до».

П р и м е р . Даны числа A, B (A > 1). Получить все степени числа A, меньшие числа B.

program STEPENI;

var A, B, C: real;

begin

readln (A, B); C := A;

repeat

writeln (C);

C := C\*A;

until C >= B;

end.

*Примечание*. Между операторами WHILE и REPEAT существует три основных различия:

1. В операторе REPEAT проверка условия выхода из цикла выполняется в конце, а не в начале цикла, как в операторе WHILE, поэтому в операторе REPEAT тело цикла выполняется хотя бы один раз.
2. В REPEAT выход из цикла осуществляется по истинности условия, а в WHILE – по ложности.
3. В операторе WHILE тело цикла чаще всего имеет форму составного оператора, в операторе REPEAT для организации тела цикла операторные скобки не нужны.

## 3.3. Оператор FOR

Оператор FOR предназначен для организации циклов, когда заранее известно, сколько раз должно повториться тело цикла. Здесь управление числом повторений осуществляется с помощью специальной переменной – параметра цикла (управляющей переменной), которой присваивается возрастающая (убывающая) последовательность значений. Оператор FOR имеет следующий вид:

**FOR <Переменная>:= <Выражение 1> TO <Выражение 2> DO;**

**FOR<Переменная>:=<Выражение1>DOWNTO<Выражение1>DO;**

Здесь «Переменная» есть параметр цикла, «Выражение 1» – начальное значение параметра, «Выражение 2» – его конечное значение. В качестве управляющей переменной должна быть переменная, объявленная локальной в блоке, который содержит данный оператор FOR. Управляющая переменная должна иметь ординальный тип. Начальное и конечное значения имеют тип, совместимый с типом параметра цикла.

Когда начинает выполняться оператор FOR, начальное и конечное значения определяются один раз, и эти значения сохраняются на протяжении всего выполнения оператора.

Оператор, который содержится в теле цикла, выполняется один раз для каждого значения управляющей переменной в диапазоне между начальным и конечным значениями. Управляющая переменная всегда инициализируется начальным значением. Она принимает все свои значения из диапазона с шагом 1, если TO, и с шагом -1, если DOWNTO.

В случае TO, если начальное значение превышает конечное, тело цикла не выполняется.

Для случая DOWNTO это имеет место, когда начальное значение меньше, чем конечное. Отсюда заключаем, что оператор цикла FOR реализует, как и WHILE, схему цикла «пока» – проверка условия повторения цикла идет до тела цикла.

*Примечание*.

1. Если тело цикла в этом операторе состоит из более одного оператора, то они все заключаются в операторные скобки (реализуют конструкцию составного оператора).
2. В отличие от школьного алгоритмического языка, оператор FOR нельзя прервать путем присваивания управляющей переменной ее конечного значения. Изменения переменной цикла не влияют на число повторений тела цикла.
3. После выполнения оператора значение управляющей переменной становится неопределенным, если только выполнение оператора FOR не было прервано с помощью оператора перехода.

Рассмотрим примеры использования оператора FOR для организации циклических процессов.

П р и м е р 1. Печать отсчета цифр при старте.

program START;

var SEC: integer;

begin

writeln ('До старта осталось ...');

for SEC := 10 downto 1 do

writeln (SEC:4);

writeln ('ноль'); writeln ('Старт !!')

end.

В данном примере управляющая переменная SEC принимает значения типа INTEGER, однако в Паскале она определена как переменная ординального типа и, следовательно, может принимать значения типа CHAR или принадлежать перечислимому типу, как показано в примере 2.

П р и м е р 2 . Подсчет числа часов рабочей недели.

program WORKTIME;

type DAYS = (MO, TU, WE, TH, FR, SA, SU);

var DEN: DAYS; WT: integer;

begin

WT := 0;

for DEN := MO to SA do

if DEN <> SA then WT := WT + 8

else WT := WT + 7; writeln (WT);

end.

## Лабораторная работа № 3

**Цель работы:** закрепить практические навыки работы с системой программирования, научиться составлять программы с использованием циклических структур и выбирать для этого нужный оператор цикла.

**Общие сведения**

Алгоритм называется циклическим, если он содержит многократное выполнение одних и тех же операторов при различных значениях промежуточных данных.

Перед выполнением работы необходимо изучить различные схемы организации циклов и операторы *for, while, repeat .*

В этом разделе и последующих мы не будем приводить подробное решение задач, а ограничимся текстами программ с пояснениями.

**Пример**: На промежутке от 1 до *M* найти все числа Армстронга. Натуральное число из *n* цифр называется числом Армстронга, если сумма его цифр, возведенных в степень *n*, равна самому числу. Например, число 153 (153=13+53+33).

Решение. После организации ввода данных программа будет содержать цикл с параметром *i* (от 1 до *М*) с двумя вложенными циклами. Первый предназначен для подсчета количества цифр *n*, второй – для вычисления суммы *s* степеней цифр числа *i*. Если числа *i* и *s* равны, то *i* – число Армстронга, его необходимо вывести на экран.

PROGRAM Primer\_1;

var i,k,s,p,n,M: Integer;

begin

Write('Введите M '); Readln(M);

for i:=1 to M do

begin

s:=0; k:=i; n:=0;

while k<>0 do

begin k:=k div 10; n:=n+1 end;

k:=i;

While k<>0 do

begin p:=k mod 10; k:=k div 10;

if p<>0 then s:=s+ Round(Exp(n\*Ln(p)))

end;

if s=i then Writeln(i);

end;

Readln;

end.

**Варианты заданий**

**Задание 1.** Целочисленная арифметика.

Найти количество натуральных двузначных чисел, каждое из которых делится на 3 и на 13.

1. Найти количество натуральных четырехзначных чисел, каждое из которых не делится ни на 2, ни на 3.
2. Найти количество натуральных чисел, не превосходящих 1000, каждое из которых кратно 25 и не кратно 3.
3. Найти те натуральные числа, не превосходящие *x*, которые при делении на 10 дают в остатке 5.

**Задание 2.** Найти алгоритм решения задачи и реализовать его в виде Паскаль-программы.

1. Начальный вклад в банк составляет *а* рублей. Через сколько лет он станет больше *b* рублей? Каждый год вклад увеличивается на 3%.
2. Ежегодный прирост рыбы в пруду составляет 15%. Запасы рыбы оценены в *А* тонн. Ежегодный план отлова *В* тонн. Подсчитать, сколько лет можно выдерживать заданный план?
3. Каждая бактерия делится на две в течение одной минуты. В начальный момент имеется *A* бактерий. Сколько времени потребуется, чтобы количество бактерий превзошло *X*?
4. Определить количество посетителей салона, которых успеет обслужить мастер-стилист, если его рабочий день составляет *t* часов и известна продолжительность (в минутах) обслуживания каждого посетителя очереди (вводится пользователем).

**Задание 3.**Составить программу для решения следующей задачи:

* 1. Вычислить количество точек с целочисленными координатами, попадающими в круг радиуса R (R>0) с центром в начале координат.
  2. Найти все натуральные числа от 1 до N, представимые в виде суммы кубов двух натуральных чисел.
  3. Найти все натуральные числа от 1 до N, представимые в виде суммы квадратов трех натуральных чисел.
  4. Даны натуральные M, N (M<N). Найти числитель и знаменатель несократимой правильной дроби p/q такой, что p/q = m/n.

**Контрольные вопросы**

1. Как записывается и как работает оператор *for*?
2. Для организации каких циклов применим оператор *for*?
3. В чем отличие оператора *while* от оператора *repeat*?
4. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?
5. Напишите пример оператора цикла, который не выполняется ни разу.
6. С какими ограничениями реализована конструкция цикла со счетчиком?
7. Замените оператор "*repeat A until B*" равносильным фрагментом программы с оператором *while*.

# ОРГАНИЗАЦИЯ ПОДПРОГРАММ. ПРОЦЕДУРЫ И ФУНКЦИИ. ПРОЦЕДУРЫ И ФУНКЦИИ

Мы уже видели, что Паскаль-программа состоит из последовательности операторов (простых и структурных). Есть в языке также понятие функции. Система TurboPascal и другие системы программирования на базе языка Паскаль имеют целый набор встроенных машинных функций. Однако Паскаль предоставляет возможность создавать новые операторы и функции на основе стандартных. Такие процедуры и функции принято называть пользовательскими.

В Паскале два типа подпрограмм – процедуры и функции. Создание процедур и функций является дальнейшим развитием идеи группирования, когда несколько операторов объединяются в программный блок, который имеет свое имя и к которому можно обратиться по этому имени с помощью соответствующих средств вызова. Процедура вызывается с помощью оператора вызова процедуры с одноименным именем. Функция пользователя также имеет свое имя, и ее активизация происходит при вычислении выражения, содержащего имя функции. Возвращаемое функцией значение подставляется в это выражение.

## 

## 4.1. Процедуры и их типизация

Итак, процедура – это часть программы (подпрограмма), имеющая имя и предназначенная для решения некоторой частной задачи (подзадачи). Процедуры делятся по способам описания и обращения к ним.

*Процедура встроенная* (*машинная*) – это процедура, описание которой считается известной транслятору, в связи с чем ее можно использовать в программе, зная только ее имя.

*Процедура пользователя* – процедура, которую создает (описывает) программист на основе имеющихся операторов и встроенных процедур и функций данного языка по определенным правилам данного языка.

*Процедура без параметров* – процедура, при обращении к которой не требуется задания начальных установок, значений и после выполнения которой в основную программу не передаются результаты работы данной процедуры.

*Процедура с параметрами-значениями* – процедура, при обращении к которой требуются только начальные значения. На выходе данные не передаются в основную программу.

*Процедура с параметрами-переменными* – процедура, не требующая начальных значений, однако передающая в основную программу результаты своей работы (передает значения некоторых переменных).

*Комбинированная процедура* – процедура, имеющая параметры-переменные и параметры-значения, т.е. входные и выходные данные.

### 4.1.1. Встроенные процедуры

Встроенные процедуры являются составной частью системы программирования. Среди них есть стандартные процедуры, которыми можно пользоваться в любом месте программы без какого-либо предварительного объявления. Сюда относятся уже ранее упомянутые процедуры ввода/вывода, управления работой программы, динамического распределения памяти, строковые процедуры и пр. Полный перечень встроенных процедур можно найти в справочнике для языка.

Помимо стандартных процедур в Паскале есть также стандартные модули, представленные в виде TPU – файлов, каждый из которых содержит в себе целый набор процедур и функций. Для того чтобы использовать процедуры из модулей, необходимо вызвать нужный модуль в разделе USES. Система TurboPascal имеет модули PRINTER, DOS, CRT, GRAPH и др.

CRT позволяет использовать все возможности дисплея и клавиатуры, включая управление режимом работы экрана, расширенные коды клавиатуры, цвет, окна и звуковые сигналы.

DOS поддерживает различные функции ДОС, включая установку и получение текущего значения даты и времени, поиск по каталогам файлов и выполнение программ.

PRINTER позволяет легко организовать доступ к устройству печати.

GRAPH – мощный графический пакет с набором процедур и функций обработки графических объектов (точек, отрезков, прямоугольников, окружностей и пр.).

Рассмотрим несколько примеров встроенных процедур:

* CLRSCR – процедура очистки экрана. Результатом работы является стирание всей информации с экрана. Данная процедура является примером процедур без параметров.
* GOTOXY (A, B) – процедура позиционирования курсора на экране дисплея в точку с координатами (A, B). A и B являются входными данными, следовательно, это пример процедуры с параметрами-значениями.
* WRITE (A, B, ..., Q) – процедура вывода информации на экран дисплея. Данная процедура – процедура с параметрами-значениями.
* READ (A, B, ..., Q) – процедура ввода информации в ЭВМ. При выполнении данной процедуры переменным A, B, ..., Q присваиваются конкретные значения, т.е. данные передаются в основную программу, и, значит, процедура является примером процедур с параметрами-переменными.

### 4.1.2. Процедуры пользователя

При работе с процедурами пользователя необходимо уметь производить два вида деятельности: описание процедуры и обращение к ней в основной программе. Вызов процедуры пользователя осуществляется так же, как и вызов встроенной процедуры, – с помощью оператора вызова процедуры, имя которого совпадает с именем процедуры, с указанием списка параметров, если таковые имеются. Описание же процедуры включает в себя разработку подпрограммы и правильное оформление ее заголовка. Остановимся на нем более подробно.

В основной программе все процедуры (а также и функции) пользователя должны быть объявлены. Объявление процедур и функций осуществляется после объявления переменных и перед первым словом BEGIN программы.

Процедура, как видно из ее определения, оформляется так же, как и основная программа. Вообще процедуру нужно воспринимать как программу в миниатюре. В свою очередь, основная программа может быть легко переделана в процедуру с заменой слова PROGRAM на PROCEDURE. Если процедура объявлена, ее можно использовать в последующих частях программы, просто записывая ее имя, за которым, если необходимо, следует список параметров. Вызов процедуры для основной программы становится новым оператором. Обращение к процедуре активизирует эту процедуру, т.е. приводит к выполнению группу операторов, содержащихся в ее теле. После этого управление переходит к оператору, следующему за вызовом процедуры.

Описание процедур будем рассматривать по той же схеме, что и машинные процедуры, а именно: сначала без параметров, далее с параметрами-значениями, с параметрами-переменными и, наконец, комбинированные процедуры.

### 4.1.3. Процедуры без параметров

Заголовок процедуры без параметров выглядит как:

**PROCTDURE <Имя процедуры>;**

Вызываются такие процедуры путем написания в основной программе имени этой процедуры. В виде процедуры без параметров оформляются такие подзадачи, у которых нет входных и выходных данных, или же эти данные удобнее передавать с помощью операторов присваивания: READ и WRITE.

Рассмотрим несколько примеров, в которых представлены эти варианты.

П р и м е р 1. Нарисовать три вертикальных квадрата 3 × 3 с помощью символа «\*».

Очевидно, что в этой программе надо выделить рисование квадрата в виде процедуры без параметров, а затем трижды вызвать ее в основной программе.

program RISUNOK;

procedure KVADRAT;

begin

writeln ('\*\*\*');

writeln ('\* \*');

writeln ('\*\*\*');

end;

begin

clrscr; KVADRAT;

writeln; KVADRAT;

writeln; KVADRAT;

end.

П р и м е р 2 . Вычислить площадь четырехугольника ABCD.

Зная длины сторон четырехугольника и длину одной из его диагоналей, например BD, можно по формуле Герона найти площади двух вспомогательных треугольников и сложить их. Отсюда следует, что в программе надо выделить процедуру вычисления площади треугольника.

program PLOCHAD\_1;

var AB, BC, CD, AD, BD, S1, S, a, b, c, p:real;

procedure GERON\_1;

begin

p := (a + b + c)/2;

S := sqrt (p\*(p - a)\*(p - b)\*(p - c));

end;

begin {\*ОСНОВНАЯ ПРОГРАММА\*}

read (AB, BC, CD, AD, AC);

a := AB; b := AD; c := BD; GERON\_1; S1:= S;

a := BC; b := CD; c := BD; GERON\_1; S1:= S1 + S;

write (S1);

end.

*Примечание.* В данной программе все вычисления проходят с помощью переменных, объявленных в разделе VAR основной программы.

### 4.1.4. Процедуры с параметрами-значениями

Как было сказано ранее, процедуры с параметрами-значениями требуют входных данных. Где они записываются и как задаются? На этот вопрос может ответить общая форма заголовка процедуры этой процедуры:

**PROCEDURE <Имя процедуры> (<Параметры-аргументы>: тип);**

Здесь под параметром понимают имя переменной, которая является «входной» для процедуры (формальный параметр-аргумент). Этот параметр с синтаксической точки зрения является параметром-значением, при его описании в заголовке процедуры не требуется писать слово VAR. Параметры-значения при вызове процедуры принимают из основной программы свои конкретные значения. Заметим также, что в самой процедуре значения параметров-значений не меняются в ходе работы процедуры.

При обращении к процедуре с параметрами-значениями в основной программе фактическими параметрами могут служить как имена переменных (которые описаны и определены выше), так и конкретные значения (константы) и выражения. При обращении необходимо следить за соответствием списка параметров при обращении и описании. Кроме того, следует строго соблюдать соответствие типов параметров.

Рассмотрим работу процедур такого типа на примерах.

П р и м е р 1. Нарисовать квадрат с произвольной длиной стороны в левом верхнем углу (длина стороны задается с клавиатуры).

В этой программе также надо оформить рисование квадрата в виде процедуры, но уже с входным параметром-значением – длиной стороны квадрата.

program RISUNOK\_2;

var I: integer;

procedure KVADRAT (N: integer);

var J, K: integer;

begin

for J := 1 to N do write ('\*'); writeln;

for J := 1 to N - 2 do

begin

write ('\*'); for K := 1 to N - 2 do write (' ');

writeln ('\*');

end;

for J := 1 to N do write ('\*');

end;

begin { Основная программа }

write ('Введите длину стороны – ');

readln (I); clrscr; KVADRAT (I);

end.

П р и м е р 2. Вычислить площадь четырехугольника с применением процедуры с параметрами-значениями.

program PLOCHAD\_2;

var AB, BC, CD, AD, AC, S1, S: real;

procedure GERON\_2 (a, b, c: real);

var P: real;

begin

P := (a + b + c)/2; S := sqrt (P\*(P - a)\*(P - b)\*(P - c));

end;

begin {\*ОСНОВНАЯ ПРОГРАММА\*}

read (AB, BC, CD, AD, AC); GERON\_2(AB, BC, AC); S1:= S;

GERON\_2 (AD, AC, CD); write ('S = ', S1 + S)

end.

В данной программе определена процедура GERON\_2 с тремя параметрами-значениями и локальной переменной P. Значение же площади треугольника помещается в глобальную переменную S. При вызове этой процедуры формальные параметры a, b, c замещаются на фактические параметры AB, BC, AC при первом обращении и на AD, AC, CD – при втором.

Заметим также, что здесь фактические параметры представлены переменными, которые получают свое значение с помощью процедуры READ. Однако если известны длины сторон треугольника, например, 6, 7, 4, то можно вычислить площадь этого треугольника, вызвав процедуру GERON\_2(6, 7, 4), и получить ответ в переменной S.

### 4.1.5. Процедуры с параметрами-переменными

В отличие от процедур с параметрами-значениями, процедуры с параметрами-переменными не имеют входных параметров, т.е. из основной программы не передаются значения переменных в процедуру, за исключением глобальных переменных. Отличие в описании и обращении к процедурам с параметрами-переменными заключается в специфическом написании заголовка процедуры. В остальном все процедуры схожи. Общий вид заголовка процедуры с параметрами-переменными выглядит так:

**PROCEDURE <Имя процедуры> ( VAR<Параметры-переменные>: тип);**

При детальном ознакомлении с синтаксической диаграммой видно, что параметрам-переменным должно предшествовать служебное слово VAR, причем оно пишется столько раз, сколько различных типов представлено в выходных данных, например:

PROCEDURE PRIMER (VAR a, b, c : INTEGER; VAR m : CHAR; VAR i, j : REAL).

При обращении к процедурам с параметрами-переменными фактическими параметрами должны являться имена переменных, которые описаны в основной программе.

П р и м е р . Вычисление площади четырехугольника.

program PLOCHAD\_3;

var AB, BC, CD, AD, AC, S1, S2, a, b, c: real;

procedure GERON\_3 (var S: real);

var P: real;

begin

P := (a + b + c)/2; S := sqrt (P\*(P - a)\*(P - b)\*(P - c));

end;

begin { Основная программа }

read (AB, BC, CD, AD, AC);

a := AB; b := BC; c := AC; GERON\_3 (S1);

a := AD; b := AC; c := CD; GERON\_3 (S2);

write ('S = ', S1 + S2)

end.

### 4.1.6. Комбинированные процедуры

Комбинированные процедуры включают в себя входные и выходные данные. В заголовке этой процедуры, как и ранее, выходные параметры предваряются словом VAR, входные параметры – без него. Порядок следования параметров может быть произвольным, например:

PROCEDURE PRIMER (VAR a, b, c: INTEGER;

m: CHAR; VAR i, j: REAL);

Здесь a, b, c, i, j – параметры-результаты (переменные); m – параметр-аргумент (значение).

В качестве иллюстрации комбинированных процедур рассмотрим последний вариант вычисления площади четырехугольника:

program PLOCHAD\_4;

var AB, BC, CD, AD, AC, S1, S2: real;

procedure GERON\_4 (a, b, c : eal; var S : real);

var P : real;

begin

P := (a + b + c)/2;

S := sqrt (P\*(P - a)\*(P - b)\*(P - c));

end;

begin {\*ОСНОВНАЯ ПРОГРАММА\*}

read (AB, BC, CD, AD, AC);

GERON\_4 (AB, BC, AC, S1);

GERON\_4 (AD, AC, CD, S2);

write ('S = ', S1 + S2)

end.

*Примечание*. Для более полного усвоения введенных ранее терминов перечислим на базе последнего примера все виды параметров и переменных:

1. глобальные переменные AB, BC, CD, AD, AC, S1, S2;
2. локальные переменные a, b, c, S, P;
3. формальные параметры a, b, c, S:
   * параметры-значения (аргументы) a, b, c;
   * параметр-переменная (результат) S;
4. фактические параметры AB, BC, CD, AD, AC, S1, S2:
   * параметры-значения (аргументы) AB, BC, CD, AD, AC;
   * параметры-переменные (результаты) S1, S2.

Заметим также, что термины «параметр-значение» и «аргумент», как и «параметр-переменная» и «результат», не всегда идентичны. Дело в том, что характеристика «значение (переменная)» отражает синтаксическую сущность параметра, а «аргумент (результат)» – его семантику (функциональную роль в процедуре). Иногда один и тот же параметр может быть аргументом и результатом одновременно, но описан в процедуре в виде параметра-переменной.

Попытка же описать выходной параметр в виде параметра-значения (без слова VAR в заголовке процедуры) приведет к тому, что результат работы процедуры не будет возвращен в основную программу. Это происходит потому, что характер «поведения» параметров-значений и параметров-переменных в процессе работы процедуры различен. Разница эта состоит в том, что преобразования, которые претерпевают формальные параметры-значения в процедуре, не вызывают изменения соответствующих им фактических параметров, в то время как изменения параметров-переменных могут изменять значения соответствующих фактических параметров.

Причиной этого феномена является неодинаковое распределение памяти под хранение параметров процедуры. Формальному параметру-значению отводится некоторая область (ячейка) памяти, куда заносится значение соответствующего фактического параметра, вычисленного на момент обращения к процедуре. На этом связь между ними обрывается. Действительно, если фактическим параметром является константа или выражение, то как изменения в формальном параметре-значении (а это есть всегда переменная) могут повлиять, например, на выражение?

Фактическим же параметром, соответствующим формальному параметру-переменной, является всегда переменная. На время выполнения процедуры эти параметры отождествляются, им соответствует одна и та же область памяти. Вполне понятно, что в этой ситуации изменения формального параметра влекут адекватные изменения фактического параметра, и после завершения процедуры его значение может отличаться от его первоначального значения.

Именно поэтому, объявив в процедуре параметр-результат как параметр-значение, этот результат так и останется в формальном параметре-переменной без его передачи в соответствующий фактический параметр (т.е нет возврата полученного процедурой значения) в основную программу.

## 4.2. Функции пользователя. Рекурсивные функции

Функция как объект языка Паскаль является другой версией реализации технологии построения программ с использованием структуры группирования. Можно также сказать, что функция есть частный вид определенного типа процедур, а именно процедур с одним параметром-переменной.

### 4.2.1. Определение функции

Функция отличается от процедуры только тем, что всегда возвращает в точку вызова одно скалярное значение. При этом функция, как и процедура, может содержать параметры-значения или быть без них.

**Общая форма записи заголовка функции**

**FUNCTION имя (список параметров: тип): тип;**

**или**

**FUNCTION имя: тип;**

Тип результата есть тип значения функции. Список параметров такой же, что и для процедуры, только здесь все параметры являются аргументами. Имя переменной, которая хранит значение функции, совпадает с именем функции.

Итак, заголовок функции отличается от заголовка процедуры не только сменой слова PROCEDURE на FUNCTION, но и удалением из списка параметров параметра-результата с присвоением его типа имени функции:

PROCEDURE <имя процедуры> (аргументы;   
VAR параметр-результат: тип);

FUNCTION <имя функции> (аргументы): тип;

Другой особенностью описания функции является наличие в нем хотя бы одного оператора присваивания, в левой части которого стоит имя определяемой функции, а в правой – выражение для вычисления результата функции. Очевидно, что тип этого выражения должен совпадать с указанным в заголовке типом функции.

Вызов функции также отличается от вызова процедуры. Если вызов процедуры осуществляется с помощью специального оператора вызова (оператора процедуры), то функция вызывается только внутри некоторого выражения. Для того, чтобы осуществить обращение к функции, необходимо использовать ее имя со списком фактических параметров в каком-либо выражении, тип которого совпадает с типом значения функции. Само же выражение, внутри которого вызывается функция, может быть правой частью оператора присваивания, частью логического выражения и пр.

### 4.2.2. Функции пользователя

Известно, что Паскаль имеет набор стандартных функций. Однако этот набор ограничен. Пользователь может по желанию расширить список функций, создав свои функции – функции пользователя. Так, например, в Паскале есть SQR (X) = X2, а вот функции F (X) = Xn, где n принадлежит множеству целых чисел Z, нет. Используя определенное ранее понятие функции, можно создать для этого универсальную функцию, которая давала бы степени произвольного вещественного числа с любым целым показателем.

Определим вещественную функцию POWER, которая должна иметь два параметра-аргумента – для показателя и для основания степени:

function POWER (FACTOR: real; EXP: integer): real;

var COUNT: integer; TFACTOR: real;

begin

if EXP = 0 then POWER := 1

else begin

TFACTOR := FACTOR;

for COUNT := 2 to ABS (EXP) do

TFACTOR := TFACTOR\*FACTOR;

if EXP < 0 then POWER := 1/TFACTOR

else POWER := TFACTOR

end

end;

Теперь можно эту функцию вызывать следующим образом:

1. F := POWER(5.25, 3);
2. WRITELN ("D = ", POWER (5.25, -2):5:2);
3. IF X > 2\*POWER (6.2, 3) THEN WRITE ('ДА');
4. A := POWER (X, 2) + POWER (X, 3) + POWER (X, 4).

### 4.2.3. Рекурсивные функции

К функциям можно обращаться тремя способами: из тела основной программы, из тела другой функции, из тела самой функции, т.е. функция может вызывать саму себя. Функции называются *рекурсивными*, если в описании функции происходит вызов самой себя, а процесс обращения – *рекурсией*. Продемонстрируем использование рекурсии на примере вычисления значения факториала произвольного натурального числа N.

В математике известно рекурсивное определение факториала:

n! = 1 при n = 0;

n! = (n - 1)! ⋅ n при n > 0.

Это рекурсивное определение можно реализовать с помощью соответствующей рекурсивной функции:

function FACTORIAL (VALUE: integer): integer;

begin

iF VALUE = 0 then FACTORIAL := 1

else FACTORIAL := VALUE\*FACTORIAL (VALUE - 1)

end;

Теперь можно обращаться к этой функции в теле основной программы, как показано в следующем примере:

program FINDFACTORIAL;

var N: integer;

begin

writeln ('Введите число');

readln (N);

if N < 0 then writeln ('Нет факториала')

else writeln ('Факториал ', N, ' равен ', FACTORIAL (N))

end.

Мы видим, что характерной особенностью построенной функции является наличие в ее теле оператора присваивания:

FACTORIAL := VALUE\*FACTORIAL (VALUE - 1),

где происходит вызов определяемой функции. Здесь идентификатор FACTORIAL в левой части оператора обозначает имя переменной для хранения значения функции, а в правой – имя вызываемой функции.

Важным моментом при составлении любой рекурсивной функции является организация выхода из рекурсии. В некоторых простых случаях должно существовать не рекурсивное решение. Рекурсивный процесс должен шаг за шагом так упрощать задачу, чтобы, в конце концов, для нее появилось не рекурсивное решение. В этих функциях должны проверяться значения аргумента для принятия решения о завершении. В нашем случае условием завершения рекурсии является VALUE = 0.

При описании рекурсивных функций необходимо хорошо представлять процесс вычислений. Всякая рекурсия состоит из двух этапов: углубления (погружения) внутрь рекурсии и выхода из нее. На первом этапе никаких вычислений не производится, а идет только настройка рабочей формулы на конкретные операнды. На втором этапе происходит процесс вычислений по настроенным формулам.

В заключение покажем, что часто рекурсивные функции строятся гораздо проще, чем «обычные», хотя вполне понятно, что не всякая функция может быть переделана на рекурсивную. Сделаем это на примере уже построенной ранее функции POWER.

Данная функция явно носит рекурсивный характер, исходя из ее определения:

Xn = 1, если n = 0;

Xn = Xn-1 \* X, если n > 1.

Ниже следует рекурсивная функция вычисления значения степени:

function POWER (FACTOR: real; EXPONENT: integer): REAL;

begin

if EXPONENT < 0

then POWER := 1/POWER (FACTOR, abs (EXPONENT))

else

if EXPONENT > 0

then POWER := FACTOR\*POWER (FACTOR, EXPONENT - 1)

else POWER := 1

end;

*Замечание*. Помимо рекурсивных функций, в языке Паскаль по тому же принципу можно определять и рекурсивные процедуры. Подробно о них будет сказано в следующих разделах, а пока покажем, как рекурсивная функция может быть переделана в рекурсивную процедуру на примере вычисления факториала:

procedure FACTORIAL (VALUE: integer; var F: integer);

begin

iF VALUE = 0 then F := 1

else begin FACTORIAL (VALUE - 1, F);

F := F\*VALUE

end;

end;

Здесь уже, в отличие от функции FACTORIAL, для вычисления N! необходимо вызвать эту процедуру с помощью оператора процедуры FACTORIAL (N, FN), где FN – переменная для возвращения из процедуры значения N!.

## Лабораторная работа №4

**Цель работы:** научиться создавать и вызывать в Паскаль-программах процедуры и функции, уметь использовать рекурсивные функции.

**Общие сведения**

Процедуры и функции являются «строительными блоками» для программ и обеспечивают их модульность, а потому представляют собой важнейшее средство увеличения эффективности программирования.

Перед выполнением работы необходимо ознакомиться с правилами оформления и вызова процедур и функций в языке программирования Паскаль.

**Пример 1.** Вычислить значения функции *f(x)=2 cos x+3*, при *x*={1; 4; 7,5; 20}. Вывести результаты в два столбца: в первом - значения *x*, во втором - значения *f(x)*. Вычисления провести двумя способами: с помощью функции и процедуры.

**Решение.** Аргумент и результат функции – действительные числа, поэтому используем тип Real. В теле функции будет только оператор присваивания – для вычисления значения выражения. Процедура отличается строкой заголовка, - для передачи в основную программу результатов вычислений добавим параметр-переменную *fx*. Чтобы вывести результаты в виде таблицы, используем форматный вывод.

program proc\_1;

function f (x: Real):Real;

begin

f:=2\*cos(x)+3

end;

procedure proc\_f (x: Real; var fx: real);

begin

fx:=2\*cos(x)+3

end;

var x, fx: real;

begin

Writeln('с использованием процедуры:');

Writeln(' x f(x)');

x:=1; proc\_f (x,fx); Writeln (x:6:2, fx:6:2);

x:=4; proc\_f (x,fx); Writeln (x:6:2, fx:6:2);

x:=7.5; proc\_f (x,fx); Writeln (x:6:2, fx:6:2);

x:=20; proc\_f (x,fx); Writeln (x:6:2, fx:6:2);

Readln;

Writeln('с использованием функции:');

Writeln(' x f(x)');

Writeln(1:6, f (1):6:2);

Writeln(4:6, f (4):6:2);

Writeln(7.5:6:2, f (7.5):6:2);

Writeln(20:6, f (20):6:2);

Readln;

end.

**Пример 2.** Создать рекурсивную функцию поиска i-го члена последовательности, заданной рекуррентной формулой A1=const1, A2=const2, Ai=3Ai-2-Ai-1. Вывести через пробел значения рекурсивной функции при значениях аргумента от 1 до 10 включительно.



Решение. По условию задачи аргумент может принимать только целые значения, поэтому функция имеет параметр-значение типа Integer. Выход из рекурсии в данном случае осуществляется при двух значениях аргумента (при *i*=1, *i*=2), поэтому в рекурсивной функции необходимы два вложенных условных оператора или же оператор выбора *case*. В приведенном примере использованы операторы *if*, но попробуйте самостоятельно записать решение с помощью оператора выбора. В основной программе значения аргумента - целые последовательные числа, поэтому следует воспользоваться оператором цикла с параметром *for*.

program proc\_2;

function A (i: Integer): Integer;

begin

if i=1 then A:=1 else

if i=2 then A:=3 else А:=3\*A(i-2)-A(i-1)

end;

var i: Integer;

begin

for i:=1 to 10 do Write (A(i),' ');

Readln

end.

**Варианты заданий**

**Задание 1.** Составить программу для решения задачи с применением функции и процедуры пользователя.

1. В правильном треугольнике проведена средняя линия. Найти площадь образовавшейся трапеции, дважды используя функцию вычисления площади правильного треугольника по формуле:



1. Для правильного треугольника со стороной *а* построены вписанная и описанная окружности. Найти площадь образовавшегося кольца, используя функцию вычисления площади круга S=πR2. Для нахождения радиусов окружностей воспользуйтесь формулами:

,



1. Тариф предусматривает оплату телефонных разговоров следующим образом: при продолжительности разговора меньше *P* минут стоимость одной минуты составляет *S1* копеек, в противном случае – *S2* коп/мин (*S1*, *S2*, *P* - константы). Используя функцию вычисления стоимости одного разговора, найти суммарную стоимость трех звонков известной продолжительности.
2. На товар дважды была сделана скидка – на *p1*, а затем на *p2* процентов. Первоначальная стоимость товара составляла *S* рублей. Используя функцию вычисления стоимости товара с учетом скидки на *P* процентов, найти стоимость товара после двойной скидки.



**Задание 2.** Вывести значения рекурсивной функции при значениях аргумента от 1 до 10 включительно.

1. Найти член последовательности, заданной формулой: *Di=7+Di-1* при i>1, где D1 определяется пользователем.
2. Найти член последовательности, заданной формулой: *Ai=Ai-1-Ai-2* при i>2. Значения первого и второго членов последовательности вводятся пользователем.
3. Найти член последовательности, заданной следующим образом:  
    y1=0; y2=10; yn=2⋅yn-1-yn-2, где n>2.
4. Найти член последовательности, заданной формулой *Bi=4·Bi-1*, при i>1. Значения первого члена последовательности вводится пользователем.

**Дополнительные задания**

1. Для чисел *a, b, c* найти значение выражения *min(a,ab)+min(a,ac)+1* с использованием функции вычисления минимального из двух чисел.
2. Дан квадрат со стороной *а*, диагональ этого квадрата является стороной второго квадрата, диагональ второго квадрата – стороной третьего. Найти длину стороны третьего квадрата, используя функцию вычисления длины диагонали квадрата по его стороне:



**Контрольные вопросы**

1. В чем состоит принципиальное отличие процедур от функций?
2. Чем отличается вызов функции от вызова процедуры?
3. Какой переменной присваивается значение в процедуре и в функции?
4. Какие переменные в языке Паскаль называются локальными, а какие глобальными?
5. Какие процедуры и функции называют рекурсивными?
6. Как отличить «обычную» функцию от рекурсивной?
7. Если значения процедуры и функции являются операндами некоторого выражения, то которую из них можно вставить «непосредственно» в это выражение?

# 5. ОБРАБОТКА МАССИВОВ

*Скалярный тип* – простой тип данных. Скалярное данное неделимо. *Массив* – это структурированный тип данных. Массив состоит из нескольких элементов. Ко всему массиву можно обращаться по его имени. Можно обращаться к его элементу, но для этого надо задать индекс (индексы). Массивы бывают одномерные и многомерные. Для объявления массива необходимо задать типы его индексов и компонент:

**ARRAY [Тип индексов] OF <Тип копонент>;**

*Тип компонент массива* – это просто тип данных, ассоциированный с каждой компонентой массива. Тип компонент может быть любым REAL, INTEGER, CHAR, BOOLEAN, перечислимым, интервальным. В качестве компоненты массива может быть взят и тип массив.

Тип индекса должен быть одним из упорядоченных типов, т.е. любым скалярным типом, кроме REAL: INTEGER, CHAR, интервальный, перечислимый. Тип индекса определяет границы изменения индекса. Если сделана попытка использовать несуществующую компоненту, то возникает ошибка (ошибка неверного индекса).

## 5.1. Одномерные массивы

Одномерный массив можно задать (объявить) двумя способами:

1. C помощью служебного слова TYPE описывается тип массива, а затем с помощью VAR вводится переменная этого типа.

**Общая форма записи**

TYPE <тип массива> = ARRAY [тип индекса] OF <тип компонент>;

VAR <переменная>: <тип массива>;

1. С помощью слова VAR сразу описывается переменная типа массив.

**Общая форма записи**

VAR <переменная>: ARRAY [тип индекса] OF <тип компонент>;

Например, объявление массива из 100 элементов типа REAL можно осуществить двумя способами:

1. type R100 = array [1..100] of real;

var A: R100;

1. var A: array [1..100] of real.

Здесь задан массив с именем А, и его элементы имеют имена: А[1],…,A[100]. Чаще всего для типа индекса используют интервальный тип на основе типов INTEGER и CHAR. Однако можно в качестве индексов брать перечислимый тип.

П р и м е р 1. Подсчет числа вхождений букв в текст определенной длины.

program COUNTER;

var COUNT: array ['a'..'z'] of integer;

CH: char; N: integer;

begin

for CH := 'a' to 'z' do

COUNT [CH] := 0; N := 0;

repeat

read (CH); N := N + 1;

if (CH >= 'a') and (CH <= 'z') then

COUNT [CH] := COUNT [CH] + 1;

until CH = '.';

for CH := 'a' to 'z' do

writeln (CH, COUNT [CH]:5);

end.

*Пояснение*. В этом примере тип индекса есть интервальный тип на базе типа CHAR, а тип компонент есть целое число. Таким образом, элементы массива – числа, а их индексы – буквы, т.е. число элементов массива равно 26 (по числу букв латинского алфавита).

Рассмотрим теперь случай, когда тип индекса задан перечислимым типом, а компоненты массива представлены компонентами интервального типа на базе типа INTEGER.

П р и м е р 2. Присваивание переменной с именем месяца числа дней этого месяца.

program NUMBRDAY;

type MONAT = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG,

SEP, OKT, NOV, DEC);

var DAY : array [MONAT] of 28..31; T : MONAT;

begin

for T := JAN to DEC do

case T of

JAN, MAR, MAY, JUL, AUG, OKT, DEC: DAY [T] := 31;

APR, JUN, SEP, NOV: DAY [T] := 30;

FEB : DAY [T] := 28;

end;

end.

## 5.2. Многомерные массивы

Для определения позиции элемента в двумерном массиве необходимы два индекса. Любой двумерный массив есть матрица, а матрица есть таблица. Поэтому удобно описывать двумерные массивы путем указания границ изменения индексов (номеров) строк и столбцов.

Например, таблица символов M × N, где M – число строк и N – число столбцов, может быть описана:

var TAB : array [1..M, 1..N] of char;

|  |
| --- |
| **Общая форма записи**  VAR <имя> : ARRAY [тип индекса строки, тип индекса столбца]  OF <тип компонент>; |

Однако двумерный массив можно интерпретировать как вектор-столбец, каждый элемент которого, в свою очередь, является одномерным массивом (вектор-строка). Этот подход к определению двумерного массива влечет его описание с помощью двух строк, где первая содержит описание строки, а вторая – описание столбца:

type LINE = array [1..N] of char;

STOLB = array [1..M] of LINE;

var TAB : STOLB.

Здесь TAB [I] – переменная типа LINE, а TAB [I][J] – переменная типа CHAR.

|  |
| --- |
| **Общая форма записи**  TYPE <тип строки> = ARRAY [тип индекса] OF <тип компонент>;  <тип столбца> = ARRAY [тип индекса] OF <тип строки>;  VAR <переменная массива> : <тип столбца>; |

Эти два вида определения массивов задают и два способа обращения к элементам массива: TAB [I, J] – в первом случае и TAB [I][J] – во втором. Вполне очевидно, что сказанное выше для двумерного массива распространяется и на массивы большей размерности. Например, описание VAR CUBE : ARRAY [1..M, 1..N, 1..K] OF INTEGER определяет задание трехмерного массива целых чисел.

## 

## 5.3. Примеры работы с массивами

Обработка массивов включает в себя, как правило, следующие компоненты: ввод массива (с клавиатуры или с помощью датчика случайных чисел), вывод полученного массива на экран и собственно его обработка. Все эти компоненты рекомендуется оформлять в виде отдельных процедур. При этом надо учитывать следующий фактор: если процедуре (или функции) будет передаваться массив, то надо объявить в ней этот массив как параметр с атрибутом VAR даже в том случае, если значение массива внутри процедуры не изменяется. Это нужно для того, чтобы не тратить времени и памяти на размещение внутри процедуры копии массива. Заметим, что параметр обязательно должен относиться к типу, имеющему имя.

П р и м е р. Сумма элементов таблицы над верхней диагональю.

program SUMMA;

const M = 10; {число строк таблицы}

N = 10; {число столбцов таблицы}

type LINE = array [1..n] of integer;

TAB = array [1..m] of LINE;

var s, i, j:integer; MAS: TAB;

procedure VVODMASSIV (var MAS: TAB);

begin

¦ for i := 1 to M do

¦ for j := 1 to N do

¦ readln (MAS [i][j]);

end;

procedure VIVODMASSIV (var MAS:TAB);

begin

¦ for i := 1 to M do

¦ begin

¦ ¦ for j := 1 to N do

¦ ¦ write (MAS [i][j]:4,' '); writeln;

¦ end;

end;

procedure OBRABOTKA (MAS: TAB; var SUM: integer);

begin

¦ SUM := 0;

¦ for i := 1 to M do

¦ for j := 1 to N do

¦ if j > i then SUM := SUM + MAS [i][j];

end;

begin

| VVODMASSIV (MAS);

| writeln ('исходный массив'); VIVODMASSIV (MAS);

| OBRABOTKA (MAS, s); writeln;

| writeln ('сумма элементов = ',s);

end.

## Лабораторная работа №5

**Цель работы**: овладеть основными приемами работы с одномерными и двумерными массивами, уметь различать в двумерном массиве обработку строк и столбцов, а также отличать нахождение первых и последних элементов последовательности, обладающих некоторым свойством.

**Общие сведения**

Табличное представление информации одно из самых распространенных, поэтому массивы широко применяются в прикладных программах.

Перед выполнением работы необходимо ознакомиться с теоретическим материалом по теме «Обработка массивов».

**Пример 1.** Составить программу, позволяющую в одномерном массиве, состоящем из *N* вещественных элементов, вычислить сумму положительных элементов.

Решение. При написании процедур ввода и вывода следует обратить внимание, что элементы – вещественные числа, поэтому необходимо позаботиться о верной обработке дробной части. Вычисление суммы оформим в виде функции с одним аргументом - массивом. Локальными переменными функции будут индексная переменная *i* и дополнительная переменная *s* для хранения текущей суммы элементов. В начале тела функции обязательно обнуление *s*. Каждый элемент массива сравним с нулем, и, если значение положительно, добавим его к искомой сумме *s*. В конце функции запишем значение переменной *s* в результирующую переменную*.*

program massiv\_1;

const N=10;

type mas=array [1..N] of Real;

procedure Vvodmas(var A:mas);

var i:Integer;

begin

for i:=1 to N do A[i]:=-50+Random(101)+random;

end;

procedure Vivodmas(A:mas);

var i:Integer;

begin

for i:=1 to N do Write(A[i]:8:2);

Writeln

end;

function Summa(A:mas):real;

var i: Integer; s:real;

begin

s:=0;

for i:=1 to N do if A[i]>0 then s:=s+A[i];

Summa:=s;

end;

var A: mas;

begin

Randomize; Vvodmas(A);

Writeln('Исходный массив:'); Vivodmas(A);

Writeln('Ответ: ', Summa(A):0:2);

Readln

end.

**Пример 2.** В двумерном массиве, состоящем из целочисленных элементов, в каждом столбце поменять местами наибольший по модулю и последний не принадлежащий интервалу *(a, b)* элементы массива.

Решение. Преобразования необходимо провести в каждом столбце массива, поэтому параметр внешнего цикла в процедуре обработки - номер столбца *j*, а вложенного – номер строки *i*.

Для перестановки двух элементов в столбце массива необходимо найти номера их строк *n1* и *n2*, а затем поменять местами значения элементов с использованием промежуточной переменной *p*.

Чтобы найти наибольший по модулю элемент столбца, введем дополнительную переменную *max*, которая будет хранить максимальное по модулю значение в текущем столбце массива на данный момент. (Можно решить задачу без использования переменной *max.* Подумайте, как это сделать.)

Программа должна корректно работать с любыми входными данными, а значит и в тех случаях, когда некоторые или даже все столбцы массива содержат только элементы из интервала *(a, b)*, и обмен значений в некоторых столбцах или во всем массиве не нужен.

const n=10; m=7;

type mas=array [1..n,1..m] of Integer;

procedure Vvodmas(var D:mas);

var i,j:Integer;

begin

for i:=1 to n do

for j:=1 to m do

D[i,j]:=-50+Random(101);

end;

procedure Vivodmas(D:mas);

var i,j:Integer;

begin

for i:=1 to n do

begin

for j:=1 to m do Write(D[i,j]:4);

Writeln;

end;

end;

procedure Obmen(a,b: real; var D:mas);

var i,j,p,n1,n2,max: Integer;

begin

for j:=1 to m do

begin

n1:=1; max:=abs(D[1,j]);{считаем первый элемент столбца наибольшим по модулю}

for i:=2 to n do

if abs(D[i,j])>max then {обнаружен больший элемент}

begin n1:=i; max:=abs(D[i,j]) end;

i:=n; {перебираем элементы столбца, начиная с последнего}

while (i>=1)and (D[i,j]>a)and(D[i,j]<b) do i:=i-1;

n2:=i;

if n2<>0 then {если элемент, не принадлежащий интервалу (a,b), был найден}

begin

p:=D[n1,j]; D[n1,j]:=D[n2,j]; D[n2,j]:=p; {обмен значений}

end;

end;

end;

var D: mas; a,b:Real;

begin

Randomize; Vvodmas(D);

Writeln('Исходный массив:'); Vivodmas(D);

Write('Введите через пробел концы интервала (a,b): '); Readln(a,b);

Obmen(a,b,D);

Writeln('Ответ:'); Vivodmas(D);

Readln

end.

**Варианты заданий**

**Задание 1.** Составить программу, позволяющую в одномерном массиве, состоящем из *N* вещественных элементов, вычислить:

1. сумму модулей отрицательных элементов массива;
2. количество элементов массива, не принадлежащих интервалу *(a, b)*;
3. наименьший из элементов массива, принадлежащих отрезку [*a*, *b*];
4. количество элементов массива, равных первому элементу;

**Задание 2.** В двумерном массиве, состоящем из целочисленных элементов, поменять местами:

1. в каждом столбце наибольший по модулю и последний положительный элементы;
2. в каждом столбце первый и последний отрицательные элементы;
3. в каждой строке наибольший и наименьший элементы;
4. в каждом столбце первый принадлежащий отрезку [*a*, *b*] и первый отрицательный элементы;

**Дополнительные задания**

1. Определить в одномерном массиве число соседств из двух чисел разного знака.
2. Дан двумерный массив целых чисел. Поменять местами строку, содержащую максимум массива, со строкой, содержащей его минимум.

**Контрольные вопросы**

1. Как описываются в языке Паскаль одномерный и двумерные массивы?
2. Может ли массив содержать разнотипные данные?
3. В каком порядке указываются индексы при обращении к элементам двумерного массива?
4. Привести пример массива, описание которого выглядит следующим образом: *var A: array [1..3, 20..24] of real.*
5. Можно ли при обработке двумерных массивов использовать однократные циклы? Если да, то приведите примеры.
6. Если в одномерном массиве проверяется «похожесть» его первой и второй части, то в каких границах надо писать оператор *for* для «прохождения» по этому массиву?
7. Каким образом надо находить первый и последний элементы одномерного массива, обладающие некоторым свойством (отрицательный, наибольший, входящий в интервал и *пр.)?*

# 6. ОБРАБОТКА СТРОКОВЫХ ВЕЛИЧИН

В Паскале, как и в других языках программирования, предусмотрена обработка текстов или строк. Для этой цели в языке существует два типа данных: SHAR и STRING.

## 

## 6.1. Тип данных CHAR

Типу данных CHAR соответствуют символьные константы и переменные. Символьная константа есть какой-то символ алфавита, взятый в апострофы. Символьные переменные получают значения символьных констант с помощью оператора присваивания:

ALPFA := 'p'; A := 't'; B := '3'; C := ' '; D := ''.

Все символы алфавита образуют множество литер. Каждый символ имеет свой код в ASCII. Это позволяет использовать булевские сравнения: =, <>, <, <=, >, >= .

Данные этого типа описываются с помощью служебного слова CHAR. Например, переменную ALPFA можно описать как VAR ALPFA : CHAR.

|  |
| --- |
| **Общая форма записи**  VAR <переменная>: CHAR; |

При работе с данными типа CHAR, если у нас есть последовательность символов, существует два способа ввода этих символов с клавиатуры.

При первом способе организуется цикл, внутри которого помещается оператор READLN. При этом способе элементы последовательности вводятся поочередно, и после набора на клавиатуре символа необходимо нажать клавишу ввода ENTER. Таким образом, здесь число нажатий клавиши ENTER совпадает с числом вводимых элементов последовательности.

Второй способ характеризуется применением для ввода символов оператора READ. С его помощью можно сразу же ввести всю последовательность символов, которая записывается в буфер клавиатуры. Последующий цикл с оператором READ осуществляет уже выборку элементов из этого буфера в соответствующие переменные, указанные в операторе READ.

П р и м е р 1. С клавиатуры последовательно вводятся символы. Признаком конца ввода является точка. Составить программу выбрасывания групп символов, расположенных между скобками ( ). Сами скобки тоже выбрасываются.

program SKOBKI;

var c: char; i: integer;

begin

¦ i := 0; read (c);

¦ while c <> '.' do

¦ begin

¦ ¦ if c = '(' then i : = 1

¦ ¦ else if c = ')' then i := 0

¦ ¦ else if i = 0 then write (c);

¦ ¦ read (c);

¦ ¦ end;

end.

*Пояснение*. I = 1 означает, что ранее была прочитана левая скобка, которой пока еще не нашлось парной правой. В этой ситуации прочитанные символы не выводятся на экран. В результате работы этой программы на экране будет представлена строка символов. Здесь вся последовательность символов вводится сразу по первому оператору READ, а затем в цикле из буфера клавиатуры выбираются, анализируются и выводятся на экран символы вне круглых скобок. Например, если вводится последовательность «asg(zx)ytr.», то экран будет выглядеть так:

asg(zx)ytr. – результат работы оператора READ;

asgytr – результат работы оператора WRITE.

В этой программе можно было бы использовать оператор READLN, но тогда после набора каждого символа необходимо нажимать клавишу ввода. Кроме того, на экран будет выводиться не строка символов, а столбец, состоящий из вводимых и отпечатанных элементов.

Упорядоченность символов языка используется при написании циклов с параметром, где параметр цикла может пробегать буквенные значения.

П р и м е р 2. Программа вывода последовательности букв a, ab, abc, ..., abc ... xyz.

program SUITE;

var c, d: char;

begin

for c := 'a' to 'z' do

begin

for d := 'a' to c do write (d);

writeln (' ');

end;

end.

## 6.2. Массивы литер

В рассмотренных программах все символы вводились последовательно в процессе работы цикла или хранились временно в буфере клавиатуры. Это не всегда удобно, поэтому в языках делают строки как последовательность литер. Строку можно задать как массив литер, при этом в качестве длины строки может выступать верхняя граница массива, например:

VAR HAMLET : ARRAY [1..17] OF CHAR.

Здесь HAMLET – массив литер, компоненты которого имеют тип CHAR; индекс имеет нижнюю границу, равную 1, верхнюю – 17. Для ввода строки в массив HAMLET необходимо организовать цикл из 17 повторений. При каждом повторе этого цикла с клавиатуры вводится очередной символ строки и нажимается клавиша ввода:

for n := 1 to 17 do readln (HAMLET [n]).

Этот массив можно образовать и с помощью однократного нажатия клавиши ввода после набора всех элементов строки:

for n := 1 to 17 do read (HAMLET [n]).

Поскольку массивы литер являются обычными массивами, но их компоненты имеют тип CHAR, то они обладают всеми свойствами регулярных массивов. Для извлечения из массива-строки отдельного символа необходимо использовать индекс этого элемента. Например, можно вывести на экран строку HAMLET в обратном порядке с помощью следующего цикла:

for n := 17 downto 1 do write (HAMLET [n]).

## 6.3. Тип данных STRING

Наряду с тем положительным, что дают нам массивы литер, они обладают существенным недостатком: их длину нельзя менять во время выполнения программы. Так, описанная переменная HAMLET в п. 7.2 есть массив из 17 элементов, и, следовательно, туда можно поместить только текст, содержащий ровно 17 символов.

Это не всегда удобно. Хотелось бы иметь такую переменную, в которую можно было бы поместить текст произвольной (но ограниченной) длины. Такую возможность предоставляет тип STRING. Так, объявив переменную var HAMLET: string [17], можно путем оператора присваивания (а не через цикл) задать ей значение текста произвольной длины (от 0 до 17), например:

HAMLET := 'Быть или не быть';

HAMLET := 'Бедный Йорик';

HAMLET := ' '; HAMLET:= ''.

Отметим также, что при компиляции программы в случае объявления строки-массива в памяти ЭВМ резервируется место под массив, который должен быть полностью заполнен в процессе работы программы. Для типа STRING также резервируется место в памяти того же объема, но здесь не обязательно заполнять его целиком. Незаполненные места представлены пробелами. Данный тип представлен следующейобщей формой записи:

|  |
| --- |
| **Общая форма записи**  TYPE <имя типа> = STRING [N];  VAR <имя переменной>: <имя типа>;  или  VAR <имя переменной>: STRING [N]; |

Здесь N – целая константа, задающая максимальную длину текста. Доступ к элементам строки производится с помощью индексов, так как в этом типе также все элементы имеют свой (числовой) индекс от 1 до N. В результате получается величина типа CHAR, например:

HAMLET := 'ПРОГРАММА';

HAMLET [1] = 'П'; HAMLET [9] = 'А'.

Тип STRING и стандартный тип CHAR совместимы. Строки и символы могут употребляться в одних и тех же строковых выражениях.

Строковое выражение состоит из строковых (символьных) констант, переменных, указателей строковых функций и операции конкатенации (склеивания) строк, обозначаемой знаком «+». Строки можно сравнивать. В результате сравнения двух строк истина получается только в том случае, если сравниваемые строки совпадают посимвольно и имеют одинаковую длину (принадлежат одному и тому же типу).

## 6.4. Строковые функции и процедуры

Строковые функции и процедуры введены в систему программирования Turbo Pascal для облегчения манипуляции со строками. Имеется восемь строковых функций и процедур.

1. *Функция CONCAT (склеивание)*

Синтаксис: concat (S1, S2, ..., Sn: string): string. Возвращает строку, полученную конкатенацией строк S1,...,Sn. Если длина результата больше 256, то излишние символы отбрасываются. Эта функция фигурирует в правой части «:=» и в строковых выражениях.

П р и м е р:

NUMBER := concat ('12', '34', '50'); NUMBER = '123450'.

1. *Функция LENGTH (длина)*

Синтаксис: length (S: string): integer. Возвращает длину строки S.

П р и м е р :

N := length ('345'); N = 3.

1. *Функция POS (позиция)*

Синтаксис: pos (S,T: string): integer. Функция POS в качестве аргументов использует две строки и определяет, содержится ли первая строка во второй. Возвращает номер символа, начиная с которого S входит в T. Если вхождения нет, то возвращает 0.

П р и м е р :

N := pos ('E', 'HELLO');

N = 2.

N := pos ('A', 'HELLO');

N = 0 .

1. *Функция COPY (вырезка фрагмента)*

Синтаксис: copy (S: string; N1, N: integer): string. Возвращает подстроку, полученную из N символов строки S, начиная с позиции N1. Значение переменной S при этом не меняется.

П р и м е р :

FRAGMENT := copy ('PROGRAMM', 2, 3);

FRAGMENT = 'ROG'.

1. *Процедура DELETE (стирание фрагмента)*

Синтаксис: delete (var S: string; POS, LEN: integer). Убирает из строки S LEN символов, начиная с POS, при этом длина строки уменьшается на LEN позиций.

П р и м е р :

FRAGMENT := 'PROGRAMM';

delete (FRAGMENT, 2, 3);

FRAGMENT = 'PRAMM'.

1. *Процедура INSERT (вставка)*

Синтаксис: insert (S: string; var D: string; POS: integer). Вставляет строку S в строку D перед символом с номером POS, при этом длина строки D увеличивается на LENGTH (S) позиций.

П р и м е р :

FRAGMENT := 'PRAMM';

insert ('ROG', FRAGMENT, 2);

FRAGMENT = 'PROGRAMM'.

7. *Процедура STR (преобразование в строку)*

Синтаксис: str (I: integer; var S: string); str (R: real; var S: string).

Преобразует I или R из числа в строку и записывает эту строку в S, причем R и I могут записываться форматно, как в процедуре WRITE.

П р и м е р :

a) R := 123.654; str (R:5:2, S); S = '123.65';

б) I := 5683; str (I, S); s = '5683'.

8. *Процедура VAL (преобразование в число)*

Синтаксис: val (S: string; var I, J: integer); val (S: string; var I: real; var J: integer).

Преобразует строковую переменную S в число типа I. Переменная J получает значение 0, если перевод прошел без ошибок. Если же сделана попытка конвертировать в число строку, где есть нецифровые символы, то переменная J принимает значение позиции первого нецифрового символа. При этом работа процедуры прерывается.

П р и м е р :

1. S := '4326'; б) S := '43p8';

val (S, I, J); val (S, I, J);

I = 4326, J = 0; I – не определено, J = 3.

Рассмотрим теперь пример на применение указанных функций и процедур обработки строк.

П р и м е р . Изменение порядка слов в строке.

program REVERSE;

var OLD\_LINE, NEW\_LINE: string [50];

PROBEL: integer; WORD: string [50];

begin

NEW\_LINE := ''; readln (OLD\_LINE);

OLD\_LINE := concat (OLD\_LINE,' ');

while OLD\_LINE <> '' do

begin

PROBEL := pos (' ', OLD\_LINE);

word := copy (OLD\_LINE, 1, PROBEL);

NEW\_LINE := concat (WORD, NEW\_LINE);

Delete (OLD\_LINE, 1, PROBEL);

end;

writeln (NEW\_LINE)

end.

*Пояснение.* С клавиатуры вводится строка OLD\_LINE, и к ней справа подклеивается пробел. Это делается для того, чтобы строка имела одну и ту же структуру: слово плюс пробел. Затем в цикле, признаком конца которого является пустая константа, выделяется очередное по порядку слово и подклеивается слева в переменную NEW\_ LINE. После выборки очередного слова из OLD\_LINE оно оттуда выбрасывается, что приводит к постепенному уменьшению самой строки. Здесь переменная PROBEL служит для хранения позиции первого пробела в строке, а WORD – для выбранного из OLD\_LINE слова.

Например, строка *'Наша Таня громко плачет'* преобразуется в строку *'плачет громко Таня Наша'*.

## Лабораторная работа №6

**Цель работы**: освоить строковые операции, процедуры и функции; научиться создавать программы обработки текстовых данных, представляющих последовательность символов и строк, состоящих из слов, разделенных одним или несколькими пробелами.

**Общие сведения**

Для обработки текстовой информации можно использовать те же методы, что применяют для одномерных массивов, так как структура строкового типа схожа с массивом. Однако для упрощения написания программ по работе с текстами были разработаны стандартные строковые процедуры и функции. Поэтому основной задачей этой лабораторной работы является освоение строковых операций, функций *Length, Pos, Copy* и процедур *Delete, Insert*.

Перед выполнением работы необходимо ознакомиться с теоретическим материалом по теме «Обработка литерных величин. Данные типа *Char* и *String*».

**Пример 1.** Составить программу обработки данной строки, позволяющую выписать все знаки сравнения и все скобки, сохранив их последовательность.

Решение. Все знаки сравнения и скобки перечислим в строковой константе *srav\_sk*. В теле программы последовательно рассмотрим все символы введенной строки *s*, проверяя каждый на вхождение в строку *srav\_sk*, выводя на экран содержащиеся в строковой константе символы строки *s*.

program string\_1;

const srav\_sk='<>=(){}[]';

var s:String; i:Integer;

begin

Writeln('Введите строку:'); Readln(s);

for i:=1 to Length(s) do

if Pos(s[i],srav\_sk)<>0 then write(s[i]);

Readln

end.

**Пример 2.** Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Составить программу обработки данной строки, выводящую через запятую слова четной длины, которые при обмене местами левой и правой половины дают то же слово (например, папа, мама, канкан).

Решение. В строке, последовательно перебирая элементы, будем выделять начало *ns* и конец *ks* каждого слова, копировать слово в другую строковую переменную, а затем проверять четность длины и равенство частей строки *slovo*. Слова, для которых выполнено условие задания будем склеивать в переменной *otv*. После добавления к ответу последнего слова в конце *otv* будет лишняя запятая, которую перед выводом на экран необходимо удалить.

program string\_2;

var s, slovo, otv:String; ns, ks, i, m: Integer;

begin

Writeln ('Введите строку:'); Readln(s);

i:=1; ns:=1; otv:=''; s:=s+' ';

while i<=Length(s) do

begin

while (i<=Length(s)) and (s[i]<>' ') do i:=i+1; {ищем очередной пробел}

ks:=i; slovo:=Copy(s, ns, ks-ns); {выделяем слово}

m:=Length(slovo) div 2; {половина длины слова}

if (Length(slovo) mod 2=0) and (Copy(slovo,1,m)=Copy(slovo,m+1,m))

{если слово имеет четную длину}

then otv:= otv+slovo+ ','; {добавляем слово к ответу}

while (i<=Length(s)) and (s[i]=' ') do i:=i+1; {пропускаем пробелы}

ns:=i; {начало следующего слова}

end;

Delete(otv, Length(otv), 1); {удаляем лишнюю запятую в конце}

Writeln(otv);

Readln

end.

**Варианты заданий**

**Задание 1.** Составить программу обработки данной строки, позволяющую:

1. выписать все прописные буквы из данной строки в порядке их следования;
2. определить, содержатся ли в тексте цифры;
3. найти общее количество круглых, квадратных и фигурных скобок в тексте;
4. выписать все знаки препинания из данной строки, сохранив их последовательность;

**Задание 2.** Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Составить программу обработки данной строки, выводящую через запятую:

1. слова, одинаково читающиеся справа налево и слева направо;
2. слова заданной длины, в которые входит данная буква;
3. слова, начинающиеся с прописной буквы, в которых все остальные буквы строчные;
4. слова, которые начинаются и заканчиваются одной и той же буквой и содержат хотя бы одну введенную с клавиатуры букву.

**Дополнительные задания**

Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Составить программу обработки данной строки, позволяющую:

* 1. определить количество слов в строке;
  2. удалить из строки избыточные пробелы, чтобы между словами осталось только по одному пробелу;
  3. найти самое короткое слово и его длину;
  4. найти в тексте слова-перевертыши.

**Контрольные вопросы**

1. Как описываются в языке Паскаль строковые величины?
2. В чем сходство и в чем различие между массивами и строками?
3. Существуют ли ограничения, накладываемые на длину строки?
4. Какие строковые процедуры существуют в языке Паскаль?
5. Для вывода значений каких строковых функций нужны переменные типа string, а для каких – integer?
6. Какие есть возможности извлечения из строки одного символа?
7. Строка для обработки процедурой должна быть ее параметром-аргументом или параметром-результатом?

# 7. ПРОГРАММИРОВАНИЕ ГРАФИКИ

Компьютер может оперировать либо с символами (символьный режим), либо с геометрическими фигурами (графический режим). Для создания на экране геометрических фигур с помощью языка Паскаль используются графические операторы, которые позволяют рисовать различные геометрические фигуры (точки, отрезки, окружности и пр.).

Графические объекты могут находиться на экране только в то время, когда работает программа. Для формирования графических изображений в системе TurboPascal предназначен стандартный библиотечный модуль GRAPH. В нем содержится 79 графических процедур, функций, десятки стандартных констант и типов данных. Все они составляют единый комплекс средств, позволяющих разрабатывать профессиональные программные продукты. Подключение модуля GRAPH к пользовательской программе осуществляется стандартным способом - с помощью зарезервированного слова USES: USES GRAPH. С этого момента все графические средства доступны пользователю.

Взаимодействие программы и видеосистемы в графических режимах обеспечивают драйверы. Драйверы собраны в файлах, имеющих расширение BGI: CGA.BGI, EGAVGA.BGI, HERC.BGI, IBM 8514.BGI, ATT.BGI , PC3270.BGI и др. Драйвер – это специальная программа, осуществляющая управление тем или иным техническим средством ПК. Графический драйвер управляет графическим адаптером в графическом режиме.

## 7.1. Инициализация графического режима

Графические возможности конкретного адаптера определяются разрешением экрана, т.е. общим количеством пикселей, а также количеством цветов. Кроме того, многие адаптеры могут работать с несколькими графическими страницами. Для инициализации графического режима используется процедура InitGraph (var Driver, Mode: integer; Path:string), где Driver – переменная типа integer , определяющая тип графического драйвера; Mode – переменная того же типа, задающая режим работы графического адаптера; Path – выражение типа string , содержащее путь доступа к файлу драйвера.

Program primer;  
Uses graph;   
Var   D , M : integer ; {переменные для установки драйвера и режима работы}   
Begin   
   D :=9;    M :=2;   
   InitGraph ( d , m , ‘здесь нужно указать путь к драйверу EGAVGA . BGI ’};

…………………………………………………………………………………………..

Hапример, InitGRAPH (d, m, 'G:\WORK\T7\BIN');

В этом примере показано, что драйвер egavga.bgi расположен в папке, путь к которой задан третьим параметром процедуры InitGRAPH: ‘G:\WORK\T7\BIN’. Если же драйвер находится в той же папке, что и программа пользователя, то можно на месте третьего параметра писать пустую константу.

Для новичков, которые могут не знать типа дисплея своего компьютера, имеется стандартная константа Detect. Если это значение присвоить параметру d (d:= Detect), то InitGRAPH автоматически инициирует нужный драйвер и устанавливает наиболее подходящий для дисплея режим работы.

Program primer ;   
Uses graph ;   
Var D , m : integer ; {переменные для установки драйвера и режима работы}   
Begin   
   D := detect ;   
   InitGraph ( d , m , ‘’};

………………………………………………………………………………………

С момента подключения модуля GRAPH программисту доступны все находящиеся в ней подпрограммы. В первую очередь вызывается процедура InitGRAPH, которая устанавливает один из возможных графических режимов.

Когда все запланированные графические работы выполнены, то необходимо выйти из графического режима (закрыть графический режим работы монитора). Это делается с помощью процедуры без параметров CloseGRAPH.

*Запомните! Любая программа, использующая графический режим, будет иметь одну и ту же структуру:*

* ***определение графического драйвера;***
* ***установка графического режима;***
* ***инициализация графического режима;***
* ***построения;***
* ***закрытие графического режима.***

Ниже следует стандартная форма оформления Паскаль – программы работы с графикой:

USES Crt, GRAPH;

VAR d, m: integer;

BEGIN

d:=Detect; InitGRAPH (dvar, mvar, '');

<. . . текст программы . . .>

Readln;

CloseGRAPH

END.

## 7.2. Простейшие графические операторы (процедуры) языка TurboPascal

Графические операторы языка Паскаль основаны на операторах вызова процедур (c параметрами и без параметров) модуля GRAPH, и все они делятся, в зависимости от их функционала, на следующие группы:

1. Работа с экраном.
2. Установка цвета.
3. Рисование графических примитивов (изображений).
4. Работа с текстом.
5. Организация анимаций.

Рассмотрим в порядке следования указанных действий наиболее распространенные процедуры и функции системы программирования TurboPascal 7.0.

* 1. **procedure InitGRAPH (dvar, mvar, path)**

Инициализирует графический режим работы дисплея.

* 1. **procedure ClearViewport**

Очищает экран монитора от имеющихся на нем ранее построенных изображений.

* 1. **procedure MoveTo (X, Y)**

Перемещает текущий указатель (CP) в точку с координатами X,Y. Например, процедура MoveTo (200,100) переместит текущий (невидимый) указатель в точку экрана с координатами (200, 100).

* 1. **procedure CloseGRAPH**

Закрывает графический режим работы монитора (переход в символьный режим).

* 1. **procedure SetColor (Цвет: word)**

Устанавливает основной цвет, которым будет осуществляться рисование изображений. Цвет задается либо своим английским названием, либо его числовым кодом.

* 1. **procedure SetBkColor (Цвет: word)**

Устанавливает цвет фона, на котором рисуется изображение, при этом надо помнить о том, что совпадение цветов фона и цвета изображения приводит к. невидимости на экране этого изображения.

* 1. **procedure SetFillStyle ( Шаблон: word, Цвет: word )**

Устанавливает шаблон заполнения и цвет. "Шаблон" (шаблон заполнения) может принимать значения от 1 до 12, например: 0 - заполнение цветом фона, 1 - заполнение текущим цветом.

* 1. **procedure FloodFill (X, Y: integer, Граница: word)**

Заполняет область ограниченную цветом "Граница", используя для этого текущий шаблон и цвет заполнения. Здесь X,Y задают координаты любой точки, расположенной внутри закрашиваемой области.

* 1. **procedure PutPiXel (X, Y, Цвет)**

Рисует точку (пиксел) с координатами X,Y.

* 1. **procedure Line (X, Y, X1, Y1: integer)**

Рисует прямую линию от точки с координатами X,Y к точке X1,Y1.

* 1. **procedure LineTo (X, Y: integer)**

Рисует прямую линию из точки, в которой находится текущий указатель, в точку с координатами (X,Y).

* 1. **procedure LineRel (DX, DY: integer)**

Рисует прямую линию из точки (X, Y), в которой находится текущий указатель, в точку с координатами (X+DX,Y+DY).

* 1. **procedure Rectangle (X, Y, X1, Y1: integer)**

Рисует прямоугольник, где: (X,Y)-координаты верхнего левого угла, (X1,Y1) - координаты правого нижнего угла прямоугольника (координаты главной диагонали).

А(X,Y)

|  |
| --- |
|  |

B(X1,Y1)

* 1. **procedure Bar (X1, Y1, X2, Y2: integer)**

Рисует полосу, используя текущий цвет и стиль. Иногда эту процедуру называют рисованием закрашенного прямоугольника.

А(X1,Y1)

|  |
| --- |
|  |

B(X2,Y2)

* 1. **procedure Circle (X, Y, radius: integer)**

Рисует окружность указанного радиуса с центром в точке (X,Y).

* 1. **procedure Arc (X, Y: integer; Angle1, Angl2, R: word)**

Вычерчивает дугу окружности от начального угла Angle1 до конечного угла Angle2 с радиусом R. Точка (X, Y) используется как центр соответствующей дуге окружности.

* 1. **procedure Ellipse (X, Y: integer; Angle1, Angl2, RX, RY: word)**

Вычерчивает дугу эллипса от начального угла Angle1 до конечного угла Angle2 с радиусом R. Точка (X, Y) используется как центр соответствующего дуге эллипса, а RX, RY – горизонтальная и вертикальная оси.

* 1. **procedure FillEllipse (X, Y: integer; RX, RY: word)**

Вычерчивает закрашенный эллипс. Точка (X, Y) используется как центр эллипса, а RX, RY задают его горизонтальную и вертикальную оси.

* 1. **procedure Sector (X, Y: integer; Angle1, Angl2, RX, RY: word)**

Вычерчивает и заполняет эллиптический сектор.

* 1. **procedure OutText (TextString: String)**

Выводит на экран строку (текст) в то место, в котором находится текущий указатель.

* 1. **procedure OutTextXY (X, Y: integer; TextString: String)**

Выводит на экран строку (текст) в точку (X, Y).

* 1. **function ImageSize (X1, Y1, X2, Y2: integer): word**

Возвращает (запоминает в буфере обмена) число байт прямоугольной области экрана (X1, Y1) – (X2, Y2), внутри которой находится изображение. Эта функция необходима для функции GetImage, которая сохраняет двоичный образ заданной области экрана в динамической памяти компьютера.

|  |  |
| --- | --- |
| |  | | --- | |  |   (X1,Y1)  (X2, Y2) |

* 1. **function GetMem (BitMap:pointer; Size: word)**

Выделяет место в динамической памяти под хранение фрагмента изображения. Здесь BitMap - переменная типа pointer хранит ссылку на начало «кучи», где будет храниться фрагмент, а Size - размер фрагмента, полученный с помощью ImageSize.

* 1. **procedure PutImage (X, Y: integer; var BitMap: pointer; Mode: word)**

Возвращает образ фрагмента из памяти на экран монитора, где X, Y - координаты точки экрана (левый верхний угол прямоугольника), начиная с которой будет выводиться фрагмент Параметр Mode определяет, какой двоичный оператор будет использоваться для вывода двоичного образа области на экран. Например, PutImage (X, Y, BitMap, NormalPut);

|  |  |
| --- | --- |
| |  | | --- | |  |   (X, Y) |

* 1. **procedure GetImage (X1, Y1, X2, Y2: integer; var BitMap: pointer)**

Сохраняет образ фрагмента в динамической памяти, где X1, Y1, X2, Y2 - координаты фрагмента, а BitMap - нетипизированная ссылка.

## 7.3. Основные приемы работы с графикой

Рассмотрим теперь основные приемы работы с графикой, которые включают в себя работу с цветом, вывод точек, построение плоских и объемных прямоугольных фигур, построение окружностей и эллипсов, а также их дуг и секторов, вывод текста и организацию эффекта анимации (движения объектов по экрану).

### 7.3.1. Работа с цветом

**Установка цвета изображения.** Драйвер egavga.bgi позволяет использовать 16 цветов. Каждому цвету присвоен код – целое число, которое используется процедурами и функциями.

Цвет выводимых в графическом режиме на экран линий и символов можно задать процедурой SetColor (Сolor: word), аргумент которой – целое число от 0 до 15 или имя одной из приведенных выше констант. Установка цвета действует на те линии и тексты, которые выводятся после ее вызова, но не меняет цвет линий и символов, выведенных на экран ранее. Таким образом, процедуру SetColor следует вызывать каждый раз перед выбором нового цвета. Если цвет не установлен, то используется белый цвет.

**Установка цвета фона.** Чтобы установить цвет фона для всего экрана, используется процедура SetBkColor (Color: word). Если процедура установки цвета фона не вызвана, экран будет черным.

Работа с цветом осуществляется, прежде всего, на базе названия и кодов цветов, используемых в соответствующих процедурах языка. Кодировка используемых цветов представлена в таб. 1.

Таб. 1. Цветовая шкала

|  |  |  |  |
| --- | --- | --- | --- |
| **Цвет** | **Код** | **Цвет** | **Код** |
| Black  Blue  Green  Cyan  Red  Magenta  Brown  LightGray | 0  1  2  3  4  5  6  7 | DarkGray  LightBlue  LightGreen  LightCyan  Lightred  LightMagenta  Yellow  White | 8  9  10  11  12  13  14  15 |

**Цвет и стили заполнения.** В графике TurboPascal работа с цветом осуществляется по-разному. Одним из приемов является окраска точки и линий (прямая, окружность, эллипс) с помощью процедуры Color. Другим – заполнение внутренних областей фигур с помощью так называемого наполнения, которое включает в себя как цвет, так и стиль заполнения.

Hекоторые операторы, например Bar, служат для построения закрашенных фигур. Цвет заполнения для этих фигур задается процедурой SetFillStyle (Стиль: word; Цвет: word), которая устанавливает шаблон заполнения и цвет. Cтандартные стили заполнения представлены в таб. 2.

Таб. 2. Стандартные стили заполнения

|  |  |
| --- | --- |
| **Значение** | **Маска** |
| 0  1  2  3  4  5  6  7  8  9  10  11 | Заполнение цветом фона  Заполнение текущим цветом  Заполнение символами --, цвет - color  Заполнение символами // нормальной толщины, цвет - color  Заполнение символами // удвоенной толщины, цвет - color  Заполнение символами \\ удвоенной толщины, цвет - color  Заполнение символами \\ нормальной толщины, цвет - color  Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет - color  Заполнение штриховкой крест-накрест по диагонали "редкими" тонкими линиями, цвет - color  Заполнение штриховкой крест-накрест по диагонали "частыми" тонкими линиями, цвет - color  Заполнение "редкими" точками  Заполнение "частыми" точками |

Перед выводом на экран какого-либо изображения иногда требуется очистить экран. Для этого используется процедура ClearViewPort. Есть еще и процедура ClearDevice, которая очищает графический экран и устанавливает указатель в левый верхний угол.

### 7.3.2. Заполнение (закрашивание) произвольной замкнутой фигуры

Процедура **FloodFill** (X, Y, border) заполняет произвольную замкнутую фигуру, используя текущий стиль и цвет заполнения. Координаты точки (X, Y) указывают, начиная с какой точки области будет производиться заливка. Если точка находится внутри замкнутой фигуры, то будет закрашена внутренняя область. Если фигура не замкнута, то заливка разольется по всему экрану. Параметр border указывает цвет граничной линии. Например, изображение некоторой замкнутой фигуры можно закрасить при помощи двух операторов: SetFillStyle (5, Red); FloodFill (320, 310, Cyan). Здесь цветом Red фигура заполняется символами **\\** удвоенной толщины, а граница – цветом Cyan. Точка (320, 310) является некоторой внутренней точкой этой области.

### 7.3.3. Построение простейших геометрических фигур

**Линии и точки.** Для построения изображений часто бывает необходимо задавать стартовую точку этого построения. Для этой цели в графике Паскаля предусмотрено понятие «Текущий указатель» (CP – CURRENT POINTER), который определяет положение «невидимого курсора». Для перемещения СР в границах экрана монитора, который, как мы знаем, представляет собой первую четверть декартовой системы координат, имеется ряд процедур и функций. В первую очередь это процедура MoveTo (X, Y), которая перемещает текущий указатель (CP) в точку с координатами X,Y. Например, процедура MoveTo (200,100) переместит текущий (невидимый) указатель в точку экрана с координатами (200, 100).

Для рисования прямых линий применяются три процедуры:

* Line (X,Y, X1, Y1:integer) рисует линию от точки с координатами (X, Y) к точке (X1,Y1);
* LineTo (X, Y) строит линию из точки текущего положения указателя в точку с координатами X,Y;
* LineRel (dX, dY) строит линию от точки с текущим положением указателя в точку (CpX + dX, CpY + dY), где CpX и CpY - текущие координаты Cp. Цвет линии задается оператором SetColor (Цвет).

TurboPascal позволяет вычерчивать линии самого различного стиля: тонкие, широкие, штриховые, пунктирные и т.д. Установка стиля производится процедурой SetLineStyle (Стиль: word; pattern: word; Толщина: word). Стили линий представлены в таблице 3.

Таб.3. Стили линий

|  |  |  |
| --- | --- | --- |
| **Константа** | **Значение** | **Описание** |
| 0  1  2  3 | SolidLn  DottedLn  CenterLn  DachedLn | Непрерывная линия  Линия из точек  Линия из точек и тире  Штриховая линия |

Толщина линии представлена в таблице 4.

Таб. 4. Толщина линии

|  |  |  |  |
| --- | --- | --- | --- |
| **Константа** | | **Значение** | **Описание** |
| 1  3 | SolidLn  TchickWidth | | Нормальная толщина  Жирная линия |

Какие бы изображения не выводились на экран, все они построены из точек. Для вывода на экран точки используется процедура РutPixel (X, Y, Цвет), которая рисует на дисплее точку (пиксел) с координатами X,Y и цветом "Цвет".

Ниже следует программа, которая заполняет экран монитора точками, цвет которых и их расположение задается датчиком случайных чисел Randomize:

Uses Crt, GRAPH;

Var X, Y, c, d, m: Integer;

вegin

d:=Detect; InitGRAPH(d, m, '');

randomize; ClearViewPort;

while not keypressed do

begin

Delay (100);

X:=Random (640);

Y:= Random (480);

c:= Random (15);

putpiXel (X, Y, c)

end;

CloseGRAPH;

end.

**Прямоугольники.** Для построения прямоугольных фигур имеется несколько процедур. Первая из них - процедура Rectangle (X1, Y1, X2, Y2) вычерчивания прямоугольника цветом, заданным предварительно оператором SetColor. Для построения закрашенного прямоугольника используется процедура: Bar (X1, Y1, X2, Y2). Помимо прямоугольников возможно построение и пространственных прямоугольных фигур с помощью процедуры Bar3D (X1, Y1, X2, Y2 : integer, Depth: Word; Top: Boolean), где X1, Y1, X2, Y2 определяют положение прямоугольника, являющегося «передней» гранью пространственной фигуры, а параметр Depth определяет "глубину" 3-х мерного прямоугольника, который обычно равен (X2 - X1 + 1) div 4. (Куб). Параметр Top определяет положение вершины прямоугольника относительно передней грани. Цвет заполнения для Bar и Bar3D устанавливается процедурой SetFillStYle.

**Дуги и окружности.** Процедура вычерчивания окружности текущим цветом имеет следующий формат: Circle (X, Y, Radius: word) где X и Y - центр окружности, а Radius - ее радиус. В ряде случаев, в частности для создания псевдообъемных фигур, используются дуги. Их можно вычертить с помощью процедуры Arc (X, Y: integer; StAngle, EndAngle, Radius: word) где X,Y - центр окружности, StAngle и EndAngle - начальный и конечный угол дуги, заданный в градусах, Radius - радиус дуги.

**Эллипсы и сектора.**  Для построения эллиптических дуг предназначена процедура Ellipse(X,Y, StAngle, EndAngle, XR, YR), где X, Y - центр эллипса в дисплейных координатах, XR и YR - горизонтальная и вертикальная оси. Дуга эллипса вычерчивается от начального угла StAngle до конечного угла EndAngle текущим цветом. Фон внутри эллипса совпадает с фоном экрана. Исходя из математических фактов связи эллипса с окружностью, можно рисовать дуги и окружности (а не эллипса), если взять в этой процедуре равными значения XR и YR. С другой стороны, для построения полного эллипса в процедуре Ellipse(X,Y, StAngle, EndAngle, XR, YR) нужно положить StAngle = 0, EndAngle=360.

Чтобы создать закрашенный эллипс, используется специальная процедура FillEllipse (X, Y, XR, YR), где X,Y - центр эллипса в дисплейных координатах, XR и YR - горизонтальная и вертикальная оси. Заполнитель устанавливается процедурой SetFillStyle. Очевидно, что для вычерчивания не закрашенного эллипса необходимо в процедуре SetFillStyle указать цвет фона.

Можно создать и заполнить сектор в эллипсе. Для этого используется процедура Sector (X,Y, StAngle, EndAngle, XR, YR), где X,Y - центр, XR, YR - горизонтальный и вертикальный радиусы, StAngle и EndAngle - начальный и конечный угол (в градусах).

Ниже представлена программа работы с различными типами линий и закрашивания фигур различными цветами.

Uses GRAPH;

Var i, d: integer;

begin

d := detect;

InitGRAPH (d, i, 'G/Work/t7'); {Инициализация графики}

SetColor (White);

SetLineStyle (SolidLn, 0, NormWidth);

SetFillStyle (SolidFill, Red);

Sector (100, 200, 45, 135, 80, 80);

{Закрашенный красным цветом сектор с центром (100, 200),

начальным углом 45, конечным углом 135 и радиусом 80}

SetColor (Red);

SetLineStyle (SolidLn, 0, ThickWidth);

SetFillStyle (SolidFill, Yellow);

Rectangle (300, 10, 400, 70);

FloodFill (350, 50, Red);

{Рисование прямоугольника сплошной толстой красной линией, закрашенного желтым цветом}

SetColor (Cayan);

SetLineStyle (SolidLn, 0, NormWidth);

SetFillStyle (7, Red);

Cercle (500, 250, 50);

FloodFill (500, 250, Cayan);

{Рисование круга голубой линией стандартной толщины, закрашенного сетчатой штриховкой красного цвета}

Readln;

CloseGRAPH

end.

**Вывод текста.** Выводимые на экран изображения обычно сопровождаются пояснительным текстом. В графических режимах для этого используются процедуры OutText и OutTextXY. Процедура OutText (Text) выводит строку текста, начиная с текущего положения Cp. Процедура OutTextXY (X, Y, Text), где X, Y - координаты точки начала вывода текста (левый верхний угол выводимого текста), Text - константа или переменная типа string.

Качественное оформление самого текста требует при его выводе использования самых различных шрифтов. Установить нужный шрифт можно процедурой: SetTextStyle (Font: word; Direction: word; CharSize: word), где Font - выбранный шрифт, Direction – направление (горизонтальное - Direction = 0, вертикальное - Direction = 1), СharSize - размер выводимых символов. Система TurboPascal имеет свои собственные шрифты, отличные от классических текстовых редакторов. Шрифты вывода текста в графике Паскаля представлены в таб. 5.

Таб. 5. Таблица шрифтов

|  |  |  |
| --- | --- | --- |
| **Константа** | **Значение** | **Описание** |
| 0  1  2  3  4 | DefaultFont  TriplexFont  SmaillFont  SanSerifFont  GothicFont | 8\*8 битовый шрифт Штриховые шрифты  Малый шрифт  Сансериф  Готический |

Нужную величину шрифта можно установить еще одним способом - с помощью процедуры SetUserCharSize (multX, divX, multY, divY: word). Первые два параметра управляют горизонтальным размером, два последних - вертикальным. Если взять за 1 ширину стандартного шрифта, то отношение multX/divX будет устанавливать ширину.

В данной таблице даны шрифты размером 0 и имеющие горизонтальную ориентацию.

Приведем пример программы, которая демонстрирует применение различных шрифтов и их размеров при выводе текста.

Uses GRAPH;

Var i, d: integer;

begin

d := detect;

InitGRAPH (d, i, 'G/Work/t7'); {Инициализация графики}

{Горизонтальный текст}

SetColor (Blue);

SetTextStyle (DefaultFont, HorizDir, 3);

OutTextXY (5, 1, ‘DefaultFont размером 3’); {Шрифт Default синего цвета}

SetColor (Red);

SetTextStyle (TriplexFont, HorizDir, 1);

OutTextXY (5, 3, ‘TriplexFont размером 1’); {Шрифт Triplex красного цвета}

SetColor (Yellow);

SetTextStyle (SmaillFont, 0, 4);

OutTextXY (5, 5, ‘SmaillFont размером 4’); {Шрифт Smaill желтого цвета}

{Вертикальный текст}

SetColor (Green);

SetTextStyle (DefaultFont, VertDir, 3);

OutTextXY (20, 7, ‘DefaultFont размером 3’); {Шрифт Default зеленого цвета}

SetColor (Blue);

SetTextStyle (TriplexFont, 1, 1);

OutTextXY (5, 3, ‘TriplexFont размером 1’); {Шрифт Triplex синего цвета}

SetColor (Yellow);

SetTextStyle (GothicFont, VertDir, 2);

OutTextXY (5, 5, ‘GothicFont размером 2’); {Шрифт Smaill желтого цвета}

Readln;

CloseGRAPH

end.

**Средства работы с фрагментами. Эффект движения в графическом режиме.** В дополнение к уже рассмотренным ранее графическим средствам построения и вывода изображений имеется еще ряд дополнительных процедур, позволяющих сохранять и восстанавливать отдельные фрагменты изображения. Для манипулирования фрагментом надо в первую очередь узнать его размер. Это делается с помощью функции ImageSize (X1, Y1, X2, Y2), где X1, Y1, X2, Y2 - координаты верхнего левого и правого нижнего угла прямоугольной области экрана. Функция возвращает размер указанного фрагмента в байтах. Затем в памяти машины выделяется место под фрагмент. Для этого пользуются процедурой GetMem (BitMap, Size), где BitMap - переменная типа pointer, Size - размер фрагмента, полученный с помощью ImageSize.

Сохранение образа фрагмента в памяти выполняется процедурой GetImage (X1, Y1, X2, Y2, BitMap), где X1, Y1, X2, Y2 - координаты фрагмента, BitMap – нетипизированная ссылка. Возвратить образ фрагмента из памяти на экран можно с помощью процедуры PutImage (X, Y, BitMap, Mode), где X, Y - координаты точки экрана, начиная с которой будет выводиться фрагмент, BitMap - уже упомянутая нетипизированная константа, которая задает высоту и ширину фрагмента. Параметр Mode определяет, какой двоичный оператор будет использоваться для вывода двоичного образа области на экран. Например, PutImage(X, Y, BitMap, NormalPut).

Описанными выше процедурами можно пользоваться для создания на дисплее каких-либо движущихся объектов. Ниже приведена подобная программа.

Uses Crt, GRAPH;

Var i, d, Size: integer; p: pointer;

begin

d := detect;

InitGRAPH (d, i, 'G/Work/t7'); {Инициализация графики}

SetColor (Red);

Circle (30, 232, 10); {Рисование окружности красного цвета}

SetFillStyle (1, Red);

FloodFill (30, 232, Red); {Закраска круга красным цветом}

Size := ImageSize (18, 220, 42, 244); {«Измерение» круга}

GetMem (р, Size); {Выделение в памяти места для круга}

GetImage (18, 220, 42, 244, р^); {Запись в память двоичного кода круга}

While not KeyPressed do

begin

For i:=18 to 600 do {Плавное перемещение круга слева направо}

PutImage (i, 220, р^, NormalPut);

For i:= 600 downto 18 do {Плавное перемещение круга справа налево}

PutImage (i, 220, р^, NormalPut);

end;

end.

Эта программа реализует перемещение красного шарика от левого края экрана до правого и обратно. Процесс прекращается по нажатию любой клавиши.

## Лабораторная работа №7

**Цель работы**: освоить графические процедуры и функции; научиться создавать программы построения на экране различных изображений, заполнять их цветом; освоить технологию вывода на экран текста в графическом режиме; организовывать движение графических объектов по монитору.

**Общие сведения**

Для обработки графической информации нужно уметь инициализировать графический режим работы монитора, знать все процедуры и функции построения простейших графических примитивов и уметь их применять для построения составных (композиционных) фигур. Движение фигур по экрану основывается на «запинании» некоторого образа в динамической памяти и последующего его вызова (как правило, в цикле) на экран монитора в последовательных точках, что создает иллюзию его движения. Следует внимательно изучить порядок вызова для этой цели соответствующих процедур и функций.

Перед выполнением работы необходимо ознакомиться с теоретическим материалом по теме «Программирование графики».

**Пример 1.** Составить программу построения узора, образованного пересечением двух семейств расходящихся отрезков, у которых k1 и *k2* - расстояния между отрезками слева и справа, а *h* – смещение-наклон вниз (вверх) всего семейства.

**Решение**. Построение горизонтальной линии осуществляется процедурой line (0, k, 640, k). Если нужно построить n линий, то эту процедурe надо поместить в цикл: for i:=1 to n do line (0, i\*k, 640, i\*k). Наклон линий получается добавлением шага h ко второй координате (h>0, то наклон вниз, h<0, то наклон вверх): for i:=1 to n do line (0, i\*k, 640, i\*k+h). Расширение и сужение прямых ljcnbuftncz путем задания различных значений k1 и k2 величины k: for i:=1 to n do line (0, i\*k1, 640, i\*k2+h). Второе семейство получается, если наклон линий производить по первой координате: for i:=1 to n do line (0, i\*k2+h, 640, i\*k1).

Uses crt, graf;

Var gD, gM, i, k1, k2, h: integer;

Begin

k1:= 8; k2 := 3; h:=110; gD:= Detect;

InutGraf (gD, gM, ‘’); setcolor(red);

For i:= 1 to (480 div k1) do

Begin

for i :=1 to n do line (0, i\*k1, 640, i\*k2+h):

for i :=1 to n do line (0, i\*k2+h, 640, i\*k1);

end;

readln; Closegraph

end.

**Пример 2.** Построить график функции y = x2  на отрезке [a, b].

**Решение**. Для построения графика функции нужно разбить отрезок [a, b] на n частей (h – шаг разбиения (b-a)/n), а затем проводить с помощью процедур line и lineto отрезки прямой, который и образуют ломаную линию графика функции. Можно, конечно воспользоваться и выводом точки (точечное построение графика). В программе построим функцию вычисления ее значений в произвольной точке и процедуру построения по точкам ломаной линии, представляющей собой график функции.

Uses crt, graf;

Var gD, gM, n: integer;

a, b: real;

function f (x: real) : real;

Begin

f:= x\*x

End;

Procedure graf (x0, x1, y0, y1, n: word; a, b: real);

Var h, m, x, t1, t2 :real;

i, u, v, xv, yv: word;

begin

h:= (b-a)/n; {Нахождение шага разбиения}

m:= abs(f(a)); {Поиск значения функции в левом конце отрезка}

t1:= (x1-x0) / (b-a); t2:= (y1-y0) / (2\*m);

{Построение координатных осей}

setfillstyle (1,15); bar( x0-5, y0-5, x1+5, y1+5);

xv:= round(x0-a\*t1); yv:= round((t0+y1) / 2);

line (x0, yv, x1, yv);

{Установка текущего курсора в начало графика}

Moveto (x0, yv-round (f (a)\*t2);

{Построение графика}

Setcolor (3);

For i:= 1 to n do

Begin

x:= a+i\*h; u:= x0+round ( (x-a)\*t1);

v:= yv:= round (f (x)\*t2); lineto (u, v);

end;

end; {Конец процедуры}

begin

clrscr; readln (a, b, n); gD:= Detect;

Initgraph (gD, gM, ‘’);

graf (100, 500, 50, 300, n, a, b);

Readln; Closegraph

end.

**Пример 3.** Вращение Земли вокруг Солнца.

**Решение.** Организуем движение точки (Земли) по окружности, в центре которой размещается круг (Солнце). Установку точки на орбите можно осуществить по параметрическим формулам окружности:

X0 := 320 + r\*sin(A);

Y0 := 240 + r\*cos(A),

где r1 – радиус орбиты Земли, А- параметрический угол, меняющийся от 0 до 360 градусов. Чтобы организовать движение, достаточно в цикле устанавливать точку с координатами (x0, y0) для всех углов, принимающих значение от 0 до 360 градусов с шагом h.

Uses Crt, GRAPH;

Var i, d: integer;

fi, h, ; real;

x0, y0, r1: integer;

begin

d := detect;

InitGRAPH (d, i, 'G/Work/t7'); {Инициализация графики}

SetColor (Red);

Circle (320, 240, 10); {Рисование окружности красного цвета}

SetFillStyle (1, Red);

FloodFill (320, 240, Red); {Закраска круга (Солнца) красным цветом}

R1:= 100; h:=5; fi:= 0;

Repeat

X0:= round (r1\*sin (fi)) +320;

Y0:= round (r1\*cos(fi)) +240;

fi:= fi+2\*pi\*h/360;

Putpixel (x0, y0, 15);

delay(50);

Putpixel ((x0, y0, 0);

Until keypressed;

CloseGraph

end.

**Варианты заданий**

**Задание 1.** Составить программу моделирования паркетов из:

1. прямоугольных треугольников;
2. равносторонних треугольников;
3. ромбов;
4. трапеций;

**Задание 2.** Разработайте программу для построения следующих графиков функций:

1. y = x3 ;
2. y= ex ;
3. y = ln (x);
4. y = sqrt (x)).

**Задание 3.** Разработайте динамическую модель:

1. Солнца для двух планет;
2. Солнца для Земли и Луны;
3. столкновение двух ядер;
4. старт ракеты.

**Контрольные вопросы**

1. Какой процедурой выполняется установка типа линий?
2. Как построить ломаную линию?
3. Какой процедурой выполняется установка типа закраски?
4. Какой процедурой выполняется установка шрифта для вывода текста?
5. Какой процедурой можно вывести на экран изображение куба?
6. Как нарисовать кольцо указанного цвета?
7. Можно ли процедурой Ellipse вывести на экран окружность?
8. Каким образом строятся сектора (круговые и эллиптические)?
9. Каким образом можно из рисунка выделить его фрагмент и вставить в другой рисунок?
10. С помощью каких процедур и функций осуществляется эффект движения изображения по экрану?

**Ли****тература**

1. Касторнов А.Ф. Методическое пособие по курсу "Алгоритмический язык Паскаль". - Череповец, 1995.
2. Касторнова В.А., Касторнов А.Ф. Демонстрационно-обучающий курс “Алгоритмический язык Паскаль”. // Информатика и образование. - 1998. - №2.-С.48-61.
3. Немнюгин С.А. Turbo Pascal: Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2007.
4. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2004.
5. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Практикум. – СПб.: Питер, 2006.
6. Ставровский А.Б. Турбо Паскаль 7.0: Учебник для вузов. – Киев, 2001.
7. Фаронов В.В. Система программирования Delphi. – СПб., 2006.
8. Фаронов В.В. Турбо Паскаль 7.0: Практика программирования: учебное пособие.-М.: ОМД Групп, 2003.
9. Программирование на языке Паскаль: задачник / под ред. Усковой О.Ф. – СПб.: Питер, 2002.
10. Юркин А.Г. Задачник по программированию. – СПб.: Питер, 2002.

**----------------------------------------------------------------------------------------------------------------**

Российский портал информатизации образования [содержит: законодательные и нормативные правовые акты государственного регулирования информатизации образования, федеральные и региональные программы информатизации сферы образования, понятийный аппарат информатизации образования, библиографию по проблемам информатизации образования, по учебникам дисциплин цикла Информатика, научно-популярные, документальные видео материалы и фильмы, периодические издания по информатизации образования и многое другое.](http://portalsga.ru)

