МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Политехнический институт
Кафедра «Технологические системы пищевых, полиграфических
и упаковочных производств»

Утверждено на заседании кафедры
«Технологические системы пищевых,
полиграфических и упаковочных
производств»
«26» января 2022 г., протокол № 6

Заведующий кафедрой
_____В.В. Прейс

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРАКТИЧЕСКИМ (СЕМИНАРСКИМ) ЗАНЯТИЯМ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

«Информатика»

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
29.03.03 Технология полиграфического и упаковочного производства

с направленностью (профилем)
Технология полиграфического производства

Формы обучения: заочная

Идентификационный номер образовательной программы: 290303-01-22

Тула 2022 год

ЛИСТ СОГЛАСОВАНИЯ
методических указаний по проведению практических занятий по дисциплине (модулю)

Разработчик:

Проскуряков Н.Е., профессор, докт. техн. наук, профессор
(ФИО, должность, ученая степень, ученое звание)

(подпись)

СОДЕРЖАНИЕ

Практическое занятие №1
МАССИВЫ

1. ЦЕЛЬ И ЗАДАЧИ ЗАНЯТИЯ

Цель занятия: усвоение технологии занятия с массивами.

Задачи занятия: построить схему алгоритма обработки двумерного массива с использованием вложенных циклов и последующая разработка программы.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Массивы.

Одним из структурированных типов данных является регулярный тип или массив. Массив представляет собой совокупность связанных данных, состоящую из фиксированного числа элементов одного типа, который называется базовым. Для определения массива достаточно указать его базовый тип, а также число элементов в массиве и метод их нумерации. Возможен доступ к отдельным элементам массива – для этого достаточно указать имя массива и номер (индекс) нужного элемента. Описание регулярного типа имеет следующий вид:

```
Type  s=Array [i] Of a;
```

Здесь Array и Of – зарезервированные слова, имеющие смысл: «массив» и «из»; s – имя регулярного типа; i – тип индексов (указывается в квадратных скобках); a – тип элементов массива (базовый тип). Элементы массива могут представлять собой значения любого типа. Для индексирования массива в Turbo Pascal используется тип данных. Это может быть любой из целочисленных типов (за исключением Longint), и Boolean, а также перечислимый и интервальный типы (за исключением интервальных типов на основе Longint). Вещественные типы для индексирования массивов не годятся.

Также, массив можно описать непосредственно в разделе описания переменных.

Пример

```
var
  a1:Array [byte] of  boolean;
  a2:Array [char] of boolean;
  a3:Array [Red,Yellow,Green] of char;
  a4:Array [1..100] of integer;
```

Обращения к отдельным элементам массивов A1 и A2 выглядят так:

```
a1 [90]:=false;
a2 [z]:=true;
```

Следует понимать, что индексы массива могут представляться не только в виде явно заданных значений, но и переменных, и выражений.

Разумеется, массивы можно объявлять не только в качестве анонимных типов, но и как типы с собственным именем – в разделе описания типов:

```
Type
   LogScale=Array [byte] of boolean;
Var
   a1:LogScale;
```

Массив – это последовательность однотипных элементов, так же, как и строка, однако если символы строки представляют собой только значения типа Char, то элементы массива могут принадлежать различным типам – как простым, так и структурированным.

Обращения к отдельным элементам строк и массивов выглядят одинаково – для этого достаточно указать имя строки или массива и индекс. Однако имеются и отличия: если текущей длиной строки можно манипулировать, то для массивов ничего подобного не предусмотрено. Длина строки также ограничена 255 символами, в то время как для массивов такого ограничения нет. Кроме того, можно ввести или вывести значение всей строковой переменной с помощью единственного оператора (например, `Read(X)` или `Write(X)`, где X – значение типа STRING), а для массивов это невозможно.

В целом же вместо массива символьных значений часто можно использовать строку, и наоборот. Какому типу данных следует отдать предпочтение в том или ином случае, зависит от особенностей применения.

2. Многомерные массивы.

В Turbo Pascal элементы массива могут представлять собой значение не только простых, но и структурированных типов. Допускается, например, существование массивов, элементами которых являются также массивы. Описание подобного массива выглядит так:

```
a1=array[1..5] of array [1..4] of integer;
```

Этот массив можно представить в качестве двумерной матрицы с 20 элементами (4 на 5). Возможна сокращенная запись приведенного выше описания массива:

```
a1=array [1..5, 1..4] of integer;
```

Эти описания эквивалентны. Обращение к одному из элементов данного массива выглядит так:

```
a1[3][4]:=13;
```

или

```
a1[3,4]:=13;
```

Возможны не только двумерные, но и многомерные массивы, причем число измерений не ограничивается. Однако при этом сохраняется ограничение в 64К (или 65520 байт) на размер переменных регулярного типа. Это связано с максимальным объемом памяти, выделяемой для данной переменной.

3. Применимые к массивам операции.

Поскольку элемент массива трактуется как переменная, он может фигурировать в выражениях. Набор операций над элементами массива соответствует операциям, допустимым для базового типа.

Таблица 1.

Примеры действий над элементами массива

Пример действия	Комментарий
`Write(6,a1[5])`	На экране отображается число 6, а затем значение 5-го элемента массива A1.
`a4[5]:=a4[3]+a4[2]`	Значения 3 и 2-го элементов массива A4 складываются, и полученная сумма в качестве значения присваивается элементу 5 того же массива.
`abc:=abc+a4[88]`	Значение 88-го элемента массива A4 суммируется со значением переменной abc, и полученная сумма присваивается этой же переменной.
`abc:=a4[55]+a4[56]`	Значения 55 и 56-го элементов массива A4 складываются, и полученная сумма присваивается переменной abc.
`a4[33]:=a4[33]+20`	Значение 33-го элемента массива A4 увеличивается на 20.

Что касается набора операций над массивами в целом, то скопировать все элементы из одного массива в другой можно единственным оператором присваивания. Например, если X и Y –массивы, принадлежащие одному типу, то правомерен оператор

```
X := Y;
```

Этот оператор копирует значения всех элементов массива X в массив Y. Однако нельзя использовать с массивами операции сравнения. Тем не менее, если такая необходимость существует, можно организовать поэлементное сравнение массивов.

Также абсолютно неприменимы к массивам арифметические и логические операции.

Кроме того, к массивам (в отличие от строк) нельзя применять стандартные процедуры Read и Write. Однако можно организовать считывание или вывод на экран каждого элемента массива в отдельности.

Массивы относятся к структурированным типам данных в Turbo Pascal. Массив состоит из фиксированного числа элементов (компонент) одного типа и характеризуется общим именем. Доступ к отдельным элементам массива осуществляется с помощью общего имени и порядкового номера (индекса или адреса) необходимого элемента.

Имя массива — это любое допустимое в Turbo Pascal имя, отличное от служебных слов, имен функций и процедур. Массив может быть описан в подразделе Var или в подразделах Var и Type, одновременно.

Массив с одним индексом называют **одномерным**, с двумя - **двумерным**, с тремя - **трехмерным** и т.д. Число индексов у массива в Turbo Pascal не ограничивается, но необходимо помнить, что размер массива не должен превышать 64 Кбайт.

Любой двумерный массив представляет собой матрицу: первому индексу можно поставить в соответствие строки, а второму - столбцы матрицы. Кроме того, двумерный массив можно интерпретировать как одномерный, элементами которого является другой одномерный массив.

Описание такого массива имеет вид:

```
Type  tstr=array[1..25] of real;
Var  masssiv:array[1..10] of tstr;
```

Это описание равносильно описанию в примере, приведенному выше для массивов с именами Matr1 и Matr2.

Оперативная память под элементы массива выделяется на этапе трансляции. В памяти компьютера элементы массива следуют друг за другом.

Если массив двумерный, то память под него выделяется так, что быстрее меняется самый правый индекс. В качестве примера рассмотрим порядок выделения оперативной памяти под массив, описанный следующим образом: Var M:array[1..2,1..4] of byte;

Например:

Этот массив будет располагаться в памяти в следующем порядке:

```
M[1,1]; M[1,2]; M[1,3]; M[1,4]; M[2,1]; M[2,2]; M[2,3]; M[2,4].
```

В Turbo Pascal можно одним оператором присваивания передать все элементы одного массива другому массиву того же типа.

Например:

```
Var   m1,m2:array[1..10] of word;
. . .
Begin
. . .
m1:=m2;  { перезапись из одного массива в другой}
. . .
End.
```

Для **сравнения** содержимого двух массивов необходимо использовать оператор цикла с параметром и указываться индексы.

У т**ипизированных констант–массивов** в качестве начального значения используется список констант, отделенных друг от друга запятыми. Список заключается в круглые скобки.

Например:

1)

```
Const Mas:array[1..10] of byte=(1,1,1,1,1,1,1,1,1,1);
```

Заполнение массива из 10 целых чисел значением, равным единице.

2)

```
Const     massim: array[0..5] of  char = ('a','b','c',
'd','e','f');
```

Заполнение массива из 6 элементов символами - буквами латинского алфавита.

3)

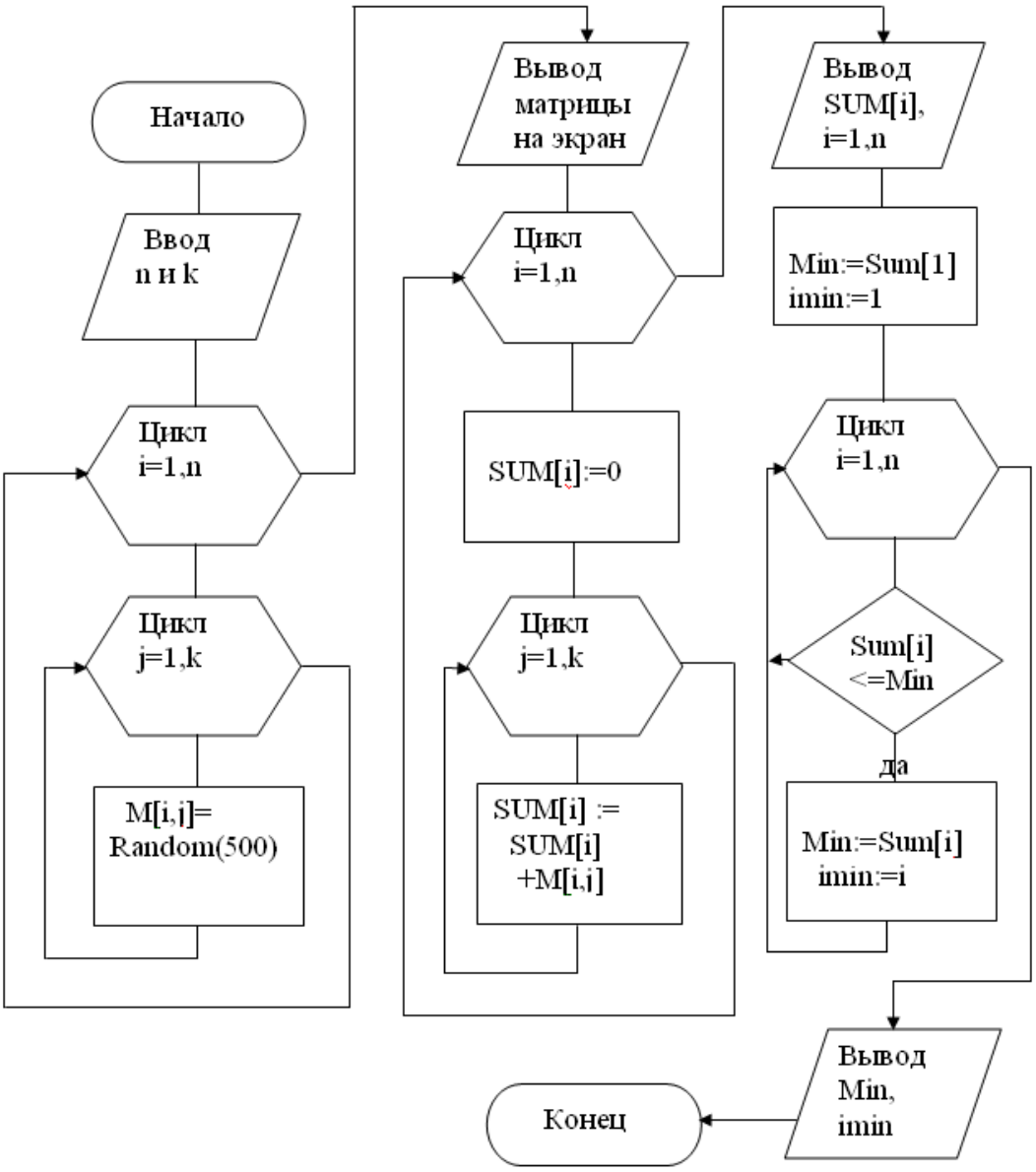
```
Const     Matr:  array[1..5,1..2]of  byte  =  ((0,0),
(0,0),(0,0),(0,0),(0,0));
```
Обнуление матрицы из 10 целых чисел.

Замечание: количество переменных в списке констант должно строго соответствовать объявленной длине массива по каждому индексу!

3. Типовой пример

Составить схему алгоритма и программу определения сумм элементов в каждой строке матрицы $M=\{m_{ij}\}$, $i=1..n$, $j=1..k$, где n - число строк, k - число столбцов матрицы, m_{ij} - целые числа из диапазона: от 0 до 500. Записать полученные значения сумм в одномерный массив «sum», а затем вывести их на экран дисплея. Числа в массив M занести с помощью функции `Random`. Определить и вывести на экран номер строки с минимальным значением суммы.

Схема алгоритма



Текст программы

```
Uses crt;
{ Описание данных}
Var M:array[1..50,1..100]of integer;
    Sum:array[1..50] of longint;
    n,k,i,j,imin:integer;
    Min:longint;
BEGIN
clrscr;
{ Ввод данных}
writeln(' Введите число строк и столбцов');
readln(n,k);
Randomize;  { Стандартная процедура см. теорию}
Заполнение матрицы случайными числами}
  for i:=1 to n do
  for j:=1 to k do
       M[i,j]:=Random(500);
writeln('   Элементы заполненной матрицы');
  for i:=1 to n do
       begin
       for j:=1 to k do
       write(M[i,j]:4);
       writeln;
       end;
writeln('  Сумма элементов в каждой строке');
write('  Номера строк :     ');
  for i:=1 to n do
       write(i,'    ');writeln;
write('  Сумма в строке :  ');
  for i:=1 to n do
        begin
        Sum[i]:=0;
           for j:=1 to k do
               Sum[i]:=Sum[i]+M[i,j];
        write(Sum[i], ' ');
        end;
writeln;
{  Поиск минимального значения}
Min:=Sum[1]; imin:=i;
   for i:=1 to n do
       if   Sum[i] <= Min then
            begin
            Min:=Sum[i]; imin:=i;
            end;
writeln('Минимальная сумма =',Min,'в строке',imin);
```

```
readln; END.
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Разработать схему алгоритма и программу решения задачи согласно варианту задания.

1. Составить схему алгоритма и программу согласно варианту задания. В прямоугольной матрице размером $T*M$, имеющей имя MATR содержатся целые числа. T - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

2. В прямоугольной матрице размером $M*T$, имеющей имя MAS содержатся целые числа. M - число строк, T - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

3. В прямоугольной матрице размером $K*M$, имеющей имя MATR содержатся целые числа. K - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

4. В прямоугольной матрице размером $M*K$, имеющей имя MAT содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждой строке. Определить строку с максимальным значением этой суммы и вывести ее номер на экран.

5. В прямоугольной матрице размером $L*M$, имеющей имя MATR содержаться целые числа. L - число строк, M - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

6. В прямоугольной матрице размером $M * K$, имеющей имя MAT содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждой строке. Определить строку с минимальным значением этой суммы и вывести ее номер на экран.

7. В прямоугольной матрице размером $M * K$, имеющей имя MA содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждом столбце. Определить столбец с максимальным значением этой суммы и вывести его номер на экран.

8. В прямоугольной матрице размером $M * K$, имеющей имя M содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

9. В прямоугольной матрице размером $M * K$, , имеющей имя MM содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждом столбце. Определить столбец с максимальным значением этой суммы и вывести его номер на экран.

10. В прямоугольной матрице размером $M * K$, , имеющей имя MAM содержаться целые числа. M - число строк, K - число столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

11. В квадратной матрице размером $K * K$, имеющей имя MA содержаться целые числа. K - число строк и столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму положительных элементов в каждом столбце. Определить столбец с минимальным значением этой суммы и вывести его номер на экран.

12. В квадратной матрице размером $T * T$, имеющей имя MKA, содержаться целые числа. T - число строк и столбцов. Ввести элементы матрицы с клавиатуры. Определить и вывести на экран сумму отрицательных элементов в каждом столбце. Определить столбец с максимальным по модулю значением этой суммы и вывести его номер на экран.

5. ПОРЯДОК ВЫПОЛНЕНИЯ ЗАНЯТИЯ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «prakt1.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) В чем отличие одномерных и многомерных массивов?
2) Что такое базовый тип массива?
3) Расскажите о создании пользовательского типа массива?
4) Какие операции применимы к массивам в целом?
5) В каких случаях необходима поэлементная обработка массивов?

Практическое занятие №2
ВНЕШНИЕ ФАЙЛЫ

1. ЦЕЛЬ И ЗАДАЧИ ЗАНЯТИЯ

Цель занятия: усвоение принципов использования внешних файлов.

Задачи занятия: освоить использование внешних файлов для ввода и вывода данных.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Концепция файла.

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

```
Assign (файловая переменная, имя файла);
```

Здесь файловая переменная - идентификатор, объявленный в программе как переменная файлового типа; имя файла - текстовое выражение, содержащее имя файла.

Имя файла - это любое выражение строкового типа, которое строится по правилам определения имен в MS-DOS (операционной системе ПК). Имя содержит до восьми разрешенных символов. За именем может следовать расширение - последовательность до трех разрешенных символов; расширение, если оно есть, отделяется от имени точкой. Перед именем может указываться так называемый путь к файлу: имя диска и/или имя текущего каталога и имена каталогов вышестоящих уровней. Весь путь к файлу отделяется от имени файла обратной косой чертой. Максимальная длина имени вместе с путем - 79 символов.

Пример

```
var  finp: text;
     fout: file of String;
const name = 'c:\dir\subdir\out.txt';
Begin
     Assign(finp,'123.dat') ;
     Assign(fout,name);
End.
```

Инициировать файл означает указать для этого файла направление передачи данных. В Турбо Паскале можно открыть файл для чтения, для записи информации, а также для чтения и записи одновременно.

Для чтения файл инициируется с помощью стандартной процедуры Reset:

```
Reset (файловая переменная);
```

Здесь файловая переменная, связанная ранее процедурой Assign с уже существующим файлом или логическим устройством-приемником информации. При выполнении этой процедуры дисковый файл или логическое устройство подготавливается к чтению информации. В результате

специальная переменная-указатель, связанная с этим файлом, будет указывать на начало файла, т.е. на компонент с порядковым номером 0.

В Турбо Паскале разрешается обращаться к типизированным файлам, открытым процедурой Reset (т.е. для чтения информации), с помощью процедуры Write (т.е. для записи информации). Такая возможность позволяет легко обновлять ранее созданные типизированные файлы и при необходимости расширять их. Для текстовых файлов, открытых процедурой Reset, нельзя использовать процедуру Write или Writeln.

```
Rewrite (файловая переменная);
```

Инициирует запись информации в файл или в логическое устройство, связанное ранее с файловой переменной. Процедурой Rewrite нельзя инициировать запись информации в ранее существовавший дисковый файл: при выполнении этой процедуры старый файл уничтожается и никаких сообщений об этом в программу не передается. Новый файл подготавливается к приему информации и его указатель принимает значение 0.

Стандартная процедура:

```
Append (файловая переменная);
```

Инициирует запись в ранее существовавший текстовый файл для его расширения, при этом указатель файла устанавливается в его конец. Процедура Append применима только к текстовым файлам, т.е. их файловая переменная должна иметь тип Text (см. выше). Процедурой Append нельзя инициировать запись в типизированный или нетипизированный файл. 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.

2. Общие файловые процедуры.

Ниже описываются процедуры и функции, которые можно использовать с файлами любого вида. Специфика занятия с типизированными, текстовыми и нетипизированными файлами рассматривается в следующих разделах.

```
Close (файловая переменная);
```

Закрывает файл, однако связь файловой переменной с именем файла, установленная ранее процедурой Assign, сохраняется.

При создании нового или расширении старого файла процедура обеспечивает сохранение в файле всех новых записей и регистрацию файла в каталоге. Функции процедуры Close выполняются автоматически по отношению ко всем открытым файлам при нормальном завершении программы. Поскольку связь файла с файловой переменной сохраняется, файл можно повторно открыть без дополнительного использования процедуры Assign.

```
Rename (файловая переменная, новое имя);
```

Переименовывает файл. Здесь новое имя - строковое выражение, содержащее новое имя файла. Перед выполнением процедуры необходимо

закрыть файл, если он ранее был открыт процедурами Reset, Rewrite или Append.

```
Erase (файловая переменная);
```

Уничтожает-файл. Перед выполнением процедуры необходимо закрыть файл, если он ранее был открыт процедурами Reset, Rewrite или Append.

```
Flush(файловая переменная);
```

Очищает внутренний буфер файла и, таким образом, гарантирует сохранность всех последних изменений файла на диске. Формат обращения:

Любое обращение к файлу в Турбо Паскале осуществляется через некоторый буфер, что необходимо для согласования внутреннего представления файлового компонента (записи) с принятым в ДОС форматом хранения данных на диске. В ходе выполнения процедуры Flush все новые записи будут действительно записаны на диск. Процедура игнорируется, если файл был инициирован для чтения процедурой Reset.

```
EOF (файловая переменная) : Boolean;
```

Логическая функция, тестирующая конец файла. Возвращает True, если файловый указатель стоит в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении - что файл исчерпан.

```
ChDir (путь);
```

Изменение текущего каталога. Здесь путь - строковое выражение, содержащее путь к устанавливаемому по умолчанию каталогу.

```
GetDir (устройство, каталог);
```

Позволяет определить имя текущего каталога (каталога по умолчанию). Здесь устройство - выражение типа Word , содержащее номер устройства: 0 - устройство по умолчанию, 1 - диск A, 2 - диск B и т.д. Каталог - переменная типа String, в которой возвращается путь к текущему каталогу на указанном диске.

```
MkDir (каталог);
```

Создает новый каталог на указанном диске. Здесь каталог - выражение типа String, задающее путь к каталогу. Последним именем в пути, т.е. именем вновь создаваемого каталога не может быть имя уже существующего каталога.^

```
RmDir (каталог);
```

Удаляет каталог. Удаляемый каталог должен быть пустым, т.е. не содержать файлов или имен каталогов нижнего уровня.

Ряд полезных файловых процедур и функций становится доступным при использовании библиотечного модуля DOS.TPU, входящего в стандартную библиотеку TURBO.TPL . Эти процедуры и функции указаны ниже. Доступ к ним возможен только после объявления Uses Dos в начале программы.

```
FindFirst (маска, атрибуты, имя);
```

Возвращает атрибуты первого из файлов, зарегистрированных в указанном каталоге. Здесь маска - строковое выражение, содержащее маску файла; атрибуты - выражение типа Byte, содержащее уточнение к маске

(атрибуты); имя - переменная типа SearchRec, в которой будет возвращено имя файла.

При формировании маски файла используются следующие символы-заменители ДОС:

* означает, что на месте этого символа может стоять сколько угодно (в том числе ноль) разрешенных символов имени или расширения файла;

? означает, что на месте этого символа может стоять один из разрешенных символов.

3. Текстовые файлы.

Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу Text. Текстовые файлы предназначены для хранения текстовой информации. Именно в такого типа файлах хранятся, например, исходные тексты программ. Компоненты (записи) текстового файла могут иметь переменную длину, что существенно влияет на характер занятия с ними.

Для доступа к записям применяются процедуры `Read, Readln, Write, Writeln`. Они отличаются возможностью обращения к ним с переменным числом фактических параметров, в качестве которых могут использоваться символы, строки и числа. Первым параметром в любой из перечисленных процедур может стоять файловая переменная. В этом случае осуществляется обращение к дисковому файлу или логическому устройству, связанному с переменной процедурой `Assign`.

```
Read (файловая переменная,список ввода);
```

Обеспечивает ввод символов, строк и чисел. Здесь список ввода - последовательность из одной или более переменных типа Char, String, а также любого целого или вещественного типа.

Процедура `Read` прекрасно приспособлена к вводу чисел. При обращении к ней за вводом очередного целого или вещественного числа процедура «перескакивает» маркеры конца строк, т.е. фактически весь файл рассматривается ею как одна длинная строка, содержащая текстовое представление чисел. В сочетании с проверкой конца файла функцией EOF процедура `Read` позволяет организовать простой ввод массивов данных.

Пример

```
const
     N = 1000; {Максимальная длина ввода}
var
     f : text;
     m : array [1..N] of real;
     i : Integer;
Begin
Assign(f, 'prog.dat') ;
Reset(f); i := 1;
While not EOF(f) and (i <= N) do
Begin
```

```
Read(f ,m[i]);
Inc(i);
End;
Close(f);
     .......
End.
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Заданием на практическое занятие является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий для расчета

№ варианта	Задание
1	Написать программу, которая создает файл result.txt, запрашивает произвольное число и число его степеней, после чего записывает в файл result.txt требуемое число степеней исходного числа.
2	Написать программу, которая позволяет дописывать в файл phone.txt фамилии и номера телефонов. В файле каждый элемент данных (имя, фамилия, телефон) должен находиться в отдельной строке.
3	Написать программу, которая создает файл result.txt, запрашивает 7 пар чисел, после чего записывает в файл result.txt попарно эти числа, а также их среднее арифметическое.
4	Написать программу, которая позволяет дописывать в файл dates.txt фамилии и даты рождения. В файле каждый элемент данных (имя, фамилия, дата рождения) должен находиться в отдельной строке.
5	Написать программу, которая создает файл result.txt, запрашивает 5 пар чисел, после чего записывает в файл result.txt попарно эти числа, а также их среднее геометрическое.
6	Написать программу, которая позволяет найти нужные сведения в телефонном справочнике phone.txt. Программа должна запрашивать фамилию человека и выводить его телефон. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.

7	Написать программу, которая создает файл result.txt, запрашивает 6 произвольных чисел после чего записывает в файл result.txt исходные числа и их факториалы.
8	Написать программу, которая позволяет найти нужные сведения в списке дней рождения справочнике dates.txt. Программа должна запрашивать фамилию человека и выводить его дату рождения. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.
9	Написать программу, которая создает файл result.txt, запрашивает 17 произвольных чисел после чего записывает в файл result.txt исходные числа, их квадраты и квадратные корни.
10	Написать программу, которая позволяет дописывать в файл countries.txt фамилии и страну проживания. В файле каждый элемент данных (имя, фамилия, название страны) должен находиться в отдельной строке.
11	Написать программу, которая создает файл result.txt, запрашивает 12 произвольных чисел после чего записывает в файл result.txt исходные числа, их кубы и кубические корни.
12	Написать программу, которая позволяет найти нужные сведения в списке адресов в справочнике countries.txt. Программа должна запрашивать фамилию человека и выводить название страны в которой он живет. Если в справочнике есть одинаковые фамилии, то программа должна вывести список всех людей, имеющих эти фамилии.

5. ПОРЯДОК ВЫПОЛНЕНИЯ ЗАНЯТИЯ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «prakt2.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите о внешних файлах.
2) Какие параметры файлов вы знаете?
3) Что понимается под файловой переменной?

4) Как инициализировать файл для записи?
5) Как инициализировать файл для чтения?

Практическое занятие №3
ПОСТРОЕНИЕ ГРАФИЧЕСКИХ ПРИМИТИВОВ

1. ЦЕЛЬ И ЗАДАЧИ ЗАНЯТИЯ

Цель занятия: освоить создание элементов графического оформление при помощи модуля Graph.

Задачи занятия: изучить принципы использования подключаемых модулей, изучить технологию использования модуля Graph, освоить создание графических примитивов.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Инициализация графического режима.

Для вывода графических изображений на экран турбо Паскаль предоставляет пользователю библиотеку Graph. Общий вид программы с использованием Graph имеет следующий вид:

```
Program ingr;
Uses Graph;
var
    grDriver, grMode, errCode: integer;
Begin
grDriver:= Detect;
InitGraph (grDriver, grMode,'');
If errCode= grOK Then
Begin
    {команды рисования}
End
Else
begin
writeln('Ошибка! Графический режим не удалось
открыть!');
readln;
end.
```

Чаще всего причиной возникновения ошибки при обращении к процедуре InitGraph является неправильное указание местоположения файла с драйвером графического адаптера. Настройка на местоположение драйвера осуществляется заданием маршрута поиска нужного файла в имени драйвера при вызове процедуры InitGraph. Если, например, драйвер зарегистрирован в подкаталоге DRIVERS каталога PASCAL на диске D, то нужно использовать вызов:

```
InitGraph(Driver, Mode, 'd:\Pascal\Drivers');
```

Процедура CloseGraph завершает работу адаптера в графическом режиме и восстанавливает текстовый режим занятия экрана. Заголовок:

```
Procedure CloseGraph;
```

Процедура RestoreCRTMode служит для кратковременного возврата в текстовый режим. В отличие от процедуры CloseGraph не сбрасываются установленные параметры графического режима и не освобождается память, выделенная для размещения графического драйвера. Заголовок:

```
Procedure RestoreCRTMode;
```

2. Процедуры и функции библиотеки Graph.

Многие графические процедуры и функции используют указатель текущей позиции на экране, который в отличие от текстового курсора невидим. Положение этого указателя, как и вообще любая координата на графическом экране, задается относительно левого верхнего угла, который, в свою очередь, имеет координаты 0,0. Таким образом, горизонтальная координата экрана увеличивается слева направо, а вертикальная - сверху вниз.

Процедура SetViewPort устанавливает прямоугольное окно на графическом экране. Заголовок:

```
Procedure SetViewPort(XI,Y1,X2,Y2: Integer; ClipOn: Boolean);
```

Здесь X1...Y2 - координаты левого верхнего и правого нижнего углов окна; ClipOn - выражение типа Boolean, определяющее «отсечку» не умещающихся в окне элементов изображения.

Координаты окна всегда задаются относительно левого верхнего угла экрана. Если параметр ClipOn имеет значение True, элементы изображения, не умещающиеся в пределах окна, отсекаются, в противном случае границы окна игнорируются.

Процедура GetViewSettings возвращает координаты и признак отсечки текущего графического окна. Заголовок:

```
Procedure GetViewSettings(var Viewlnfo: ViewPortType);
```

Здесь Viewlnfo - переменная типа ViewPortType.

Процедура ClearDevice Очищает графический экран. После обращения к процедуре указатель устанавливается в левый верхний угол экрана, а сам экран заполняется цветом фона, заданным процедурой SetBkColor. Заголовок:

```
Procedure ClearDevice;
```

Процедура ClearViewPort очищает графическое окно, а если окно не определено к этому моменту - весь экран. При очистке окно заполняется цветом с номером 0 из текущей палитры. Указатель перемещается в левый верхний угол окна. Заголовок:

```
Procedure ClearViewPort;
```

Ниже приведены наиболее распространенные процедуры графического модуля.

Процедура Rectangle вычерчивает прямоугольник с указанными координатами углов. Заголовок:

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

Здесь X1... Y2 - координаты левого верхнего и правого нижнего углов прямоугольника. Прямоугольник вычерчивается с использованием текущего цвета и текущего стиля линий.

Процедура Circle вычерчивает окружность. Заголовок:

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

Здесь X, Y- координаты центра; R - радиус в пикселях.

Окружность выводится текущим цветом. Толщина линии устанавливается текущим стилем, вид линии всегда SolidLn (сплошная). Процедура вычерчивает правильную окружность с учетом изменения линейного размера радиуса в зависимости от его направления относительно сторон графического экрана, т.е. с учетом коэффициента GetAspectRatio. В связи с этим параметр R определяет количество пикселей в горизонтальном направлении.

Процедура Ellipse вычерчивает эллипсную дугу. Заголовок:

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

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

Процедура SetColor устанавливает текущий цвет для выводимых линий и символов. Заголовок:

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

Здесь Color - текущий цвет.

Функция GetColor возвращает значение типа Word, содержащее код текущего цвета. Заголовок:

```
Function GetColor: Word;
```

Процедура SetBkColor устанавливает цвет фона. Заголовок:

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

Здесь Color - цвет фона. В отличие от текстового режима, в котором цвет фона может быть только темного оттенка, в графическом режиме он может быть любым. Установка нового цвета фона немедленно изменяет цвет графического экрана.

Процедура SetFillStyle устанавливает стиль (тип и цвет) заполнения. Заголовок:

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

Здесь Fill - тип заполнения; Color - цвет заполнения.

Пример

Программа демонстрирует все стандартные типы заполнения.

```
Uses Graph, CRT;
var
d,r,e,k,j,x,y: Integer;
begin
{Инициируем графику}
```

```
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
x := GetMaxX div 6;{Положение графика}
y := GetMaxY div 5;{на экране}
for j := 0 to 2 do{Два ряда}
for k := 0 to 3 do{По четыре квадрата}
begin
Rectangle((k+1)*x,(j+1)*y,(k+2)*x,(j+2)*y);
SetFillStyle(k+j*4,j+1);
Bar((k+1)*x+1,(j+1)*y+1,(k+2)*x-1,(j+2)*y-1)
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Процедура `FloodFill` заполняет произвольную замкнутую фигуру, используя текущий стиль заполнения (узор и цвет). Заголовок:

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

Здесь `X, Y`- координаты любой точки внутри замкнутой фигуры; `Border` - цвет граничной линии.

Если фигура незамкнута, заполнение «разольется» по всему экрану. Следует учесть, что реализованный в процедуре алгоритм просмотра границ замкнутой фигуры не отличается совершенством. В частности, если выводятся подряд две пустые строки, заполнение прекращается.

Процедура `Bar` заполняет прямоугольную область экрана. Заголовок:

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

Здесь `X1...Y2` - координаты левого верхнего (и правого нижнего углов закрашиваемой области. Процедура закрашивает (но не обводит) прямоугольник текущим образцом узора и текущим цветом, которые устанавливаются процедурой SetFillStyle.

Процедура `Bar3D` вычерчивает трехмерное изображение параллелепипеда и закрашивает его переднюю грань . Заголовок:

```
Procedure Bar3D (X1,Y1,X2,Y2,Depth: Integer; Top: Boolean);
```

Здесь `X1... Y2` - координаты левого верхнего и правого нижнего углов передней грани; `Depth` - третье измерение трехмерного изображения («глубина») в пикселях; Top - способ изображения верхней грани.

Процедура `FillPoly` обводит линией и закрашивает замкнутый многоугольник. Заголовок:

```
Procedure FillPoly(N: Word; var Coords);
```

Здесь `N` - количество вершин замкнутого многоугольника; `Coords` -

переменная типа PointType, содержащая координаты вершин.

Иногда бывает целесообразным создав один элемент графической оболочки использовать его несколько раз (например однотипные кнопки). В этом случае могут быть полезными функции сохранения и выдачи изображений.

Функция ImageSize возвращает размер памяти в байтах, необходимый для размещения прямоугольного фрагмента изображения. Заголовок:

```
Function ImageSize(X1,Y1,X2,Y2: Integer): Word;
```

Здесь X1... Y2 - координаты левого верхнего и правого нижнего углов фрагмента изображения.

Процедура Getlmage помещает в память копию прямоугольного фрагмента изображения. Заголовок:

```
Procedure Getlmage(X1,Y1,X2,Y2: Integer; var Buf);
```

Здесь X1...Y2 - координаты углов фрагмента изображения; Buf - переменная или участок кучи, куда будет помещена копия видеопамяти с фрагментом изображения. Размер Buf должен быть не меньше значения, возвращаемого функцией ImageSize с теми же координатами X1....Y2.

Процедура PutImage выводит в заданное место экрана копию фрагмента изображения, ранее помещенную в память процедурой Getlmage. Заголовок:

```
Procedure Putlmage(X,Y: Integer; var Buf; Mode: Word);
```

Здесь X,Y- координаты левого верхнего угла того места на экране, куда будет скопирован фрагмент изображения; Buf - переменная или участок кучи, откуда берется изображение; Mode - способ копирования.

Как видно, координаты правого нижнего угла не указываются, так как они полностью определяются размерами вновь выводимой на экран копии изображения. Координаты левого верхнего угла могут быть какими угодно, лишь бы только выводимая копия уместилась в пределах экрана (если копия не может разместиться на экране, она не выводится и экран остается без изменений). Параметр Mode определяет способ взаимодействия вновь размещаемой копии с уже имеющимся на экране изображением. Взаимодействие осуществляется путем применения кодируемых этим параметром логических операций к каждому биту копии и изображения.

При выполнении данной занятия широко используются возможности текствого вывода. Описываемые ниже стандартные процедуры и функции поддерживают вывод текстовых сообщений в графическом режиме. Это не одно и то же, что использование процедур Write или WriteLn. Дело в том, что специально для графического режима разработаны процедуры, обеспечивающие вывод сообщений различными шрифтами в горизонтальном или вертикальном направлении, с изменением размеров и т.д. Однако в стандартных шрифтах, разработанных для этих целей фирмой Borland,

отсутствует кириллица, что исключает вывод русскоязычных сообщений.

С другой стороны, процедуры `Write` и `WriteLn` после загрузки в память второй половины таблицы знакогенератора (а эта операция легко реализуется в адаптерах EGA и VGA) способны выводить сообщения с использованием национального алфавита, но не обладают мощными возможностями специальных процедур.

Ниже описываются стандартные средства модуля Graph для вывода текста.

Процедура `OutText` выводит текстовую строку, начиная с текущего положения указателя. Заголовок:

```
Procedure OutText(Txt: String);
```

Здесь `Txt` - выводимая строка.

Процедура `OutTextXY` выводит строку, начиная с заданного места. Заголовок:

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

Здесь `X, Y` - координаты точки вывода; `Txt` - выводимая строка. Отличается от процедуры `OutText` только координатами вывода. Указатель не меняет своего положения.

Процедура `SetTextStyle` устанавливает стиль текстового вывода на графический экран. Заголовок:

```
Procedure SetTextStyle(Font,Direct,Size: Word);
```

Здесь `Font` - код (номер) шрифта; `Direct` - код направления; `Size` - код размера шрифта.

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

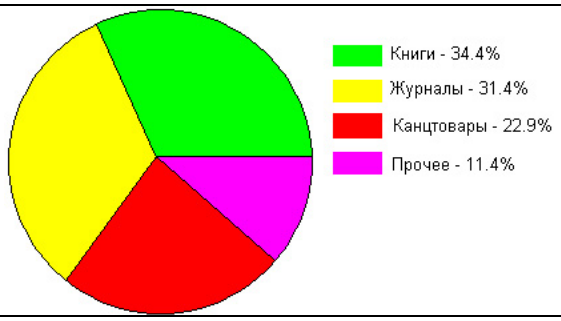
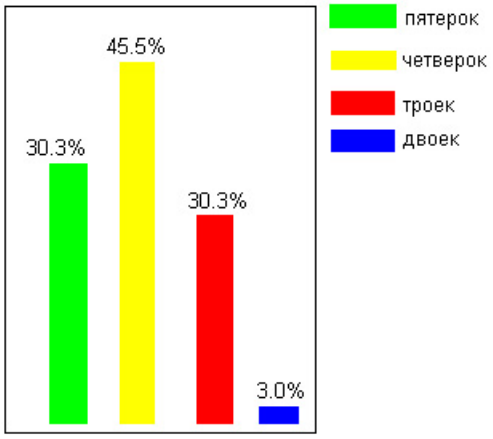
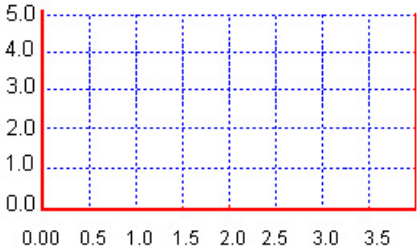
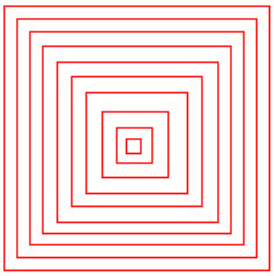
1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

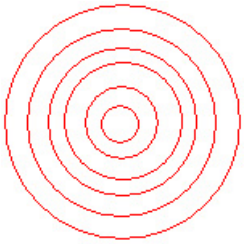
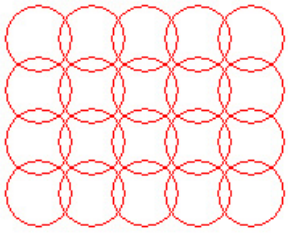
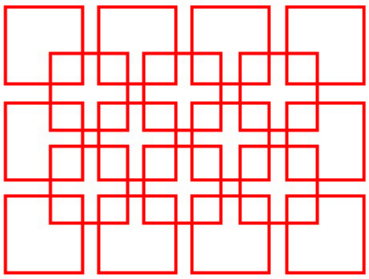
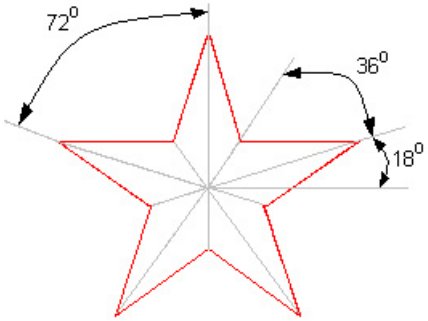
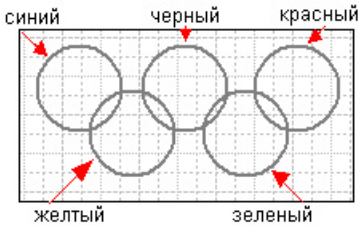
4. ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Заданием на практическое занятие является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий

№ варианта	Задание
1	Написать программу, которая выводит круговую диаграмму, отражающую товарооборот (в процентах) книжного магазина. Исходные данные (объем продаж в рублях по категориям: книги, журналы, открытки и канцтовары) вводятся во время занятия программы. Пример диаграммы приведен ниже.

	
2	Написать программу, которая выводит на экран гистограмму успеваемости студентов группы, например, по итогам контрольной занятия. Исходные данные следует ввести в алфавитно-цифровом режиме занятия. 
3	Написать программу, которая рисует окружность, движущуюся по экрану.
4	Написать программу, которая выводит на экран точечный график функции y = 0,5x2 + 4x — 3. Диапазон изменения аргумента — от —15 до 5, шаг аргумента — 0,1. График вывести на фоне координатных осей, точка пересечения которых должна находиться в центре экрана.
5	Написать программу, которая выводит на экран оцифрованную координатную сетку. 
6	Написать программу, которая выводит на экран узор, изображенный ниже. 

7	Написать программу, которая выводит на экран изображенный ниже узор. Окружности должны быть разного цвета (см. таблицу кодировки цветов). 
8	Написать программу, которая выводит изображенный ниже узор. 
9	Написать программу, которая выводит изображенный ниже узор. 
10	Написать программу, которая выводит на экран контур пятиконечной звезды. 
11	Написать программу, которая выводит на экран флаг Олимпийских игр. Изображение флага приведено ниже (одной клетке соответствует пять пикселов). 
12	Написать программу, которая выводит на экран изображение шахматной доски.

5. ПОРЯДОК ВЫПОЛНЕНИЯ ЗАНЯТИЯ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «prakt3.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите об инициализации графического модуля.
2) Расскажите об окнах и координатах.
3) Какие процедуры, регулирующие типы обводки заполнения вы знаете?
4) Какие процедуры, предназначенные для рисования геометрических фигур, вы знаете?
5) Какие существуют процедуры для вывода текста в графическом режиме?

Практическое занятие №4
ПРОЦЕДУРЫ И ФУНКЦИИ

1. ЦЕЛЬ И ЗАДАЧИ ЗАНЯТИЯ

Цель занятия: усвоение принципов модульного программирования.

Задачи занятия: изучить основные положения модульного программирования, получить навыки создания процедур и функций.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Основные принципы структурного программирования.

При разработке сложных программ используют так называемый структурный подход к программированию и нисходящее проектирование программ, когда сложная программ разбивается на более (или менее) функционально-законченные части, каждая из которых проще исходной программы. Такие программы легче отлаживать и использовать. Отдельные части программы называют подпрограммами. Использование подпрограмм наиболее эффективно в тех случаях, когда одна и та же подпрограмма может использоваться в программе не один раз, возможно с различными параметрами. Это позволяет экономить память компьютера.

Подпрограммы, в свою очередь, могут разбиваться на более мелкие части, реализуемые также в виде подпрограмм более низкого уровня.

В языке Турбо Паскаль используют подпрограммы двух типов: процедуры (Procedure) и функции (Function). Подпрограммы по структуре сходны с программой, но они обязательно имеют оригинальное имя, которое указывается в заголовке. Подпрограммы описываются в разделе описаний, использующих (вызывающих) их программ (или подпрограмм).

2. Процедуры.

Описание процедур в Паскале имеет вид:

```
Procedure   Имя процедуры (формальные параметры);
        Раздел  описаний
     Begin
     Раздел операторов
     End;
```

Формальные параметры вместе с круглыми скобками могут отсутствовать. Формальные параметры представляют собой список переменных с указанием их типа. Все типы, используемые в заголовках процедур и функций, кроме простых, должны быть описаны в подразделе Type вызывающей эти процедуры или функции программной единицы. Те параметры, которые изменяются в процедуре, называют выходными и перед ними в заголовке процедуры обязательно ставится слово Var. Параметры, имеющие файловый тип, должны быть обязательно описаны как Var - параметры и в процедурах и в функциях.

Вызов процедуры в использующих ее программных единицах (основной программе или подпрограммах) имеет следующий вид:

```
Имя процедуры (фактические параметры);
```

Фактические параметры могут отсутствовать вместе со скобками, в том случае, если нет формальных параметров в описании указанной процедуры.

Если параметры все же необходимы, то между фактическими и формальными параметрами должно быть установлено соответствие по их количеству, порядку следования и типу данных.

Имена фактических и формальных параметров могут быть как одинаковыми, так и различными.

Пусть в программе две процедуры P1 и P2 вызываются из основной программы. В свою очередь в процедуре P1 используется процедура P11 и она должна быть описана в разделе описаний вызывающей ее процедуры P1.

```
Раздел описаний основной программы
                Procedure P1;
                     Раздел описаний процедуры P1
…
Procedure P11
     Раздел описаний процедуры P11
     Begin
     Раздел операторов процедуры P11
     End;
Procedure P1
     Begin
     Раздел операторов процедуры P1
     End;
Procedure P2;
     Раздел описаний процедуры P2
     Begin
     Раздел операторов процедуры P2
     End;
BEGIN
…
Раздел операторов основной программы
…
END.
```

Имена, объявленные в разделе описаний основной программы, действуют в разделе операторов основной программы и в любой подпрограмме. Эти имена называются **глобальными**. Имена, объявленные в какой-либо подпрограмме, действуют в этой подпрограмме и в любой, объявленной в ней процедуре или функции. Такие имена называются **локальными**. Они недоступны для операторов основной программы. Область действия меток переходов в пределах каждой программной единицы своя. Нельзя перейти по оператору GOTO из одной процедуру в другую.

Рассмотрим **пример** разработки программы, содержащей две процедуры, каждая из которых используется дважды с различными фактическими параметрами.

Даны два массива M1 и M2, содержащие K1 и K2 целых чисел, соответственно. Определить максимальные числа в каждом из этих массивов, сравнить найденные значения между собой и вывести большее из них на экран.

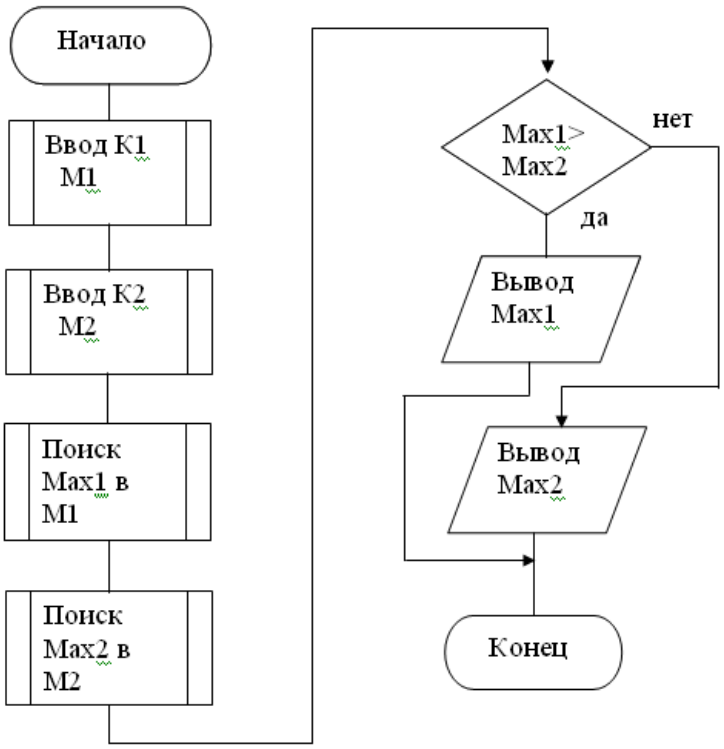


Рис. 1. Схема алгоритма.

Выделим глобальные переменные, которые используются в главной программе: М1, М2, К1, К2, Max1, Max2.

Текст программы:

```
Uses crt;
     Type Tmas=array[1..1000] of integer;
     Var M1, M2: Tmas;
         K1, K2, Max1, Max2 : integer;
{ Процедура ввода длины массива и самого массива }
Procedure Vvod(Var K:integer; Var M:Tmas);
     Var i:integer;
          Begin
               Write('  Введите длину массива');
               Readln(K);
          Writeln('  Введите элементы массива целых чисел, через
пробел');
          For i:=1 to K do
          Read(M[i]); readln;
End; { конец процедуры ввода}
{ Процедура поиска максимального элемента в массиве}
Procedure Poisk_max(K:integer; M:Tmas; Var Max:integer);
     Var i:integer;
          Begin
          Max:=M[1];      { За максимум принимаем первый элемент}
          For i:=2  to K do
          If M[i]>Max then  Max:=M[i]; {Запоминаем новый максимум}
End;
{  Начало основной программы}
Begin
     Clrscr;
```

```
     Writeln('  Ввод первого массива');
     Vvod(K1, M1);
     Writeln('  Ввод второго массива');
     Vvod(K2,M2);
     Poisk_max(K1, M1, Max1);
     Poisk_max(K2, M2, Max2);
     If Max1>Max2 then writeln(' Max1 больше и оно =  ',Max1)
          Else writeln(' Max2 больше и оно =  ',Max2);
     Readkey; { Останов для просмотра результатов}
End.
```

3. Функции.

Подпрограммы - функции (Function) имеют следующие отличительные особенности (по сравнению с процедурами):

1) Функция имеет только один результат выполнения. Этот результат обозначается именем функции и передается в вызвавщую эту функцию программную единицу.
2) Для функции обязательно указывается ее тип. Тип может быть простым (скалярным) или строковым.

Описание функции в Паскале имеет вид:

```
Function   Имя функции (формальные параметры):Тип результата;
          Раздел  описаний
Begin
Раздел операторов
End;
```

Вызов функции производится по ее имени с указанием фактических параметров. Аналогично процедурам, фактические и формальные параметры в функции могут отсутствовать.

Правила использования фактических и формальных параметров, а также локальных и глобальных переменных, совпадают с использованием их в процедурах.

Если функция кроме выдачи своего значения меняет значения каких-либо глобальных переменных, то говорят, что она имеет побочный эффект.

Рассмотрим **пример** создания программ с использованием функций. Пусть необходимо произвести следующие вычисления:

$$S = \frac{\sum_{i=1}^{n} x_i + \sum_{j=1}^{m} y_j}{a^n + b^m},$$

где a, b, n, m - целые переменные;

$\{x_i\}$, $\{y_j\}$ - массивы, содержащие n и m вещественных чисел, соответственно.

В Паскале нет стандартных функций сумммирования элементов массива и возведения чисел в степень больше 2. Разработаем свои функции для

решения этих задач и будем использовать их для решения поставленной задачи.

Текст программы

```
Uses crt;
     Type Tmas=array[1..100] of real;
     Var  a,b,n,m,i,j:byte;
          X,Y:Tmas;
          S:Real;
Function Summa(Dlmas:byte; Mas:Tmas):Real;
     Var Sum:real;  i:byte;
     Begin
     Sum:=0.0;
     For i:=1 to Dlmas do
     Sum:=Sum+Mas[i];
     Summa:=Sum;
End;
Function step(Pok:byte;Osn:byte):real;
     Var i:byte; St:real;
     Begin
     St:=Osn;
     For i:=2 to Pok do
     St:=St*Osn;
     step:=St;
End;
{ Главная программа}
BEGIN
     Write('  Введите длину первого массива - N  ');
     Readln(N);
     Write('  Введите длину второго массива - M  ');
     Readln(M);
     Writeln('   Введите элементы массива X');
               For i:=1 to N do Read(X[i]); readln;
     Writeln('   Введите элементы массива Y');
               For j:=1 to M do Read(Y[j]); readln;
     Write('   Введите a и b ');  Readln(a,b);
     S:=(Summa(N,X)+Summs(M,Y))/(step(N,a)+step(M,b));
     Writeln('  Полученный результат  S = ',S);readkey;
END.
```

4. Рекурсии.

Рекурсия - это такой способ организации вычислительного процесса, при котором подпрограмма обращается сама к себе. Такая рекурсия называется прямой. Рекурсии позволяют писать более короткие программы, но при выполнении они работают медленнее и могут вызвать переполнение стека, так как при каждом входе в подпрограмму ее локальные переменные размещаются в программном стеке, размер которого ограничен.

Типичным **<u>примером</u>** прямой рекурсии является вычисление n!.

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

Текст программы с использованием прямой рекурсии для вычисления $n!$.

```
Var n:integer;
```

```
{Рекурсивная функция}
Function Fact(n:integer):real;
     Begin
     If n=0 then Fact:=1
     Else Fact:=n*Fact(n-1);
End;
{  Главная программа}
Begin
     Repeat
     Writeln(' Введите положительное n<=33');
     Readln(n);
     Writeln(Fact(n));
     Until Eof;
  End.
```

Примечание: для выхода из этой программы можно задать значение n>33 или нажать Ctrl/z и Enter. При n>33 возникает переполнение при умножение чисел с плавающей запятой.

Рекурсивный вызов может быть также косвенным. В этом случае подпрограмма обращается к себе опосредованно, путем вызова другой подпрограммы, в которой, в свою очередь, содержится обращение к первой. Такой вызов иногда называют закольцованным.

Для реализации такой возможности в ТР используется опережающее описание процедур и функций и директива Forward.

Для этого в самом начале программы вставляют только заголовки процедуры или функции в виде:

```
Procedure  имя-процедуры(параметры);Forward;
```

или

```
Function имя_функции(параметры);Forward;
```

Позже, в необходимых местах, описываются сами процедуры или функции в обычном виде и в их заголовке параметры уже указывать не нужно.

Например: процедура с именем A вызывает процедуру с именем B, а процедура B, в свою очередь, вызывает процедуру A.

```
Procedure A(y:тип);Forward;  {Опережающее описание процедуры A}
Procedure B(x:тип);  { Заголовок процедуры B}
     . . .   {Раздел описаний процедуры B}
     begin
     . . .
     A(p);  {Вызов процедуры A из B}
     . . .
end;
Procedure A;  {Основное описание процедуры A}
     . . .                  {Раздел описаний процедуры A}
     Begin
     . . .
     B(g);  {Вызов процедуры B из A}
     . . .
End;
```

3. ОБОРУДОВАНИЕ И МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Компьютерные классы, оборудованные ПК с установленным программным обеспечением не ниже:

1) Операционная система Windows XP;
2) Microsoft Калькулятор Плюс;
3) Turbo Pascal 7.0 или ABC Pascal.

4. ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Заданием на практическое занятие является написание программы, реализующей алгоритм решения поставленной задач на языке Turbo Pascal.

Варианты заданий

№ вар.	Функция
1	$F = \sum_{i=1}^{t}(x_i + a)^n + \sum_{j=1}^{m}\frac{y_j}{c^m}$
2	$F = \frac{(a+b)^n}{\sum_{j=1}^{m}((a+b)^m - y_j)}$
3	$F = \sum_{i=1}^{n} x_i + a^n + \sum_{k=1}^{m}\frac{y_k}{ac^m}$
4	$F = \frac{(a^m - b^n)}{\sum_{j=1}^{m}((a-b)^m - y_j)}$
5	$F = \frac{\sum_{i=1}^{t}(x_i - c^t)}{c^{t+2} \cdot (t-2)!}$
6	$F = \frac{\sum_{i=1}^{k}(x_i - c)^k}{c! - (k-c)!}$
7	$F = \sqrt{(x^n + y^m)} + \frac{(n+m)!}{\sum_{k=1}^{t} z_k - \sum_{j=1}^{p} s_j}$
8	$F = \frac{\sum_{i=1}^{s}(x_i - y_i)^s + s!}{\sum_{k=1}^{n}(a_k - b_k)^n - n!}$
9	$F = \frac{\sqrt{a^k - b^t}}{\sum_{i=1}^{k}(x_i + b)} + \sum_{i=1}^{t}(y_i + a)$
10	$F = \frac{\sum_{i=1}^{n}(x_i - y_i)^n - (n+3)!}{\sum_{k=1}^{n}(a_k - b_k)^n - n!}$
11	$F = \frac{\sqrt{a^{k+2} + b^{t-2}}}{\sum_{i=1}^{t}(a^k x_i - b)} + \sum_{i=1}^{k}(b^t y_i - a)$

12	$F = \frac{\sqrt{a^k + b^t}}{\sum_{i=1}^{k} x_i} + \sum_{i=1}^{t} y_i$

5. ПОРЯДОК ВЫПОЛНЕНИЯ ЗАНЯТИЯ

1) Прочитать задание и выписать все переменные, которые следует использовать при его решении.
2) Продумать порядок решения задачи и составить алгоритм ее решения.
3) Составить список операторов, которые понадобятся при написании программы.
4) Описать в разделе описаний все используемые переменные и их типы.
5) Составить программу решения задачи.
6) Проверить правильность программы путем введения данных, результат решения для которых известен.
7) Если необходимо, отладить программу.
8) Оформить ввод и вывод данных удобочитаемым образом.
9) Сохранить результат, как «prakt4.pas»

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1) Расскажите о модульном программировании.
2) Что такое функция?
3) Что такое процедура?
4) В чем принципиальное отличие функции и процедуры?
5) Что такое рекурсия? Приведите пример использования рекурсии.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Основная литература

1. Мурат Е.П. Информатика III : учебное пособие / Мурат Е.П.. — Ростов-на-Дону, Таган-рог : Издательство Южного федерального университета, 2018. — 150 с. — ISBN 978-5-9275-2689-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87415.html

2. Старыгина С.Д. Информатика: технологии и офисное программирование : учебное по-собие / Старыгина С.Д., Нуриев Н.К., Нургалиева А.А.. — Казань : Казанский национальный ис-следовательский технологический университет, 2018. — 232 с. — ISBN 978-5-7882-2565-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/100670.html.

3. Оболонин И.А. Основы компьютерного проектирования в инфокоммуникационных тех-нологиях : учебно-методическое пособие / Оболонин И.А.. — Новосибирск : Сибирский госу-дарственный университет телекоммуникаций и информатики, 2018. — 250 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/84070.html.

Дополнительная литература

1) Курносов, А.П. Информатика: учеб. пособие для вузов / А.П. Курносов [и др.]; под ред. А.П. Курносова. — М.: КолосС, 2005 .— 272 с.

2) Королев, Л.Н. Информатика: Введение в компьютерные науки: Учебник для вузов / Л.Н. Королев, А.И. Миков .— М.: Высш. шк., 2003 .— 314 с.

3) Симонович, С.В. Информатика: Базовый курс: Учеб. пособие для вузов / Под ред. С.В. Симоновича. — СПб. и др. : Питер, 2003 .— 640 с.

Периодические издания

1) Компьютерра: компьютерный еженедельник. — М.: ООО Журнал "Компьютерра".

2) Мир ПК: журнал для пользователей персональных компьютеров. — М.: Открытые системы.

Интернет-ресурсы

1) http://cictemnik.ru/ - сайт, посвященный современным компьютерным системам;

2) http://www.computerra.ru/ - сайт «Компьютера онлайн»