ФГБОУ ВПО «Воронежский государственный
технический университет»

Кафедра высшей математики
и физико-математического моделирования

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ТУРБО ПАСКАЛЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по курсам
«Информатика» и «Специальные главы информатики»
для студентов направлений 150400 «Металлургия»,
140700 «Ядерная энергетика и теплофизика» и
131000 «Нефтегазовое дело» очной формы обучения

Воронеж 2014

Составители: канд. техн. наук С.А. Кострюков,
канд. техн. наук В.В. Пешков,
канд. физ.-мат. наук Г.Е. Шунин

УДК 004.42+004.43

Основы программирования на Турбо Паскале: методические указания к выполнению лабораторных работ по курсам «Информатика» и «Специальные главы информатики» для студентов направлений 150400 «Металлургия», 140700 «Ядерная энергетика и теплофизика» и 131000 «Нефтегазовое дело» очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. С.А. Кострюков, В.В. Пешков, Г.Е. Шунин. Воронеж, 2014. 56 с.

В методических указаниях кратко рассмотрены основные приемы создания простейших программ на языке Турбо Паскаль. Присутствует большое число разобранных программ, что позволяет использовать указания для всех форм обучения.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по направлениям 150400.62 «Металлургия», 140700.62 «Ядерная энергетика и теплофизика» и 131000.62 «Нефтегазовое дело», профилям «Технология литейных процессов», «Техника и физика низких температур» и «Эксплуатация и обслуживание объектов транспорта и хранения нефти, газа и продуктов переработки» по дисциплинам «Информатика» и «Специальные главы информатики».

Методические указания подготовлены на магнитном носителе в текстовом редакторе Microsoft Word 2003 и содержатся в файле MU-Pascal.doc.

Ил. 6. Табл. 2. Библиогр.: 5 назв.

Рецензент канд. физ.-мат. наук, доц. В.В. Ломакин

Ответственный за выпуск зав. кафедрой д-р физ.-мат. наук, проф. И.Л. Батаронов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

© ФГБОУ ВПО «Воронежский государственный технический университет», 2014

ВВЕДЕНИЕ

Несмотря на большое разнообразие и доступность современных математических программ, способных осуществлять решение широкого спектра задач, как в аналитическом, так и в численном виде, владение навыками алгоритмизации и программирования по-прежнему является необходимым для студентов любой технической специальности. Практически все современные математические пакеты, такие, как MathCAD, Maple, Mathematica и т.д., содержат встроенные средства программирования, использующие разветвления, циклы, подпрограммы и др. Для получения студентами навыков использования этих средств и освоения основных приемов программирования наилучшим образом подходит язык программирования высокого уровня Паскаль.

Паскаль был разработан Никлаусом Виртом в 1968-1970 гг. и поначалу предназначался для обучения программированию, однако затем, с появлением Турбо Паскаля, получил широкое распространение благодаря наглядности программ, удобству и легкости при изучении. Он послужил основой для разработки других языков программирования (Ада, Модула-2), на его основе создана популярная RAD-среда Delphi.

Наиболее популярная реализация языка Паскаль – Турбо Паскаль фирмы Borland International – появилась на рынке программных продуктов в 1984 году. В 1992 г. было выпущено два пакета программирования на языке Паскаль в среде MS DOS – это Borland Pascal 7.0 и Turbo Pascal 7.0.

Пакет Turbo Pascal 7.0 использует новейшие для своего времени достижения в программировании. Язык этой версии обладает широкими возможностями, имеет большую библиотеку модулей. Среда программирования позволяет создавать тексты программ, компилировать их, находить и исправлять ошибки, компоновать программы из отдельных частей, использовать модули, отлаживать и выполнять отлаженную программу.

Знакомство со средой программирования Turbo Pascal 7.0

Для начала работы с текстовым редактором системы программирования Turbo Pascal 7.0, нужно запустить на выполнение программу **turbo.exe** с помощью ярлыка на рабочем столе или непосредственно из файлового менеджера (Проводника или Total Commander). Программа находится в папке C:\TP7.

Основные команды

F10 – войти в основное меню;
F3 – открыть файл;
F2 – сохранить файл;
Alt+F9 – компилировать программу;
Ctrl+F9 – компилировать и запустить программу;
F8 – пошаговый режим выполнения программы при отладке;
Alt+F5 – просмотр результатов работы программы;
F6 – переход к следующему окну редактора Turbo Pascal;
Ctrl+F1 – справка Turbo Pascal (алфавитный указатель)
Alt+X – выход из редактора Turbo Pascal.

Команды для работы с текстом

Большинство команд для работы с фрагментами текста (блоками) выполняются в два приема. Например, при копировании выделенного блока нужно нажать одновременно клавиши **Ctrl** и **K**, а затем, отпустив их, клавишу **C**.

Ctrl+K, B – отметить начало выделяемого блока;
Ctrl+K, K – отметить конец выделяемого блока.

Также выделить блок можно с помощью мыши или с помощью стрелок при нажатой и удерживаемой клавише **Shift**.

Ctrl+K, C – копировать выделенный блок в текущую позицию курсора;

Ctrl+K, V – переместить выделенный блок в текущую позицию курсора;

Ctrl+K, H – снять выделение блока;

Ctrl+K, Y – удалить выделенный блок.

Ctrl+Y – удалить текущую строку.

Кроме этого, можно использовать буфер обмена:

Ctrl+Insert – копировать выделенный блок в буфер обмена;

Shift+Insert – вставить блок из буфера обмена в текущую позицию курсора.

Лабораторная работа №1
Структура программы на Паскале.
Организация ввода-вывода данных.
Использование стандартных функций.

Любая программа (или программная единица — процедура или функция) на Паскале состоит из трех основных частей:

1. Раздел заголовка (объявления программной единицы);
2. Разделы описаний;
3. Раздел операторов (тело программы).

В разделе заголовка содержится одна-единственная строчка, которая указывает компилятору, что он имеет дело именно с программой (процедурой или функцией) и, более того, с программой под определенным именем. Для программ эта срока начинается с зарезервированного слова **Program**, после которого следует собственно имя программы. В конце обязательно ставится точка с запятой. Раздел заголовка для основной программы является необязательным.

В разделах описаний должны содержаться описания всех идентификаторов, используемых в разделе операторов или нижележащих разделах описаний. Разделы описания в общем случае включают:

- раздел описания используемых библиотек **Uses**;
- раздел описания меток **Label**;
- раздел описания констант **Const**;
- раздел описания типов **Type**;
- раздел описания переменных **Var**;
- раздел описания процедур и функций.

Разделы **Type**, **Const**, **Var**, **Label** могут следовать друг за другом в любом порядке и встречаться в разделе описаний сколько угодно раз. Единственное ограничение – идентификаторы, которые используются для определения других идентификаторов, должны описываться раньше. В простейшей программе из всех перечисленных подразделов раздела описания обычно присутствует только раздел описания переменных **Var**.

Тело программы (раздел исполняемых операторов) содержит собственно программный код, отвечающий за реализацию алгоритма. При этом тело программы обязательно оформляется так называемыми *операторными скобками* – словами **Begin**, **End**. Т.е. все операторы, реализующие ваш алгоритм, должны помещаться между ними.

Все разделы программы, кроме раздела операторов, могут отсутствовать, если в них нет нужды.

Таким образом, структура программы на Турбо Паскале имеет вид:

Program <*имя программы*>; {*например: program First;*}
Uses <*список библиотек*>; {*например: uses CRT;*}
Label <*список меток*>; {*например: Label Home;*}
Const <*имя константы*>=<*значение константы*>; {*например: const n=7;*}
Type <*имя типа*>=<*описание типа*>; {*например: type mass=array[1..3, 1..5] of real;*}
Var <*имя переменной*>:<*тип данных*>; {*например: var x: real;*}
Раздел описания процедур и функций. Каждая процедура должна начинаться с ключевого слова **Procedure**, а каждая функция – со слова **Function**.

```
Begin
<операторы программы>
End.
```

Ввод и вывод данных в Паскале полностью реализованы с помощью стандартных процедур.

Для ввода данных с клавиатуры служат процедуры **Read** и **ReadLn**. Примеры:

```
Read(a);
ReadLn(x,y,z);
```

При вводе нескольких значений одной процедурой их разделяют пробелом, пробелами или нажатием клавиши <*Enter*>.

При вводе с клавиатуры используют обычно **ReadLn**, а процедура **Read** предназначена для работы с внешними файлами. Особенность процедуры **ReadLn** в том, что она вызыва-

ет **Read** для ввода указанных переменных, а затем, игнорируя остаток введенной строки, переходит к началу следующей строки (*Ln* – сокращение от *Line – строка*).

Для вывода данных на дисплей применяются процедуры **Write** и **WriteLn**. Примеры:

```
Write(a);
WriteLn(x, '   ', y); {между x и y выводится несколько
                        пробелов, взятых в апострофы}
Write('Максимальное значение: ', max);
```

При использовании **WriteLn** после вывода указанных в скобках величин происходит переход на новую строку, чтобы следующий вывод данных шел с новой строки. **Write** не заканчивает строку, и следующая выводимая величина будет печататься в той же строке.

При выводе данных допустимо использование *шаблона вывода*. После выводимой величины ставится двоеточие и указывается число позиций, которые отводятся под нее в текущей строке экрана. Для вещественных чисел также можно указать число позиций под дробную часть числа:

```
WriteLn(x:9:6); { отводится 9 знаков, из них 6 после точки}
```

Обычно при вводе с клавиатуры исходных данных программы предварительно выводят на экран подсказку – что именно пользователь должен ввести. Примеры:

```
Write('Введите x: '); ReadLn(x);
Write('Введите число элементов массива (от 1 до 10):');
ReadLn(n);
```

Основные стандартные функции

Математические функции (тип результата вещественный):

Abs(x)	$\lvert x \rvert$	Sin(x)	$\sin x$
Sqr(x)	x^2	Cos(x)	$\cos x$
Sqrt(x)	$\sqrt{x}$	ArcTan(x)	$\operatorname{arctg} x$
Exp(x)	e^x	Ln(x)	$\ln x$
Frac(x)	$\{x\}$ – дробная часть	Int(x)	$[x]$ – целая часть
Pi	π		

Функции преобразования типов

c:=Chr(i) – возвращает символ, код которого равен *i*;

i:=Ord(c) – возвращает номер значения перечислимого типа, например, код символа *c*;

Round(x) – округляет *x* до целого (тип результата – Longint);

Trunc(x) – усекает *x* до целого (тип результата – Longint).

Другие функции

Функция **SizeOf(x)** – определяет число байт памяти, занимаемое переменной x;

Функция **Odd(i)** – возвращает TRUE, если целое число *i* нечетное, FALSE – если четное.

Выражения

Выражение – это формальное правило для вычисления некоторого значения. Выражения строятся из операндов, знаков операций и круглых скобок.

Операндами могут быть константы, переменные (в том числе элементы массивов, поля записей, и т.п.), вызовы функций, и др.

Операции – действия по получению новых значений из значений операндов. Большинство операций являются *бинарными*, т.е. определенными для двух операндов. В этом случае знак операции ставится между ними. Примеры: **a + b**, **n div 2**, **x > y**. *Унарные* операции определены для одного операнда, здесь знак операции ставится слева. Примеры: **-x** (унарный минус), **not a**.

Очередность выполнения операций в выражении определяется их *приоритетами*. Первыми выполняются те операции, чей приоритет выше. Если приоритеты операций равны, то операции выполняются слева направо. Если этот порядок нужно изменить, используются круглые скобки, тогда часть выражения в скобках будет вычислена первой.

Операции, определенные в языке Паскаль, и их приоритеты приведены в таблице:

Операция		Приоритет	Категория
@	вычисление адреса переменной	Первый (наивысший)	
not	инверсия		
*	умножение	Второй	Операции типа умножения
/	деление		
div	деление нацело		
mod	остаток от деления нацело		
and	конъюнкция		
shl	логический сдвиг влево		
shr	логический сдвиг вправо		
+ –	сложение, вычитание, унарный плюс, унарный минус	Третий	Операции типа сложения
or	дизъюнкция		
xor	строгая дизъюнкция		
= <> < > <= >=	сравнения	Четвертый (низший)	Операции отношения
in	проверка принадлежности к множеству		

Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, «многоэтажных» дробей и т.д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры. Различают выражения *арифметические*, *логические* и *строковые*.

Арифметические выражения служат для определения одного числового значения. Например, **(1+sin(x))/2**.

Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения – «истина» или «ложь» (да или нет). Пример: логическое выражение **x*x + y*y < r*r**, определяющее принадлежность точки с координатами (x, y) внутренней области круга радиусом r с центром в начале координат. При $x=1$, $y=1$, $r=2$ значение этого выражения – "истина", а при $x=2$, $y=2$, $r=1$ – "ложь".

Строковые выражения – выражения, значениями которых являются наборы символов. В строковые выражения могут входить символьные и строковые константы, символьные и строковые переменные, строковые функции, разделенные знаками операции сцепки. Например, **A + B** означает присоединение строки **B** к концу строки **A**. Если **A='куст '**, а **B='зеленый'**, то значение выражения **A + B** есть **'куст зеленый'**.

Практические задания

1. Составить программу, вычисляющую и выводящую на экран значения приведенных ниже выражений. Значения переменных вводятся с клавиатуры.

$$S=\frac{\arcsin(x^2/(x^2+1))-\lg(1+\sqrt{y^4+1})}{2^{y-x^2}-\operatorname{ctg}(2xy)};\qquad P=\left\{|x|^y-\sqrt{|y|^x+1}\right\};$$

$$Q=\frac{\operatorname{tg}(x+y+\frac{\pi}{4})\cdot e^{\pi-xy}}{xy-\arccos(2^{-|y|})};\quad T=\left(|S|<|P| \text{ or } |P|<|Q|\right) \text{ and } x^2-y^2<1;$$

$m=(n \text{ shl } 2)+n$; $k=j$ div $(j \text{ shr } 1)$ (m, n, k, j – целые).

Указание. Использовать соотношения: $\lg x=\ln x/\ln 10$;

$$\operatorname{tg}x=\frac{\sin x}{\cos x};\ a^x=e^{x\ln a};\ \arcsin x=\operatorname{arctg}\frac{x}{\sqrt{1-x^2}};\ \arccos x=\operatorname{arctg}\frac{\sqrt{1-x^2}}{x}.$$

2. Составить программу, печатающую на экране код, который соответствует нажимаемой клавише.

```
Uses crt;
Var c: char;
    i: integer;
Begin
  Writeln('Нажмите клавишу: '); c:=ReadKey;
  Writeln('ASCII код символа ''',c,
          ''' равен: ',Ord(c));
  Readln
End.
```

Задания для самостоятельной работы

1. Составить программы, вычисляющие и выводящие на экран значения выражений:

а) $y = (2^{-x-1} + 3^{-x} + 4^{-x+1}) \cdot \sqrt[3]{|x+1|}$;

б) $y = \arccos\{x\} \cdot \log_2(x^2 + \pi)$ ({ } – дробная часть числа);

в) $y = 3^{\sqrt{x}} - \operatorname{tg}\dfrac{\sqrt{x^2+1}}{|1-3x|}$;

г) $z = \dfrac{1}{3}\left(\left(\dfrac{\sin^2 x - \cos^2 y}{\sin\dfrac{x+y}{2}} - e^{|\cos x|+\sin y}\right)\lg x - \sqrt{y^2+1}\right)$;

д) $T = ((|x| < 2)$ or $(\{x\} \geq \frac{1}{2}))$ and $((x \neq 1)$ or $(\operatorname{ctg} x < 1))$.

2. Составить программу, печатающую на экране символ, соответствующий введенному с клавиатуры коду (от 0 до 255).

3. Составить программу, которая для введенного положительного числа *a* вычисляет площади следующих фигур: равностороннего треугольника со стороной *a*, квадрата со стороной *a*, круга радиуса *a*.

4. Составить программу, которая для введенного целого трехзначного числа *N* находит и печатает сумму цифр числа и число, составленное из тех же цифр в обратном порядке.

5. Составить программу, которая для заданных координат вершин треугольника вычисляет его периметр и площадь.

6. Поменять местами значения двух целых переменных *x* и *y*, не используя дополнительные переменные.

7. Составить программу, которая для введенного номера дня в году (от 1 до 365) определяет день недели, при условии, что 1 января – среда.

Лабораторная работа №2
Программы разветвляющейся структуры

Для программирования разветвлений в программах на языке Turbo Pascal используются два оператора:

1) *Условный оператор* обеспечивает разветвление на две ветви:

```
IF <условие> THEN <оператор1>
ELSE <оператор2>
```

Вторая ветвь (т.е. **ELSE <оператор2>**) может отсутствовать.

Например:

```
IF x>y THEN z:=x ELSE z:=y;
IF a<b THEN a:=0;
```

Если требуется после слов **THEN** или **ELSE** использовать несколько операторов, то их необходимо объединить в один составной оператор с помощью операторных скобок **BEGIN** и **END**:

```
IF x>y THEN BEGIN a:=x; b:=y END
ELSE BEGIN a:=y; b:=x END
```

При использовании сложных логических выражений, включающих операции **AND**, **OR**, **NOT**, операции отношения необходимо заключать в скобки, например:

```
IF (a>b) AND ((x>y) OR (x>z)) THEN ...
```

Порядок выполнения операций в выражении определяется их приоритетами. Среди логических операций сначала выполняется операция **NOT** (наивысший приоритет), затем **AND**, последней – **OR** (низший приоритет). Для изменения этого порядка нужно использовать скобки.

2) *Оператор варианта* (оператор множественного выбора) обеспечивает разветвление на произвольное число ветвей:

```
CASE <выражение> OF
<список значений 1> : <оператор 1>;
<список значений 2> : <оператор 2>;
...
<список значений N> : <оператор N>;
ELSE <оператор>
END
```

Здесь также ветвь **ELSE** может отсутствовать.

Выражение после **CASE** должно быть дискретного типа, совместимого со значениями из списков.

Например:

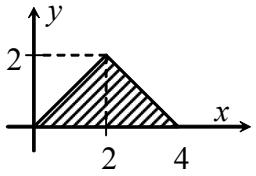
```
CASE ErrorCode OF
1:       writeln('Файл не найден');
2..4,6:  writeln('Ошибка открытия файла: ',
         ErrorCode);
5:       writeln('Недопустимое имя файла');
END
```

Практические задания

1. Изучить и запустить программу, определяющую, попало ли введенное пользователем число *x* в интервал [*a*, *b*].

```
Program Interval;
Var a,b,x: real;
Begin
  a:=-5; b:=5;
  Write('x='); Readln(x);
  If (x>=a) and (x<=b) then
    writeln('x принадлежит [a,b]')
  else writeln('x не принадлежит [a,b]');
  Readln
End.
```

2. Составить программу, определяющую, попала ли точка с координатами (*x, y*) внутрь заштрихованной области.



3. С клавиатуры вводятся три целых числа *a*, *b*, *c*. Изучить и запустить программу, которая выводит их на экран в порядке возрастания (использовать наименьшее количество условных операторов).

```
Program ThreeNumbers;
Var a,b,c,x: integer;
Begin
  Write('a='); Readln(a);
  Write('b='); Readln(b);
  Write('c='); Readln(c);
  {Упорядочим сначала два числа a, b – должно быть a < b}
  If a>b then begin x:=a; a:=b; b:=x end;
  {a и b упорядочены, теперь может быть c < a, c > b или a < c < b}
  If c<a then Writeln(c,' ',a,' ',b)
  else If c>b then Writeln(a,' ',b,' ',c)
       else Writeln(a,' ',c,' ',b);
  Readln
End.
```

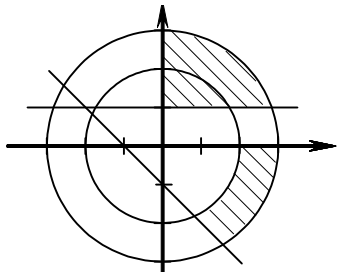
4. Изучить и запустить программу, которая вычисляет сумму, разность, произведение или частное двух чисел в зависимости от знака операции +, –, *, / (т.е. простейший калькулятор). Добавить в программу операцию возведения *x* в степень *y*.

```
Program Calc;
Var x,y,r:real;
c:char;
Begin
  write('Операнд 1: '); readln(x);
  write('Операция : '); readln(c);
  write('Операнд 2: '); readln(y);
  Case c of
  '+': r:=x+y;
  '-': r:=x-y;
  '*': r:=x*y;
  '/': If y<>0 then r:=x/y
       else
       begin writeln('Деление на 0');exit end;
  else
  begin writeln('Неверная операция');exit end;
  end;
  writeln(x:6:3,' ',c,' ',y:6:3,' = ', r:9:6);
  readln
End.
```

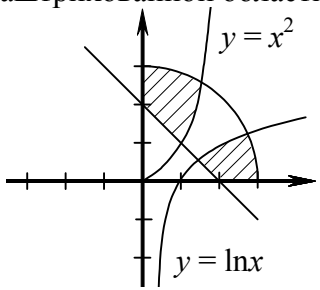
Задания для самостоятельной работы

1. Составить программу, определяющую, попала ли точка с координатами (*x, y*) внутрь заштрихованной области.

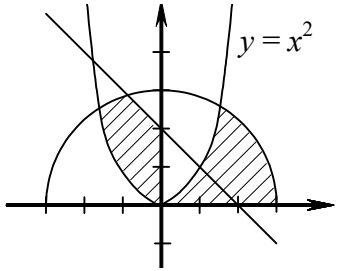
а)



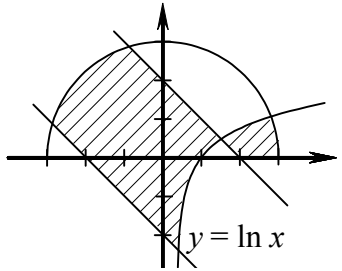
б)



в)



г)



2. Составить программу, которая преобразует введенное с клавиатуры целое число от 0 до 15 в шестнадцатеричную систему и выводит результат на экран.

3. Используя оператор варианта, составить программу вычисления площадей геометрических фигур – круга, квадрата, прямоугольника, прямоугольного треугольника, произвольного треугольника.

4. Составить программу, определяющую, можно ли построить треугольник со сторонами *a*, *b*, *c*, а также его форму – равносторонний, равнобедренный или разносторонний; прямоугольный, остроугольный или тупоугольный.

5. Составить программу вычисления действительных корней квадратного уравнения $ax^2 + bx + c = 0$.

6. На клавиатуре вслепую нажимается клавиша. Составить программу, определяющую тип введенного символа – цифра, прописная буква, строчная буква, спецсимвол (использовать оператор варианта).

Лабораторная работа №3
Программы циклической структуры

В Турбо Паскале имеется три оператора цикла:

1) Оператор цикла с параметром For … do. Оператор безусловного цикла имеет следующую структуру:

```
For i:=N1 to N2 do < тело цикла >;
```

где **i** – параметр цикла, который должен быть дискретного типа (целого, символьного и т.д.); **N1** – начальное значение параметра цикла; **N2** – конечное значение параметра цикла.

Безусловный цикл выполняется заданное число раз. Чтобы прервать выполнение досрочно, необходимо увеличить i до конечного значения (**i:=N2**), либо использовать процедуру **Break**, которая прерывает выполнение циклов. Процедура **Continue** начинает новую итерацию цикла, даже если предыдущая не была завершена.

При программировании может быть использован другой вид оператора **For**, в котором происходит уменьшение значения переменной:

```
For i:=N2 downto N1 do < тело цикла >;
```

2) Оператор цикла с постусловием Repeat … Until. В этом операторе проверка условия выхода из цикла осуществляется после каждого выполнения тела цикла, поэтому тело цикла обязательно выполнится хотя бы один раз:

```
Repeat
< тело цикла >
Until < условие >;
```

Цикл выполняется до тех пор, пока условие не станет истинным. Например, при задании длины массива проверка может осуществляться следующим образом:

```
Repeat
  Write('Введите длину массива N (N<10): ');
  ReadLn(N)
Until (N>0) AND (N<10);
```

Цикл выполняется до тех пор, пока пользователь не введет положительное число, меньшее 10.

Здесь в теле цикла можно использовать несколько операторов, не заключая их в операторные скобки **begin** … **end**.

3) **Оператор цикла с предусловием While … do.** В этом цикле проверка условия проводится до начала выполнения тела цикла:

```
While < условие > do < тело цикла >;
```

Цикл выполняется, пока условие истинно. Как только условие становится ложным, выполнение цикла завершается. Если условие ложно с самого начала, тело цикла не выполнится ни разу. Например, та же проверка вводимой длины массива может быть задана так:

```
N:=0;
While (N<=0) OR (N>=10) do
begin
  Write('Введите длину массива N: ');
  ReadLn(N)
END;
```

Обратите внимание, что до первого входа в цикл переменной **N** уже должно быть присвоено какое-либо значение!

Процедуры **Break** и **Continue** могут использоваться в условных циклах так же, как и в цикле **For**.

Практические задания

1. Изучить и запустить программу, вычисляющую сумму $\sum_{i=1}^{n}\frac{1}{i!}$, $n=10$. Определить максимальное значение n, при котором факториал (переменная **f** типа **longint**) вычисляется верно.

```
var f: longint;
    i: integer;
    s: real;
Begin
  s:=0; f:=1;
  for i:=1 to 10 do begin
    f:=f*i;
    s:=s+1/f
  end;
  writeln('s=',s:9:6); readln
End.
```


2. Составить программы, вычисляющие значения выражений:

а) $\sum_{i=1}^{100}\frac{x}{i^2}$; б) $\sum_{i=1}^{N}\frac{1}{\sqrt[3]{i^2}}$; в) $\sum_{i=1}^{10}\frac{x^{i+1}}{(2i)!}$; г) $\sum_{k=1}^{N}\left(1+\frac{\sin(kx)}{k!}\right)$;

д) $\prod_{k=1}^{N}\frac{\log_2(kx)}{k}$; е) $\prod_{i=1}^{5}\frac{i+2}{i^2+1}$; ж) $\sum_{i=1}^{10}\sum_{j=i}^{10}\frac{1}{i+j^2}$; з) $\sum_{i=1}^{100}\sum_{j=1}^{i}\frac{1}{2j+i}$;

и) $\sin x+\sin^2 x+...+\sin^n x$; к) $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+...+\frac{1}{\sin 1+...+\sin n}$.

3. Составить программу, вычисляющую значение выражения $f=n\cdot 10!+(n-1)\cdot 9!+...+(n-10)\cdot 0!$, где n – любое целое число.

```
var n,u,f: longint;
    i, j: integer;
begin
  writeln('Введите число n');
  readln(n);
  f:=0;
  for i:=10 downto 0 do { цикл вычисления суммы}
  begin
    u:=1;
    for j:=2 to i do u:=u*j; {цикл вычисления факториала}
    f:=f+(n+i-10)*u; {добавление в сумму}
  end;
  writeln('f=', f:12);
end.
```

4. Вычислить с помощью рекуррентной формулы сумму ряда с заданной точностью ε (использовать оператор цикла с постусловием):

$$\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!}.$$

Решение. Каждый член данного ряда требует вычисления только двух функций – степенной и факториала. Поэтому в рекуррентную формулу целесообразно включить весь член ряда

$$u_n = (-1)^n \frac{x^{2n}}{(2n)!}.$$

Итак, требуется получить формулу

$$u_{n+1} = R_n u_n .$$

Определим множитель R_n

$$R_n = \frac{u_{n+1}}{u_n} = \frac{(-1)^{n+1} x^{2(n+1)}}{(2(n+1))!} \frac{(2n)!}{(-1)^n x^{2n}} = -\frac{x^2}{(2n+1)(2n+2)}.$$

Таким образом, сумму указанного ряда можно найти по следующей схеме:

$$S = u_0 + u_1 + ... + u_n ,$$

где

$$u_0 = 1, \quad u_{n+1} = -\frac{x^2}{(2n+1)(2n+2)} u_n ,$$

суммирование заканчивается, когда

$$|u_{n-1}| > \varepsilon, \quad |u_n| < \varepsilon.$$

```
var s,u,x,eps: real;{s – сумма, u – член ряда, eps - точность}
    n: integer;  {x – аргумент функции, n – номер члена ряда}
begin
  writeln('Введите аргумент x и точность eps');
  readln(x, eps);
  u:=1;  s:=u;  n:=0;  { инициализация переменных}
  repeat
    u:=-u*x*x/(2*n+1)/(2*n+2); {рекуррентная формула}
    s:=s+u; {суммирование ряда}
    n:=n+1;  {счетчик}
  until abs(u)<eps;
  writeln('Сумма ряда равна ', s:10:6);
  writeln('Для сравнения cos(x)=', cos(x):10:6);
  readln
end.
```


5. Вычислить суммы рядов с точностью ε = 0,0001. Использовать, где это целесообразно, рекуррентные формулы. Использовать цикл с предусловием (а, в), с постусловием (б, г):

а) $\sum_{n=0}^{\infty} \frac{1}{(2n+1)^2} = \frac{\pi^2}{8}$; б) $\sum_{n=1}^{\infty} \frac{1}{n^4} = \frac{\pi^4}{90}$;

в) $S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{2^n}{n!}$; г) $S = \sum_{n=1}^{\infty} \frac{(-1)^n 2^{-n+1}}{n(n+1)(n+2)}$;

6. Изучить и запустить программу вычисления значений функции на интервале $x_1 \le x \le x_2$ с шагом h и найти ее минимальное значение и среднее арифметическое:

$$y = \begin{cases} \sin^2 |x|, & \text{если } x > a; \\ \cos x^2, & \text{если } x \le a. \end{cases}$$

```
var x,y,a, x1, x2, h, xmin, ymin, SA: real;
    n,i: integer;
begin
writeln('Введите число a, начало и конец интервала и шаг h');
readln(a, x1, x2, h);
if (x2>x1) and ((x2-x1)>2*h) then { если x2 > x1 и}
begin                    { на интервале не менее трех точек}
  x:=x1;                 { инициализация }
  n:=round((x2-x1)/h); {кол-во точек равно n+1 }
  if(x>a) then y:= sqr(sin(abs(x)))
  else y:= cos(x*x);
  writeln('x=', x:7:3,'  y=', y:9:4); {вывод x1 и y1}
  xmin:=x; ymin:=y; SA:=y; { присваивание начальных }
                           { значений перед циклом }
  for i:=1 to n do begin   { выполняется цикл }
    x:=x1+i*h;
    if(x>a) then y:= sqr(sin(abs(x)))
    else y:=cos(x*x);      {вычисление функции }
    writeln('x=', x:7:3,'  y=', y:9:4); {вывод x и y}
    if y<ymin then begin
      ymin:= y; xmin:= x;  {корректируем ymin, xmin}
    end;
```

```
    SA:=SA+y;          {накопление суммы значений функции}
  end;                 { конец цикла }
  writeln('Минимальное значение y =',ymin:9:4,'
при x=', xmin:7:3);
  writeln('Среднее значений функции ',
SA/(n+1):9:4);
end
else writeln ('Неверно заданы x1 и x2');
readln
end.
```

Задания для самостоятельной работы

1. Составить программу, выводящую на экран все символы кодовой таблицы ASCII.

2. Составить программу, которая вводит с клавиатуры четное целое число X в интервале от 0 до 100 с проверкой введенного значения и предложением повторить ввод в случае ошибки.

3. Составить программу, находящую сумму всех двузначных чисел: а) которые делятся на 7 и при этом не делятся на 5; б) которые делятся на 3 и не делятся на 4; в) у которых число единиц делится нацело на число десятков.

4. Составить программу, выводящую на экран все простые числа в интервале от 1 до 100 (простым называется число, которое делится нацело только на 1 и на себя).

5. Вычислить суммы рядов с точностью ε = 0,0001. Использовать, где это целесообразно, рекуррентные формулы.

а) $\sum_{n=2}^{\infty}\frac{1}{\sqrt{n}}\ln\frac{n+1}{n-1}$; б) $\sum_{n=2}^{\infty}\left(\frac{n-1}{n+1}\right)^{n^2}$; в) $\sum_{n=1}^{\infty}\frac{1}{n!(x+3)^n}$; г) $\sum_{n=1}^{\infty}\frac{(-1)^{n-1}\ln(n+1)}{n(x+1)^n}$.

6. Составить программу вычисления значений функций на интервале $x_1 \le x \le x_2$ с шагом h и найти минимальное, максимальное и среднее арифметическое этих значений:

1) $y=\dfrac{2^{-x}+2^{-2x}}{2^{x}+2^{2x}}$; 2) $y=\begin{cases}10^{x}, & x<0;\\ \cos(\pi x/2), & x\ge 0.\end{cases}$

Лабораторная работа №4
Работа с массивами и строками

1) Тип «массив»

Массив – последовательность однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним, в случае одномерного массива, или несколькими, если массив многомерный). Иногда многомерные массивы называют таблицами или матрицами. Описание:

```
Var <имя>: Array[<тип1>, <тип2>, …, <типN>]
of <базовый тип>;
```

Здесь **<тип1>, <тип2>, …, <типN>** – типы индексов, их число определяет размерность массива (один – для одномерного массива, два – для двумерного, и т.д.). Чаще всего здесь используется тип «диапазон». Базовый тип – тип элементов массива. Примеры:

```
Var A: Array[1..3] of real;  {массив из трех вещественных чисел}
    B: Array[1..10, 1..10] of integer;  {матрица 10×10 из целых чисел}
    C: Array[Char] of integer;  {массив из 256 целых чисел, значения индекса – символы}
```

Также массив может быть определен с помощью описания нового типа:

```
Type Vector = Array[1..3] of real;
     Matrix = Array[1..10, 1..10] of integer;
Var A: Vector;
    B: Matrix;
```

Этот способ применяется, если массив нужно передать в подпрограмму в качестве параметра.

Единственная операция над массивом в целом – присваивание, например, `A:=B;` (при этом массивы **A** и **B** должны быть определены одинаково). Все остальные операции над массивами производятся поэлементно, чаще всего в циклах.

Обращение к элементам массивов производится путем указания в квадратных скобках значений индексов: **A[1], B[2,3], B[i,j],** и т.п. Допустима другая форма: **B[2][3]**.

2) Тип «строка»

Строковый тип расширяет понятие символьного массива (**Array[…] of char**), предоставляя новые возможности. Описание:

```
Var <имя>: String[<макс. длина>];
```

Если максимальная длина не указана явно, то по умолчанию берется значение 255. Примеры:

```
Var S: String[80];
    mess: string;
```

При создании в программе строки с максимальной длиной N символов под нее отводится $N+1$ байт памяти. Элементы с номерами от 1 до N предназначены для хранения символов, а элемент с номером 0 – для хранения динамической (т.е. текущей) длины строки. Определить текущую длину строки **S** можно с помощью функций **Length(S)** или **Ord(S[0])**.

Строковые константы, как и символьные, заключаются в апострофы:

```
S:='Пример строки';
```

Если при присваивании максимальная длина строковой переменной оказывается меньше длины строковой константы, то лишние символы отбрасываются.

Операции над строками:

а) конкатенация (т.е. слияние строк): две строки соединяются в одну:

```
S:='Пример ' + 'строки';
```

б) сравнение по правилам:

– более короткая строка меньше, чем более длинная;

– если длины строк равны, то попарно сравниваются коды первых символов, если они равны, то вторых, и т.д.

Доступ к отдельным символам строки производится так же, как и к элементам массивов: **S[3]:='a'**;

Турбо Паскаль содержит большое число стандартных функций для работы со строками, познакомиться с которыми можно в справочной системе среды программирования.

Практические задания

1. Изучить и запустить программу, осуществляющую ввод массива с клавиатуры, сортировку его по убыванию и вывод на экран. С ее помощью составить программу, выполняющую следующие действия над вещественным одномерным массивом $A(n)$:

а) поиск и вывод на экран минимального и максимального элементов с указанием номера;

б) перестановку элементов массива (1-го с последним, 2-го с предпоследним, и т.д.);

в) сортировку элементов массива по возрастанию методом пузырька;

г) замену каждого из элементов массива a_{ij} на разность $a_{ij} - a_{cp}$, где a_{cp} – среднее арифметическое элементов массива.

```
var A: array[1..100] of real;
    i,j,n: integer;
    z: real;

begin
  {Ввод массива A(n)}
  write('Введите длину массива (n<100):');
  readln(n);
  writeln('Введите элементы массива:');
  for i:=1 to n do
  begin
    write('A[', i, ']=');
    readln(A[i])
  end;

  {Вывод массива в одну строку}
  writeln('Массив A:');
  for i:=1 to n do write(A[i]:6:3, ' ');
  writeln;
```

```
  {Сортировка}
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if A[i]<A[j] then
      begin
        z:=A[i]; A[i]:=A[j]; A[j]:=z
      end;
  {Вывод отсортированного массива}
  writeln('Отсортированный массив:');
  for i:=1 to n do write(A[i]:6:3, ' ');
  writeln;
  readln
end.
```

2. Изучить и запустить программу, осуществляющую ввод матрицы A с клавиатуры, перестановку указанных строк и вывод матрицы на экран в виде таблицы. С ее помощью составить программу, выполняющую следующие действия над вещественным двумерным массивом $A(m\times n)$, $m \neq n$:

а) поиск и вывод на экран минимального и максимального элементов матрицы с указанием их позиции;

б) подсчет количества положительных и отрицательных элементов матрицы A;

в) транспонирование матрицы A;

г) вычисление и печать результатов перемножения матриц A и A^{T}: $B = AA^{\mathrm{T}}$, $C = A^{\mathrm{T}}A$.

д) перестановку столбцов матрицы A, чтобы максимальный элемент оказался в первом столбце, и перестановку строк, чтобы он оказался в левом верхнем углу матрицы.

```
var A: array [1..10,1..10] of real;
    i,j,n,m,m1,m2: integer;
    z: real;
begin
  {Ввод матрицы A}
  write('Введите число строк (m<10):');
  readln(m);
```



```
  write('Введите число столбцов (n<10):');
  readln(n);
  for i:=1 to m do
    for j:=1 to n do
    begin
      write('A[', i, ',', j, ']=');
      readln[a[i,j]]
    end;
  {Вывод матрицы в виде таблицы}
  writeln('Исходная матрица:');
  for i:=1 to m do
  begin
    for j:=1 to n do write(a[i,j], '  ');
    writeln
  end;
  {Ввод номеров переставляемых строк}
  write('Введите номера переставляемых строк: ');
  readln(m1,m2);
  if (m1>m) or (m2>m) or (m1=m2) then
  begin
    writeln('Неверные номера строк');
    exit
  end;
  {Перестановка}
  for j:=1 to n do
  begin
    z:=a[m1,j]; a[m1,j]:=a[m2,j]; a[m2,j]:=z
  end;
  {Вывод матрицы в виде таблицы}
  writeln('Матрица после перестановка:');
  for i:=1 to m do
  begin
    for j:=1 to n do write(a[i,j], '  ');
    writeln
  end;
  readln
end.
```

3. Изучить и запустить программу, подсчитывающую, сколько раз встречается каждый символ в задаваемой с клавиатуры строке.

```
Var s: string;
    i: integer;
    Num: array [char] of byte;
    c: char;
begin
 writeln('Введите строку:'); readln(s);
 for c:=chr(0) to chr(255) do Num[c]:=0;
 for i:=1 to length(s) do
   Num[s[i]]:=Num[s[i]]+1;
 for c:=chr(0) to chr(255) do
  if Num[c]>0 then
   writeln('Символов ''',c,''' - ',Num[c],' шт');
 readln
end.
```

Задания для самостоятельной работы

1. Составить программу, которая в массиве $A(15)$ подсчитывает число отрицательных и число неотрицательных элементов и формирует из них два массива, в один из которых войдут неотрицательные, а в другой – отрицательные элементы.

2. Составить программу, осуществляющую ввод с клавиатуры вещественных матриц $A(3\times3)$ и $B(3\times3)$ и выполняющую следующие действия:

а) сложение матриц: $C=A+B$;

б) перемножение матриц: $C=AB$;

в) вычисление скалярного произведения указанной строки матрицы A и того же столбца матрицы B.

3. Составить программу, которая во введенной с клавиатуры строке удаляет лишние пробелы между словами, оставляя один.

4. Составить программу, которая во введенной с клавиатуры строке упорядочивает слова по возрастанию длины.

Лабораторная работа №5
Процедуры и функции

В Паскале существует два типа подпрограмм – процедуры и функции. Описание всех процедур и функций, созданных пользователем, должно присутствовать в разделе подпрограмм. Для использования большинства стандартных подпрограмм необходимо в разделе **Uses** подключить к программе модуль, содержащий эту подпрограмму.

1) Процедуры

Структура подпрограммы, в том числе процедуры, в целом идентична структуре основной программы – заголовок, разделы описаний и раздел операторов, только в отличие от основной программы заголовок здесь не может отсутствовать:

```
Procedure <имя>(<список формальных параметров>);
<разделы описаний>
begin
  <операторы>
end
```

Список формальных параметров представляет собой перечисление имен параметров с указанием их типов. Например, процедура поиска и вывода максимума из двух вещественных чисел может выглядеть так:

```
Procedure Print_max(x,y: real);
begin
  if x>y then writeln(x:9:6)
  else writeln(y:9:6)
end
```

Вызов процедуры может производиться из любого места основной программы или других подпрограмм с помощью оператора вызова процедуры. Указывается имя процедуры и список фактических параметров, которые будут подставлены вместо формальных:

```
Print_max(a, b);
```

или

```
Print_max(a+b, 2*c);
```

Между формальными и фактическими параметрами должно быть строгое соответствие по количеству, типу и порядку следования.

Чтобы передать в подпрограмму в качестве параметра массив, строку или другую переменную составного типа, этот тип данных должен быть описан в разделе **Type** с присвоением нового имени. Например, процедура печати первых *N* элементов массива:

```
Type mass = array[1..100] of real;
Var a: mass;
    n: integer;
Procedure print_mass(z: mass; n: integer);
var i: integer;
begin
  for i:=1 to n do write(z[i]:6:2,'  ');
  writeln
end
```

2) Функции

Описание функции похоже на описание процедуры – заголовок, разделы описаний, раздел операторов. Отличие заключается в том, что функция имеет результат, тип которого указывается после круглых скобок (его имя совпадает с именем функции):

```
Function <имя>(<форм. параметры>): <тип ре-
                                   зультата>;
<разделы описаний>
begin
  <операторы>
end
```


Чтобы результат функции был вычислен, внутри нее должен быть хотя бы один оператор присваивания, в котором имя функции стоит слева. Например, функция вычисления длины вектора имеет вид:

```
Type vector = array[1..3] of real;
Function Len(z: vector): real;
begin
  Len:=sqrt(z[1]*z[1]+ z[2]*z[2]+z[2]*z[2])
end;
```

Вызов функции, как стандартной, так и определенной самим программистом, обязательно должен входить в какое-либо выражение в качестве операнда:

```
L:=Len(a);
x:=(1+sin(t))/2;
writeln('sin y = ', sin(y));
```

Обмен данными между основной программой и подпрограммой производится с помощью параметров. Параметры могут быть следующих типов: *параметры-значения*, *параметры-переменные*, *нетипизированные параметры* и *параметры-константы*.

С помощью *параметров-значений* можно передавать данные только в одном направлении – из основной программы в подпрограмму, т.е. через них передаются исходные данные для подпрограммы. При этом в качестве фактических параметров, стоящих в скобках при вызове подпрограммы, могут использоваться любые выражения, тип результата которых совместим с типом соответствующего формального параметра. Эти выражения могут включать переменные, константы, вызовы функций и др.

Если в подпрограмме параметры-значения изменятся в процессе вычислений, то эти изменения никак не отразятся ни на одной переменной той программы, откуда была вызвана подпрограмма. Поэтому для возвращения результатов из под-

программы в основную программу параметры-значения не подходят. Такой способ передачи параметров называется передачей *по значению*, поскольку формальному параметру присваивается *значение* фактического параметра. В этом случае формальный параметр будет содержать копию значения, имеющегося в фактическом, и никакое воздействие, производимое внутри подпрограммы на формальные параметры, не отражается на параметрах фактических.

Параметры-переменные позволяют передавать данные в обоих направлениях – и из основной программы в подпрограмму, и обратно. Поэтому с их помощью можно возвращать в основную программу результаты, полученные в подпрограмме. Это возможно потому, что в подпрограмму передается не значение фактического параметра, а ссылка на него, т.е. его адрес в памяти. Поэтому все изменения формального параметра в подпрограмме – это на самом деле изменения фактического параметра, так как в этом случае оба параметра суть одно и то же. Такой способ передачи параметров называется передачей *по ссылке*. В этом случае в качестве фактических параметров можно использовать только переменные (в том числе элементы массивов, поля записей и т.п.), а выражения и константы использовать нельзя. При описании подпрограммы в списка формальных параметров перед параметрами-переменными ставится слово **Var**. Например, заголовок процедуры, возвращающей минимальное и максимальное значения из элементов массива длиной *N*:

```
Type mass = array[1..100] of real;
Var a: mass;
Procedure MinMax(z: mass; n: integer; var min,
max: real);
```

В случае *параметров-констант* перед их именами ставится слово **Const**. При их использовании в подпрограмму также передается ссылка, т.е. адрес параметра в памяти, однако компилятор блокирует все попытки изменить его в подпрограмме, т.е. присвоить ему какое-либо значение.

Если формальный параметр является *нетипизированным* параметром-переменной, то соответствующий ему фактический параметр может представлять собой любую ссылку на переменную, независимо от ее типа. Описание таких параметров выглядит как группа параметров с ключевым словом **Var** впереди, за которыми не следует указание типа, например:

```
Function Fun(var First, Second): Boolean;
```

Если некая переменная объявлена в разделе **Var** основной программы, то ее называют *глобальной* переменной. Она существует от места описания до конца программы (эту область называют областью видимости данной переменной). Таким образом, она видна и может быть использована и во всех подпрограммах этой программы.

Если некая переменная объявлена в разделе **Var** какой-либо подпрограммы, то она считается *локальной* переменной этой подпрограммы. Она существует от места описания до конца этой подпрограммы, т.е. только в этой подпрограмме.

Если некоторое имя определено в скобках в заголовке подпрограммы, то это формальный параметр этой подпрограммы.

Встретив в тексте подпрограммы какой-либо идентификатор, компилятор начинает искать его описание сначала в разделе **Var** этой подпрограммы, затем в списке формальных параметров, и если не находит, то продолжает поиск во внешнем по отношению к этой подпрограмме блоке, пока не доберется до раздела **Var** основной программы. Таким образом, если имя некоторой глобальной переменной совпадет с именем локальной переменной подпрограммы, то в этой подпрограмме можно будет обращаться только к локальной переменной, а глобальная будет недоступна. Говорят, что в этих случаях локальное имя закрывает (маскирует) глобальное.

Считается, что в подпрограммах использовать глобальные переменные следует с осторожностью, так как это может привести к ошибкам.

Практические задания

1. Изучить и запустить программу вычисления площади треугольника, заданного тремя сторонами *a*, *b*, *c*, по формуле Герона, в которой проверка существования треугольника и вычисление площади реализованы в виде функций.

```
Program Geron;
var a,b,c,s: real;

function exist(x,y,z: real): boolean;
begin
  if (x<=0) or (y<=0) or (z<=0) or
     (x+y<z) or (x+z<y) or (y+z<x) then
    exist:=false
  else exist:=true;
end;

function square(x,y,z: real): real;
var p: real;
begin
  p:=(x+y+z)/2;
  square:=sqrt(p*(p-x)*(p-y)*(p-z))
end;

Begin
  writeln('Введите длины сторон a,b,c:');
  readln(a,b,c);
  if exist(a,b,c) then
  begin
    s:=square(a,b,c);
    writeln('Площадь S=',s:9:6)
  end
  else writeln('Треугольник не существует');
  readln
End.
```


2. Изучить и запустить программу вычисления длины вектора, заданного в виде массива из 3 чисел, в которой вычисление длины вектора реализовано в виде функции. С ее помощью составить программу обработки векторов, заданных в виде массива из 3 чисел, выполняющую следующие действия: а) вычисление скалярного произведения двух векторов; б) вычисление угла между двумя векторами; в) вычисление векторного произведения двух векторов.

Указание. Функцию $\arccos(t)$ вычислять как $\operatorname{arctg}\dfrac{\sqrt{1-t^2}}{t}$.

```
Program Vectors;
type vect=array[1..3] of real;
var a,b: vect;
    z: real;

function len(x:vect):real;
begin
  len:=sqrt(x[1]*x[1]+x[2]*x[2]+x[3]*x[3])
end;

Begin
  writeln('Введите компоненты вектора');
  readln(a[1],a[2],a[3]);
  z:=len(a);
  writeln('Длина вектора a: ',z:6:2);
  readln
End.
```

3. Изучить и запустить программу, осуществляющую ввод квадратных матриц A и B и вывод их суммы $C=A+B$ (ввод, вывод и сложение матриц реализованы в виде процедур). С ее помощью составить программу обработки матриц, выполняющую следующие действия (реализовать в виде процедур):
а) транспонирование матрицы A;
б) вычисление произведения двух матриц $C=AB$;
в) вычисление следа матрицы (т.е. суммы элементов главной диагонали);

г) поиск максимального и минимального из элементов матрицы и их позиции;
д) поменять местами элементы матрицы *A* симметричным образом относительно главной (или побочной) диагонали.

```
Program Matrix;
type matr=array[1..10,1..10] of real;
var a,b,c: matr;
    n: integer;

procedure inp(var x:matr; n:integer; t:char);
var i,j:integer;
begin
  for i:=1 to n do
    for j:=1 to n do
    begin
      write(t,'[', i, ',', j, ']=');
      readln(x[i,j])
    end
end;

procedure outp(x:matr; n:integer);
var i,j:integer;
begin
  for i:=1 to n do
  begin
    for j:=1 to n do write('  ',x[i,j]:6:2);
    writeln
  end
end;

procedure sum(x,y:matr; n:integer; var s:matr);
var i,j:integer;
begin
  for i:=1 to n do
    for j:=1 to n do s[i,j]:=x[i,j]+y[i,j]
end;
```



```
Begin
  writeln('Введите размерность матриц n: ');
  readln(n);
  writeln('Введите матрицу A');
  inp(a,n,'A');
  writeln('Матрица A:');
  outp(a,n);
  writeln('Введите матрицу B:');
  inp(b,n,'B');
  writeln('Матрица B: ');
  outp(b,n);
  sum(a,b,n,c);
  writeln('Матрица C=A+B:');
  outp(c,n);
  readln
End.
```

Задания для самостоятельной работы

1. Используя функцию, составить программу решения квадратного уравнения $ax^2+bx+c=0$.

2. Составить программу решения системы трех линейных уравнений с тремя неизвестными методом Крамера. Вычисление определителя 3-го порядка оформить в виде функции.

3. Треугольник задан координатами вершин на плоскости. Используя функции, составить программу вычисления длин сторон треугольника, углов между сторонами и определения самой длинной из сторон.

4. Используя процедуры, составить программу обработки одномерного целочисленного массива, которая: а) находит количество четных и нечетных элементов; б) формирует из одного исходного массива два, из четных и нечетных элементов соответственно; в) удаляет из массива идущие подряд одинаковые элементы, оставляя один.

5. Используя процедуры, составить программу обработки строк, которая: а) печатает самое длинное слово в строке; б) находит и печатает все слова заданной длины; в) заменяет в заданной строке все прописные буквы на строчные.

Лабораторная работа №6
Типизированные и текстовые файлы

Под файлом понимается либо именованная область внешней памяти ПК (жесткого диска, гибкой дискеты, и др.), либо логическое устройство – потенциальный источник или приемник информации.

В программах на Турбо Паскале могут использоваться три вида файлов: *типизированные*, *текстовые* и *нетипизированные*. Примеры описания переменных файловых типов:

```
Var <имя>: File of <тип>;   {для типизированных файлов}
    <имя>: Text;            {для текстовых файлов}
    <имя>: File;            {для нетипизированных файлов}
```

Например:

```
Var f: File of real;
    g: File of integer;
    h: Text;
```

Можно ввести новый тип данных:

```
Type text80 = File of string[80];
Var t: text80;
```

Любой программе доступны два предварительно объявленных файла со стандартными файловыми переменными: **Input** (для чтения данных с клавиатуры) и **Output** (для вывода на экран). Стандартный Паскаль требует обязательного упоминания этих файлов в заголовке программы, например, так:

```
Program NameOfProgram(input, output);
```

В Турбо Паскале это необязательно, вот почему заголовок программы можно опускать.

Любые другие файлы, а также логические устройства становятся доступны программе только после выполнения особой процедуры *открытия файла* (логического устройства). Эта процедура заключается в связывании ранее объявленной файловой переменной с именем существующего или вновь создаваемого файла, а также в указании направления обмена информацией: чтение из файла или запись в него.

Файловая переменная связывается с именем файла в результате обращения к стандартной процедуре **Assign**:

```
Assign(<файловая переменная>, <имя файла или
                               логическое устройство>);
```

Например:

```
Assign(f, 'file1.dat');
```

Если имя файла задается в виде пустой строки, например, **Assign (f, '')**, то в зависимости от направления обмена данными файловая переменная связывается со стандартным файлом **Input** или **Output**. Перед именем может указываться так называемый путь к файлу: имя диска и/или имя текущего каталога и имена каталогов вышестоящих уровней. Например:

```
Assign(f, 'c:\dir\subdir\file1.dat');
```

Все прочие действия с файлами также реализованы в виде стандартных процедур и функций.

1) Типизированные файлы

Главной особенностью типизированных файлов является возможность доступа к любому элементу файла по его номеру, без необходимости чтения предыдущих элементов, поскольку все эти элементы имеют одинаковый размер в байтах.

Типизированный файл можно открыть (инициировать) одной из двух стандартных процедур: **Reset** или **Rewrite**. Параметр – файловая переменная:

```
Reset(f);
Rewrite(g);
```

Файловая переменная здесь должна быть предварительно связана процедурой **Assign** с уже существующим файлом или логическим устройством – приемником информации.

Использование **Reset** предполагает, что открываемый файл уже существует, иначе возникает ошибка. При использовании **Rewrite**, напротив, открываемый файл может не существовать, в этом случае он создается. Если же он уже существовал, то он очищается – т.е. все записанные в нем данные уничтожаются.

При выполнении этих процедур дисковый файл или логическое устройство подготавливается к чтению или записи данных. В результате специальная переменная-указатель, связанная с этим файлом, будет указывать на начало файла, т.е. на компонент с порядковым номером 0.

Чтение из ранее открытого типизированного файла осуществляет процедура **Read**. Она обеспечивает считывание очередных компонентов типизированного файла, начиная с того, на который указывает указатель файла. Формат обращения:

Read(<*файловая переменная*>, <*список ввода*>)

Список ввода может содержать одну или более переменных такого же типа, что и компоненты файла. Файл необходимо открыть процедурой **Reset**. Если файл исчерпан, обращение к **Read** вызовет ошибку ввода-вывода. Процедуру **ReadLn** для типизированных файлов использовать нельзя.

Запись в ранее открытый типизированный файл выполняется с помощью процедуры **Write**:

Write(<*файловая переменная*>, <*список вывода*>)

Список вывода может содержать одно или более выражений того же типа, что и компоненты файла. Файл может быть открыт как процедурой **Rewrite**, так и **Reset**. Процедурой **WriteLn** для типизированных файлов пользоваться нельзя.

Также для типизированных файлов могут использоваться:

– Процедура **Seek**. Смещает указатель файла к требуемому компоненту файла. Формат обращения:

Seek(<*файловая переменная*>, <*номер компонента*>)

Здесь *номер компонента* – выражение типа **Longint**. Первый компонент файла имеет номер 0, т.е. вызов **Seek(f, 0)** перемещает указатель файла в начало файла. Процедуру нельзя применять к текстовым файлам.

– Функция **FileSize**. Возвращает значение типа **Longint**, которое содержит количество компонентов файла. Формат обращения:

FileSize(<*файловая переменная*>)

Функцию нельзя использовать для текстовых файлов. Чтобы переместить указатель в конец типизированного файла, можно написать:

```
Seek(f, FileSize(f));  { f – файловая переменная }
```

– Функция **FilePos**. Возвращает значение типа **Longint**, содержащее порядковый номер компонента файла, на который указывает указатель файла. Формат обращения:

FilePos(<*файловая переменная*>)

Для текстовых файлов **FilePos** использовать нельзя.

– Функция **EOF**. Логическая функция, проверяющая, достигнут ли конец файла (EOF – End Of File). Возвращает True, если достигнут конец файла, т.е. файловый указатель стоит в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении – что файл исчерпан. Может использоваться для файлов любого типа. Например, считывание элементов массива x из файла, если их число заранее неизвестно, можно оформить так:

```
i:=1;
While not EOF(f) do
begin
  read(f, x[i]); i:=i+1
end;
```

– Процедура **Close**. Чтобы внесенные в файл данные сохранились на диске, файл требуется закрыть процедурой **Close**. Она закрывает файл, однако связь файловой переменной с именем файла, установленная ранее процедурой **Assign**, сохраняется. Формат обращения:

Close(<*файловая переменная*>)

Если в конце работы программы открытые файлы не закрыть процедурой **Close**, то они будут автоматически закрыты операционной системой, но при этом все внесенные в них изменения будут потеряны.

2) Текстовые файлы

Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу **Text**. Текстовый файл трактуется в Турбо Паскале как совокупность строк переменной длины. Доступ к каждой строке возможен лишь последовательно, начиная с первой, т.е. для доступа к какой-либо записи необходимо сначала просмотреть все предыдущие. При создании текстового файла в конце каждой записи (строки) ставится специальный признак EOLn (End Of Line – конец строки), а в конце всего файла – признак EOF (End Of File – конец файла). Эти признаки можно протестировать одноименными логическими функциями. При формировании текстовых файлов используются следующие системные соглашения:

EOLn – последовательность кодов ASCII #13 (CR) и #10 (LF);

EOF – код #26 стандарта ASCII.

Текстовый файл может быть открыт или только *для чтения* (процедурой **Reset**), или только *для записи* (**Rewrite**). Как и для типизированных файлов, использование **Reset** предполагает, что открываемый файл уже существует, иначе возникает ошибка. При использовании **Rewrite** открываемый файл может не существовать, в этом случае он создается, а если он уже существовал, то он очищается.

Для текстовых файлов, помимо **Reset** и **Rewrite**, имеется еще одна процедура открытия (инициации) файла – **Append**. При этом ранее существовавший текстовый файл открывается *для добавления* (или расширения) – т.е. для записи, но без очистки содержимого. При этом указатель файла устанавливается в его конец. Формат обращения:

Append(<*файловая переменная*>)

Если текстовый файл ранее уже был открыт с помощью **Reset** или **Rewrite**, использование процедуры **Append** приведет к закрытию этого файла и открытию его вновь, но уже для добавления записей.

Для доступа к записям служат процедуры **Read**, **ReadLn**, **Write**, **WriteLn**. К ним можно обращаться с переменным числом фактических параметров, в качестве которых могут использоваться символы, строки и числа. Первым параметром в любой из этих процедур должна стоять файловая переменная, связанная с файлом или логическим устройством процедурой **Assign**. Если файловая переменная не указана, происходит обращение к стандартным файлам **Input** и **Output**, т.е. ввод данных будет происходить с клавиатуры, а вывод – на экран.

Процедура **Read** обеспечивает ввод символов, строк и чисел. Формат обращения:

Read(<*файловая переменная*>, <*список ввода*>)
Read(<*список ввода*>)

Здесь *список ввода* – последовательность из одной или более переменных типа **Char**, **String**, а также любого целого или вещественного типа.

Процедура **ReadLn** обеспечивает ввод символов, строк и чисел. Эта процедура идентична процедуре **Read** за исключением того, что после считывания последней переменной оставшаяся часть строки до маркера **EOLn** пропускается, поэтому следующее обращение к **ReadLn** или **Read** начинается с первого символа новой строки. Кроме того, эту процедуру можно вызвать без списка ввода, что приведет к пропуску всех символов текущей строки вплоть до **EOLn**.

Процедура **Write** обеспечивает вывод информации в текстовый файл или передачу ее на логическое устройство. Формат обращения:

Write(<*файловая переменная*>, <*список вывода*>)
Write(<*список вывода*>)

Здесь список вывода – последовательность из одного или более выражений типа **Char**, **String**, **Boolean**, а также любого целого или вещественного типа. Если файловая переменная отсутствует, подразумевается вывод в стандартный файл **Output**, который обычно связан с экраном ПК.

Любой элемент списка вывода может иметь формат

<выводимое выражение> : N1 : N2

Например,

```
WriteLn('x=',x:12:9);
```

Здесь **N1**, **N2** – целые числа (типа **Word**). Оба они могут отсутствовать. Подпараметр **N1**, если он есть, указывает минимальную ширину поля, в которое будет записываться символьное представление выводимого значения. Если символьное представление имеет меньшую длину, чем **N1**, оно будет дополнено слева пробелами, если большую длину, то подпараметр **N1** игнорируется и выводится необходимое число символов.

Подпараметр **N2** задает количество десятичных знаков в дробной части вещественного числа. Он может использоваться только совместно с **N1** и только по отношению к выводимому выражению одного из вещественных типов. Если не указан подпараметр **N2**, вещественные числа выводятся в экспоненциальном формате (например, **-2.34567890234567E+0002**).

Если ширина поля вывода не указана, то значение выводится вслед за предыдущим без какого-либо их разделения.

Символы и строки передаются выводному файлу без изменений, но снабжаются ведущими пробелами, если задана ширина поля вывода и эта ширина больше требуемой для вывода.

Процедура **WriteLn** полностью идентична процедуре **Write** за исключением того, что выводимая строка символов завершается кодами CR и LF. При вызове **WriteLn** можно опускать параметр ***<список вывода>***: в этом случае в файл передается маркер **EOLn**, что при выводе на экран приведет к переводу курсора в начало следующей строки.

Логическая функция **EOLn** возвращает TRUE, если во входном текстовом файле достигнут маркер конца строки. Формат обращения:

EOLn(<*файловая переменная*>)

Если параметр ***<файловая переменная>*** опущен, функция проверяет стандартный файл **Input**.

Практические задания

1. Дан файл *f*, компоненты которого являются целыми числами. Получить файл *g* из компонент файла *f* делением их на максимальное число среди этих компонент.

```
Program files;
var  n,i,x,xmax: integer;
     y: real;
     f: file of integer; {файловые переменные}
     g: file of real;
begin
  assign(f,'name1.dat');   {связывание файловых пере-}
  assign(g,'name2.dat');   { менных с именами файлов}
  rewrite(f);              {открытие нового файла }
  writeln ('Введите количество чисел ');
  readln(n);
  writeln ('Введите целые числа:');
  for i:=1 to n do         {в цикле – ввод числа }
  begin                    {и запись его в файл f }
    read(x);
    write(f, x);
  end;
  close(f);                {файл f закрыт}
  reset(f);                {открытие файла f для чтения}
  read(f, xmax);           {инициализация переменной xmax}
  while not EoF(f) do {чтение файла f и поиск максимума}
  begin
    read(f, x);
    if (xmax < x) then xmax:= x;
  end;
  seek(f, 0);          {установить указатель файла f на 0}
  rewrite(g);          {открытие файла g для записи}
  while not EoF(f) do {в цикле формирование файла g}
  begin
    read(f, x);
    y:= x/max;
    write(g, y);
  end;
```

```
  close(f); close(g);      {закрыть файлы}
  reset(g);          {файл g открыть как существующий}
  writeln ('Вывод файла g:');
  while not EoF(g) n do
  begin
    read(g, y);
    write('   ', y:7:3);
  end;
  close(g);
  readln
end.
```

2. В текстовом файле заданы: в первой строке порядок n квадратной матрицы A, в следующих n строках – элементы матрицы A построчно. Изучить и запустить программу, считывающую матрицу из файла и выполняющую следующие действия: а) поиск максимального и минимального элементов и их позиций, б) транспонирование матрицы, в) сложение матриц A и A^T. Результаты всех действий записать в конец исходного файла.

Указание. Исходный файл можно создать с помощью редактора Турбо Паскаля или программы Блокнот.

```
Program textfile;
var i,j,n:integer;
    a,at:array[1..10,1..10] of real;
    f: text;
    fn: string[20];

Begin
  write('Имя файла:'); readln(fn);
  assign(f,fn);
  reset(f);
  readln(f,n);
  for i:=1 to n do
  begin
    j:=1;
```



```
    while not EoLn(f) do begin
      read(f,a[i,j]); j:=j+1 end;
    readln(f)
  end;
  close(f);
  writeln('Матрица A:');
  for i:=1 to n do
  begin
    for j:=1 to n do write('  ',a[i,j]:7:3);
    writeln
  end;
  {обработка матрицы}
  max:=a[1,1]; min:=a[1,1];
  i1:=1; j1:=1; i2:=1; j2:=1;
  for i:=1 to n do
    for j:=1 to n do
    begin
      if max<a[i,j] then
      begin max:=a[i,j]; i1:=i; j1:=j end;
      if max<a[i,j] then
      begin max:=a[i,j]; i1:=i; j1:=j end;
      at[i,j]:=a[j,i];
      s[i,j]:=a[i,j]+at[i,j];
    end;
  {запись результатов в файл}
  append(f);
  writeln(f);
  writeln(f,'Максимальный элемент a[', i1,
',', j1,']=',max:7:3);
  writeln(f,'Минимальный элемент  a[', i2,
',', j2,']=',min:7:3);
  writeln(f,'Транспонированная матрица AT:');
  for i:=1 to n do
  begin
    for j:=1 to n do write(f,' ',at[i,j]:7:3);
    writeln(f)
```

```
  end;
  writeln(f,'Сумма S=A+AT:');
  for i:=1 to n do
  begin
    for j:=1 to n do write(f,' ',s[i,j]:7:3);
    writeln(f)
  end;
  close(f);
  readln
End.
```

Задания для самостоятельной работы

1. Дан файл *f*, компоненты которого являются действительными числами. Найти:

а) сумму компонент файла *f*;

б) наибольшее и наименьшее значение среди компонент;

в) разность первой и последней компонент.

2. Дан файл *f*, компоненты которого являются целыми числами. Найти:

а) количество четных чисел среди компонент;

б) среднее арифметическое всех компонент, не кратных 3;

в) максимальную сумму, составленную из трех идущих подряд компонент.

3. Дан текстовый файл *f*.

а) Найти самое длинное слово в файле. (Словом считать группу отображаемых символов, не содержащую внутри себя пробелы, запятые, точки и т.п.);

б) посчитать количество слов в файле;

в) определить, сколько в файле слов, состоящих из одного, двух и трех символов;

Указание. В заданиях, где требуется обработать уже *существующий* файл, последний должен быть предварительно создан этой же или отдельной программой. Для создания текстового (символьного) файла может быть использован любой текстовый редактор.

Лабораторная работа №7
Графика в Турбо Паскале

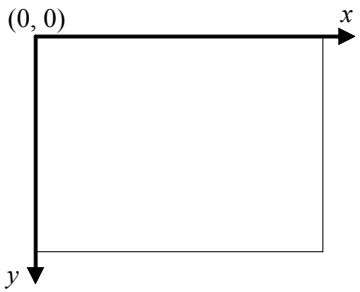
В отличие от телевизора, компьютерный монитор имеет не один (с точки зрения вывода изображения) режим работы, а два – *текстовый* и *графический* режимы.

Различие между этими режимами заключается в возможностях управления выводом визуальной информации. В текстовом режиме минимальным объектом, отображаемым на экране, является *символ*, алфавитно-цифровой или какой-либо иной. Обычно экран монитора, работающего в текстовом режиме, позволяет отобразить 80 символов по горизонтали и 25 символов по вертикали, т.е. получаем матрицу 25×80 (25 строк, 80 столбцов), что дает всего 2000 визуальных объектов. Нумерация строк и столбцов начинается с 1, т.е. символ, стоящий в верхнем левом углу, имеет координаты (1, 1).

В текстовом режиме имеется возможность управлять только цветом символов, фоновым цветом и включать/отключать режим мерцания. Процедуры и функции для работы в текстовом режиме содержит стандартный модуль CRT. Для работы с изображениями текстовый режим абсолютно не подходит.

В графическом режиме минимальным объектом, отображаемым на экране, является так называемый *пиксел* – от английского *pixel*, возникшего в результате объединения слов «рисунок» (picture) и «элемент» (element). Пиксел имеет меньшие размеры по сравнению с символом – на один символ в текстовом режиме отводится площадка размером в несколько пикселов. Его геометрические размеры определяются разрешением монитора, которое обычно задается в виде $rx \times ry$, где rx – количество пикселов на экране по горизонтали, а ry – количество пикселов по вертикали. На практике используются не произвольные, а некоторые определенные значения разрешения, например, 320×200, 640×480, 800×600, 1024×768, 1280×1024 и т.д. Наиболее часто при программировании на Турбо Паскале для DOS используется разрешение 640×480 (режим VGA).

Чтобы запрограммировать вывод какого-либо графического объекта, надо научиться задавать его координаты на экране.



Графические координаты задают положение точки на экране дисплея. Поскольку минимальным элементом, к которому имеет доступ программист, является пиксел, естественно в качестве графических координат использовать порядковые номера пикселов. Допустимый диапазон изменения графических координат составляет [0, $rx - 1$] для координаты x и [0, $ry - 1$] для y-координаты. Точкой отсчета является верхний левый угол экрана, его координаты (0, 0). Значения x-координаты отсчитываются слева направо, а y-координаты – сверху вниз. Последнее отличает графические координаты от обычных декартовых координат, принятых в математике, и служит источником ошибок для начинающего программиста.

Проблема заключается в том, что при разработке программы график или другое изображение обычно проектируется в привычной для нас декартовой системе координат. Но для правильного отображения такого графика на экране необходимо учесть различия между декартовой и графической системами координат. Таких различий три:

1. Графические координаты принимают только целочисленные значения.

2. Графические координаты принимают значения, ограниченные как снизу (нулевым значением), так и сверху (значением разрешения).

3. Графическая координата y отсчитывается сверху вниз.

Таким образом, геометрические декартовы координаты точки (x, y) для отображения ее на экране следует пересчитать в графические (xg, yg) по формулам

$$xg = [sx \times x] + dx, \quad yg = ry - [sy \times y] - dy,$$

где $[x]$ – целая часть x; sx и sy – масштабные множители, выбираемые из условия

$$rx = [sx \times x_{max}] + 1, \quad ry = [sy \times y_{max}] + 1.$$

Здесь x_{max} и y_{max} – максимальные значения геометрических координат. Пересчет координаты y по такой же формуле, что и для x, привел бы к выводу перевернутого изображения. Слагаемые dx и dy обеспечивают смещение изображения относительно левого верхнего угла экрана. Изображение будет смещено в центр экрана при

$$dx = [rx / 2], \quad dy = [ry / 2].$$

Все процедуры и функции для работы в графическом режиме находятся в стандартном модуле **`Graph`** (файл Graph.tpu). Этот модуль должен явно подключаться к программе в разделе **`Uses`**. В модуль Graph входит около 80 процедур и функций.

Для отображения графики видеоадаптер ПК должен поддерживать работу дисплея в графическом режиме. Работой видеоадаптера управляет специальная программа, которая называется *драйвером.* Драйвер хранится в отдельном файле на диске и содержит как исполняемый код, так и необходимые ему для работы данные. Признак файла с драйвером – расширение .bgi имени файла (bgi – Borland Graphics Interface). Имя файла с драйвером соответствует типу видеоадаптера вашего компьютера, так, для подавляющего большинства моделей это файл egavga.bgi (или более современный svga.bgi).

Большинство видеоадаптеров могут работать в нескольких *графических режимах.* Эти режимы различаются прежде всего разрешением и набором доступных цветов. Итак, программа при переключении в графический режим должна определить тип видеоадаптера. Это можно сделать, явно указав в программе тип видеоадаптера или дав программе возможность самостоятельно определить значение соответствующих параметров. Для этого необходимо ввести переменную целого типа, пусть ее идентификатор будет **`gd`**. При явном определении ви-

деоадаптера в программе должен присутствовать оператор присваивания **gd:=<*значение*>**, где **<*значение*>** – это либо некоторое число, либо встроенная константа (встроенные константы не надо описывать специально, так как их описания содержатся в модулях). Возможные значения приведены в справочной системе.

При автоматическом распознавании видеоадаптера в правой части оператора присваивания используется константа **Detect** (или нулевое значение).

Второе, что должна сделать программа, – задать определенный графический режим. Для этого следует ввести еще одну переменную целого типа, назовем ее **gm**, и присвоить ей значение. Допустимые значения также приведены в справочной системе.

Проще всего, однако, вызвать стандартную процедуру **DetectGraph(gd,gm)**, которая автоматически определит номер типа видеоадаптера и номер режима с максимальным разрешением и присвоит эти значения переменным **gd**, **gm**.

Переключение в графический режим работы дисплея выполняется вызовом процедуры **InitGraph**:

```
InitGraph(gd, gm, 'c:\tp\bgi');
```

Первый параметр в этой процедуре задает тип видеоадаптера, второй определяет режим, а третий представляет собой строку с указанием расположения файла драйвера на диске. Пустая строка означает, что графический драйвер находится в том же каталоге, что и программа.

Завершение работы в графическом режиме производится с помощью процедуры **CloseGraph**, которая выгружает драйвер из памяти и восстанавливает предыдущий видеорежим.

Процедуры **RestoreCrtMode** и **SetGraphMode** позволяют переключаться между текстовым и графическим режимами, не закрывая графический режим.

В каждый момент времени в графическом режиме один из пикселов считается *текущей позицией* (current pointer – CP). Многие процедуры и функции производят вывод соответствующего объекта (линии, прямоугольника, текста, и т.д.), начиная именно с этой текущей позиции. Определить ее можно с помощью функций **GetX**, **GetY**, которые возвращают координаты *x* и *y* текущей позиции соответственно.

Среди графических процедур и функций наиболее часто используются следующие (подробное описание и параметры смотрите в справочной системе):

DetectGraph – автоопределение графического режима и графического драйвера.
InitGraph – переход в графический режим.
CloseGraph – выход из графического режима.
PutPixel – рисует точку (пиксел) указанного цвета в позиции с указанными координатами.
GetPixel – функция, возвращающая цвет пиксела с указанными координатами.
SetColor – устанавливает новый цвет вывода (номера смотрите в справочной системе).
GetColor – функция, возвращающая текущий цвет вывода.
SetLineStyle – устанавливает стиль, шаблон и толщину для выводимых далее линий.
GetLineSettings – возвращает текущие значения стиля, шаблона и толщины линий.
Line – рисует линию от первой точки с указанными координатами (x1, y1) до второй (x2, y2).
LineTo – рисует линию от CP до конечной точки с указанными координатами.
LineRel – рисует линию от CP, смещаясь по *x* и *y* на указанные величины.
MoveTo – смещает CP в точку с с указанными координатами.
MoveRel – смещает CP на указанные величины по *x* и *y*.
OutText – выводит указанную строку, начиная с CP.

OutTextXY – выводит указанную строку, начиная с указанной позиции.
Rectangle – рисует четырехугольник линиями текущего цвета и стиля.
Bar – рисует четырехугольник текущим цветом и закрашивает его.
Bar3D – рисует прямоугольный параллелепипед.
Circle – рисует окружность указанного радиуса с центром в указанной точке.
Arc, **Ellipse** – рисуют соответственно дугу окружности или эллипса.
Sector – рисует и закрашивает эллиптический сектор.
DrawPoly – рисует многоугольник.
SetViewPort – устанавливает окно вывода со своей системой координат.
ClearViewPort – очищает текущее окно вывода.
GetMaxX, **GetMaxY** – возвращают текущее разрешение экрана по оси *x* и *y* соответственно.
FloodFill – закрашивает замкнутую область.
FillPoly – закрашивает многоугольник.
GetImage – сохраняет указанную часть изображения в памяти по указанному адресу.
PutImage – выводит на экран в указанной позиции изображение, сохраненное **GetImage**.
ImageSize – определяет размер в байтах указанной части изображения.

Практические задания

1. Изучить и запустить программу вывода на экран графика функции $f(x)=\sin(3x)/(x^2+5)$ на отрезке [–10, 10]. Изменить программу, добавив вывод графика еще одной функции (по выбору).

```
Program function_graph;
uses graph,crt;
```



```
var gd,gm,n,i: integer;
    a,b,h,xmas,ymas,max,min,x,y: double;
    c: char;

function f(x: double):double;
begin f:=sin(3*x)/(x*x+5) end;

Begin
  detectgraph(gd,gm);
  initgraph(gd,gm,'');  {Вход в графический режим}
  a:=-10; b:=10; n:=500; h:=(b-a)/n;
  {Вычисление масштабных коэффициентов}
  xmas:=500/(b-a);
  max:=f(a); min:=f(a);
  for i:=1 to n do
  begin
    x:=a+i*h;
    if max<f(x) then max:=f(x);
    if min>f(x) then min:=f(x)
  end;
  ymas:=400/(max-min);
  setviewport(70,40,570,440,ClipOn);
  setcolor(15);
  {Рисование осей координат}
  y:=max*ymas;
  line(0,round(y),500,round(y));
  outtextxy(490,round(y)-10,'X');
  x:=-a*xmas;
  line(round(x),0,round(x),400);
  outtextxy(round(x)-10,0,'Y');
  {Рисование графика}
  setcolor(14);
  moveto(0, round(y-f(a)*ymas));
  for i:=1 to n do
  begin
    x:=a+i*h;
    lineto(round((x-a)*xmas),round(y-f(x)*ymas));
  end;

  c:=readkey;
```

```
  closegraph   {Выход из графического режима}
End.
```

2. Изучить и запустить программу вывода на экран квадрата, вращающегося относительно одного из углов. Изменить программу, чтобы квадрат вращался вокруг центра симметрии.

```
uses graph,crt;
type arr5=array[1..5] of double;
var a,r: arr5;
    t,dt: double;
    gd,gm: integer;

procedure rect(r,a:arr5; t:double);
var i,x,y: integer;
begin
  x:=320+round(r[1]*cos(a[1]+t));
  y:=240-round(r[1]*sin(a[1]+t));
  moveto(x,y);
  for i:=2 to 5 do begin
    x:=320+round(r[i]*cos(a[i]+t));
    y:=240-round(r[i]*sin(a[i]+t));
    lineto(x,y);
  end
end;

Begin
  r[1]:=0; r[2]:=100; r[3]:=100*sqrt(2);
  r[4]:=100; r[5]:=0;
  a[1]:=0; a[2]:=0; a[3]:=pi/4; a[4]:=pi/2; a[5]:=0;
  detectgraph(gd,gm);
  initgraph(gd,gm,'');
  setlinestyle(0,0,3);
  t:=0; dt:=pi/180;
  repeat
    setcolor(15);
    rect(r,a,t);
    delay(1000);
    setcolor(0);
    rect(r,a,t);
    t:=t+dt;
  until keypressed;
```



```
  closegraph;
End.
```

Задания для самостоятельной работы

1. Построить:

а) треугольник с вершинами (100, 100), (150, 100), (80, 170);

б) два прямоугольника 300×200 в левом верхнем углу и в центре экрана;

в) шестиугольник, вписанный в окружность радиусом 120 в центре экрана.

2. Дано натуральное число n ($n \le 999999$). Записать его шестью цифрами, используя девятисегментный шаблон (как на почтовых конвертах).

3. Вывести на экран графики следующих кривых:

а) астроида $x = \cos^3 t, y = \sin^3 t, t \in [0, 2\pi]$;

б) кардиоида $x = a\cos t(1+\cos t)$, $y = a\sin t(1+\cos t)$, $a > 0$, $t \in [0, 2\pi)$.

в) циклоида $x = at^2(1+t^2)$, $y = at^3(1+t^2)$, $a > 0$, $t \in (-\infty, \infty)$;

г) улитка Паскаля $x = a\cos^2 t + b\cos^2 t$, $y = a\cos t \sin t + b\sin t$, $a > 0$, $b > 0$, $t \in [0, 2\pi)$. Рассмотреть случаи, когда $b \ge 2a$, $a < b < 2a$, $a > b$.

4. Изобразить на экране точку, движущуюся по окружности с постоянной угловой скоростью.

5. Изобразить на экране две касающиеся окружности радиусов r_1 и r_2, вращающиеся в плоскости экрана вокруг точки касания, расположенной в центре экрана.

6. Изобразить на экране стрелку, вращающуюся в плоскости экрана вокруг тупого конца, который расположен в центре экрана.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. М.: Издательство «ОМД Групп», 2003. – 616 с.

2. Фаронов В.В. Турбо Паскаль 7.0: Учеб. пособие. СПб.: Питер, 2007. – 367 с.

3. Программирование на языке Паскаль: задачник / под ред. Усковой О.Ф. – СПб: Питер, 2003. – 336 с.

4. Немнюгин С.А. Turbo Pascal. – СПб: Питер, 2000. – 496 с.

5. Кострюков С.А., Моисеев С.И., Пантелеев И.Н. Практикум на Паскале. Учеб. пособие. Воронеж: Изд-во ВГТУ, 2000.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 1
Знакомство со средой программирования Turbo Pascal 7.0 2
Лабораторная работа №1 3
Лабораторная работа №2 10
Лабораторная работа №3 14
Лабораторная работа №4 20
Лабораторная работа №5 26
Лабораторная работа №6 35
Лабораторная работа №7 46
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 55

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ТУРБО ПАСКАЛЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по курсам
«Информатика» и «Специальные главы информатики»
для студентов направлений 150400 «Металлургия»,
140700 «Ядерная энергетика и теплофизика» и
131000 «Нефтегазовое дело» очной формы обучения

Составители:
Кострюков Сергей Александрович
Пешков Вадим Вячеславович
Шунин Генадий Евгеньевич

В авторской редакции

Компьютерный набор В.В. Пешкова

Подписано к изданию 28.01.2014.

Уч.-изд. л. 3,4. “С” .

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14