МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательноеучреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра биомедицинской инженерии

Медицинская информатика – лабораторный практикум (Часть 2л)

Методические рекомендации по выполнению лабораторных работ для студентов специальности 30.05.03 «Медицинская кибернетика»

Курск 2016

УДК 004

Составители: М.В. Артеменко, Скопин Д.Е., Шаталова О.В.

Рецензент
Кандидат технических наук, доцент кафедрыпрограммной инженерии ЮЗГУ Белов В.Г.

Медицинская информатика – лабораторный практикум: методические указания выполнения лабораторных работ/ Юго-Зап. гос. ун-т; сост. М.В. Артеменко, Д.Е.Скопин, О.В. Шаталова − Курск, 2016. − 137 с.: ил.5, - Библиография

Методические указания: содержат краткие теоретические сведения, порядок выполнения и содержание отчета по лабораторным работам по дисциплине «Информатика, медицинская информатика», касающиеся приобретение знаний и навыков студентами в части: программирования на языках высокго уровня, работы в медицинских информационных системах и базах данных, математического имитационного моделирования, обработки символьной информации, работе в электронных таблицах. Рассмотьрены примеры решения различных задач на языках высого уровня: Паскаль, Бейсик, Java. Содержание методических указаний соответствуют требованиям Федерального государственного образовательного стандарта высшего образования подготовки специалистов 30.05.03 «Медицинская кибернетика», утвержденной учебно-методическим объединением.

Текст печатается в авторской редакции

Подписано в печать Формат 60х84х 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 100 экз. Заказ. Бесплатно.
Юго-Западный государственный университет.
305040, г.Курск, ул. 50 лет Октября,9 4

Содержание

ЛАБОРАТОРНАЯ РАБОТА №1. ОБРАБОТКА ЧИСЛЕННЫХ МАССИВОВ

Краткие теоретические сведения

А. Обработка массивов с помощью языков высокого уровня на примере языка Паскаль.

Массивом называют упорядоченный набор однотипных переменных (элементов). Каждый элемент имеет целочисленный порядковый номер, называемый индексом. Число элементов в массиве называют его размерностью. Массивы используются там, где нужно обработать сразу несколько переменных одного типа: например, оценки студентов группы или координаты точек на плоскости. Строку текста можно рассматривать как массив символов, а текст на странице как массив строк.

Массив описывается в разделе var оператором следующего вида:

```
var ИмяМассива: array [НИ .. ВИ] of Тип;
```

Здесь

НИ (нижний индекс) - целочисленный номер 1-го элемента массива;.. - оператор задания диапазона в Паскале;ВИ (верхний индекс) -- целочисленный номер последнего элемента;

Тип - любой из известных типов данных Паскаля. Каждый элемент массива будет рассматриваться как переменная соответствующего типа.

Опишем несколько массивов разного назначения.

```
var a: array [1..20] of integer;
```

Здесь описан массив с именем A, состоящий из 20 целочисленных элементов;

```
var x,y : array [1..10] of real;
```

Описаны 2 массива с именами x и y, содержащие по 10 вещественных элементов;

```
var t : array [0..9] of string;
```

Массив t состоит из 10 строк, которые занумерованы с нуля.
Размерность (число элементов) массива вычисляется как ВИ - НИ + 1.

Для обращения к отдельному элементу массива используется оператор вида ИмяМассива [Индекс].

Здесь Индекс - целочисленный номер элемента (может быть целочисленным выражением или константой). Индекс не должен быть меньше значения нижнего или больше верхнего индекса массива, иначе возникнет ошибка "Constantoutofrange". Отдельный элемент массива можно использовать так же, как переменную соответствующего типа, например:

```
A[1]:=1;x[1]:=1.5; y[1]:=x[1]+1;t[0]:='Hello';
```

В одномерных массивах каждый элемент имеет один номер (индекс), характеризующий его положение в массиве. В математике понятию одномерного массива из n элементов соответствует понятие векториз n компонент: A = {Ai}, i=1, 2 ,..., n.

Как правило, ввод, обработка и вывод массива осуществляются поэлементно, с использованием цикла for.

Простейший способ ввода – это ввод массива с клавиатуры, например:

```
const n = 10;
var a: array [1..n] of real; i:integer;
beginwriteln ('Введите элементы массива');
for i:=1 to n do read (A[i]);
```

Размерность массива определена константой n, элементы вводятся по одному в цикле for - при запуске этой программы пользователю придется ввести 10 числовых значений. При решении ряда задач вводить массивы "вручную", особенно если их размерность велика, не всегда удобно. Существуют, как минимум, два альтернативных решения.

Описание массива констант удобно, если элементы массива не должны изменяться в процессе выполнения программы. Как и другие константы, массивы констант описываются в разделе const. Приведем пример такого описания:

```
consta:array [1..5] ofreal=( 3.5, 2, -5, 4, 11.7);
```

Как видно из примера, элементы массива перечисляются в круглых скобках через запятую, их количество должно соответствовать его размерности.

Формирование массива из случайных значений уместно, если при решении задачи массив служит лишь для иллюстрации того или иного алгоритма, а конкретные значения элементов несущественны. Для того чтобы получить очередное случайное значение, используется стандартная функция random(N), где параметром N передается значение порядкового типа. Она вернет случайное число того же типа, что тип аргумента и лежащее в диапазоне от 0 до N-1 включительно. Например, оператор вида a[1]:=random(100); запишет в a[1] случайное число из диапазона [0,99].

Для того чтобы при каждом запуске программы цепочка случайных чисел была новой, перед первым вызовом random следует вызвать стандартную процедуру randomize;, запускающую генератор случайных чисел. Приведем пример заполнения массива из 20 элементов случайными числами, лежащими в диапазоне от -10 до 10:

```
var a:array [1..20] of integer;
 i:integer;
begin
 randomize; for i:=1 to 20 do begin  a[i]:=random(21)-10;  write (a[i]:4);
 end;
end.
```

Еще более удобный путь - чтение элементов массива из текстового или численногоо файла. К массивам применимы все типовые алгоритмы, изученные в теме "Циклы". Приведем один пример, в котором вычисляется сумма s положительных элементов массива.

```
var b:array [1..5] of real;
s:real; i:integer;
begin
writeln ('Введите 5 элементовмассива');for i:=1 to 5 do read (b[i]);
s:=0;for i:=1 to 5 do if b[i]>0 then s:=s+b[i];
```

Вывод массива на экран также делается с помощью цикла for.

```
for i:=1 to 5 do write (b[i]:6:2);
```

Здесь 5 элементов массива b напечатаны в одну строку, каждый элемент состоит из 6 символов и имеет 2 десятичных разряда после запятой. Для вывода одного элемента на одной строке можно было бы использовать оператор writeln вместо write.

Существенно то, что если обработка массива осуществляется последовательно, по 1 элементу, циклы ввода и обработки зачастую можно объединить, как в следующем примере.

Найти арифметическое среднее элементов вещественного массива t размерностью 6 и значение его минимального элемента.

```
varb:array [1..6] ofreal; s, min:real; i:integer;
begin
s:=0; min:=1e+30;writeln ('Ввод B[6]');
for i:=1 to 6 do begin read (b[i]); s:=s+b[i]; if b[i]<min then min := b[i];end;
writeln ('min=',min,' s=', s/6);
end.
```

Теоретически в этой программе можно было бы обойтись и без массива - ведь элементы b[i] используются только для накопления суммы и поиска максимума, так что описание массива вполне можно было заменить описанием вещественной переменной b. Однако, в реальных задачах данные, как правило, обрабатываются неоднократно и без массивов обойтись трудно. Приведем пример учебной задачи, где использование массива дает выигрыш за счет уменьшения объема вычислений, выполняемых программой.

Задана последовательность Ti = max {sin(i), cos(i)}, i= -5, -4, ..., 5. Найти элемент последовательности, имеющий минимальное отклонение от арифметического среднего положительных элементов.

Здесь в первом цикле можно сформировать массив по заданному правилу и найти арифметическое среднее положительных элементов. Во втором цикле, когда среднее известно, можно искать отклонение. Без использования массива необходимо было бы считать элементы последовательности дважды.

```
var t : array [-5..5] of real; i,k:integer; s, ot : real;
begin
 s:=0; k:=0;
 for i:=-5 to 5 do begin
  t[i]:=sin(i);  if t[i]<cos(i) then t[i]:=cos(i);
  if t[i]>0 then begin   k:=k+1; s:=s+t[i];  end;
 end;
 s:=s/k; ot:=1e30;
 for i:=-5 to 5 do begin
```

```
 if abs(t[i]-s)<ot then ot:= abs(t[i]-s);
end;
writeln ('Ot=', ot:8:2);
end.
```

Распространена обработка в одной задаче сразу нескольких массивов. Приведем пример.

Координаты 10 точек на плоскости заданы массивами x={xi}, y={yi}, i=1, 2, ..., 10. Найти длину ломаной, проходящей через точки (x1, y1), (x2, y2), ..., (x10, y10), а также номер точки, лежащей дальше всего от начала координат.

При решении задачи используем формулу для нахождения расстояния между 2 точками на плоскости, заданными координатами (x1,y1) и (x2,y2): $r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Обозначим через r расстояние между текущей точкой и следующей, Len искомую длину ломаной, Dist -- расстояние от текущей точки до начала координат, max -- максимальное из этих расстояний, Num -- искомый номер точки.

```
var x,y : array [1..10] of real;
I, num:integer;
r, Len, Dist, max : real;
begin {найдем max при вводе данных}
 max:=0; {т.к. расстояние не м.б. <0 }
 num:=1; {на случай, если все точки (0,0)}
 writeln ('Введитекоординаты 10 точек');
 for i:=1 to 10 do begin
  read (x[i], y[i]);  Dist:=sqrt (sqr(x[i]) + sqr (y[i]));
  if dist > max then begin    max:=dist; {запомнилиновоерасстояние}
    Num:=i;    {и номер точки}
  end;
end;
writeln ('Номерточки=',num, ' расстояние=',dist:8:2);
Len:=0;{длина ломаной - сумма длин сторон}
for i:=1 to 9 do begin
       {у 10-йточкинетследующей!}
  r:= sqrt (sqr(x[i]-x[i+1])+sqr(y[i]-y[i+1]));   Len:=Len+r;
end;
writeln ('Длиналоманой=',len:8:2);
```

```
end.
```

Приведем пример задачи формирования массива по правилу.

Задан массив x из 8 элементов. Сформировать массив y по правилу

$$y_i = \begin{cases} 4x_i, & \text{если } i - \text{четное} \\ \cos(2x_i), & \text{если } i - \text{нечетное} \end{cases}$$

и найти количество его положительных элементов.

```
var x,y: array [1..8] of real; i,k:integer;
begin
 writeln ('Введите массив x из 8 эл.');
for i:=1 to 8 do begin  read (x[i]);
if i mod 2 =0 then y[i]:=4*x[i]
elsey[i]:=cos(2*x[i]);
end;
K:=0;writeln ('Массив y');
 for i:=1 to 8 do begin    if y[i]>0 then k:=k+1;   write (y[i]:8:2); end;
 writeln;
 writeln ('K=',k);
end.
```

Пример обработки многомерного массива

Как правило, при обработке многомерных массивов используются вложенные циклы, т.е. цикл по столбцам располагается внутри цикла по строкам.

Дана матрица A(3,4), и вектор B (4), состоящие из целых чисел . Умножитьматрицу AнавекторB .

```
program  pr4-2 ;
const m=3;   n=4;
var
a  : array [ 1 .. m, 1 .. n ] of    integer; (* описаниематрицы *)
b  : array [ 1 .. n ] of   integer;                  (* описаниевектора *)
c  : array [ 1 .. m ]  of  integer;                   ( * описание  C  *)
i,  j: integer;
begin
for i:=1 to m do                (* вводматрицы  *)
begin writeln (‘введитеэлементы ‘, i , ‘-тойстроки’);
for j:=1 to n do read (a [i, j] ); writeln;
end;
```

```
writeln (‘введитеэлементывектора’);
for j:=1 to n  do                        (* ввод  вектора *)
read (b[ j]);   writeln;
for i:=1 to m dobegin
c [ i ]:=0;        for j:=1 to n do       c[i] := c[ i ]+ a[i , j]* b[j];
end;
for i:=1 to m do      (*форматныйвыводматрицы  *)begin
for j:=1 to n do write (a [i, j]: 4 );  writeln;
end;
for j:=1 to n do  write (b [ j ] :4);    (* вывод  массива B *)
writeln ;
for i:=1 to m do  write (c [ i ] :4);   (* вывод  массиваC  *)
readln;
end.
```

В программе элементы матрицы вводятся по строкам по одному с подтверждением клавишей Enter. А выводятся в общепринятом виде: каждая строка матрицы с новой строки экрана (цикл i по строкам внешний, а цикл j – внутренний)

Б. Обработка массивов с помощью языков высокого уровня на примере языка Java.

Язык Java является на сегодняшний момент самым распространенным языком в мире, поскольку на нем создаются как программные продукты всемирной сети Интернет, так и мобильные приложения самой распространенной операционной системы Android. Язык Java создана на основе C++, к небольшим отличиям между этими языками следует отнести работу с массивами и отсутствие указателей в языке Java.

В отличае от языка Паскаль, массив Java создается в любом месте программы с использованием следующей конструкции:

Тип[] ИмяМассива=new Тип [количество элементов];

Здесь

Тип - любой из известных типов данных языка Java. Каждый элемент массива будет рассматриваться как переменная соответствующего типа.

new - оператор создания блока памяти. Так как массивы в языке Java динамические, при создании массива выделяется область памяти за пределами основной программы. Это позволяет создавать массивы любой длинны, даже размером сопоставимым с физической памятью компьютера.

Количество элементов - описывает общее количество элементов, которые будут созданны. При этом, как и в языке C++ нижний индекс массива всегда равен нулю, а верхний всегда на единицу меньше количества элементов в массиве. То есть, при создании массива, содержащего n элементов, индексы этого массива всегда будут в диапазоне 0…n-1.

Опишем несколько массивов разного назначения.

```
var a: array [1..20] of integer;
int[] a=newint[20];
```

Здесь описан массив с именем A, состоящий из 20 целочисленных элементов; Индексы этих элементов начинаются с 0 и заканчиваются числом 19.

```
var x,y : array [1..10] of real;
float[]  x=new float[10];
double[] y=new double[4];
```

Описаны 2 массива с именами x и y. Массив x содержит 10 вещественных элементов, массив y содержит 4 вещественных числа с двойной точностью.

```
String[] t=newString[10];
```

Массив t состоит из 10 строк, которые занумерованны с нуля.

Для обращения к отдельному элементу массива используется оператор вида ИмяМассива [Индекс].

Здесь Индекс - целочисленный номер элемента (может быть целочисленным выражением или константой). Индекс в языке Java не должен быть меньше нуля и не должен быть больше значения n-1, где n является количеством элементов в массиве, иначе возникнет ошибка времени выполнения. Отдельный элемент массива можно использовать так же, как переменную соответствующего типа, например:

```
a[1]=1; x[1]=1.5; y[1]=x[1]+1; t[0]="Hello";
```

Помните, что во всех современных языках программирования (C++, C#, Java) имена чувствительны к регистру. Это означает, что a[1] и A[1] являются разными массивами.

Как и в языке Паскаль, ввод, обработка и вывод массива осуществляются поэлементно, с использованием любого цикла:for, while, do…while. При этом, так как массивы в языке Java динамические, можно задать размер массива в процессе выполнения программы. Приведем пример ввода массива произвольной длинны с клавиатуры и печать его элементов:

```
Scanner input=new Scanner(System.in);
intn;
System.out.println("Введите количество элементов");
n=input.nextInt ();
int[] a=newint[n];
for (inti=0;i<n;i++)
{System.out.println ("Введитеэлементномер "+(i+1));
a[i]=input.nextInt();
}
```

В данном примере в первой строке создается объект с именем input для чтения данных с клавиатуры компьютера. Далее создается целочисленная переменная n. У пользователя спрашивается о требуемом количестве элементов в массиве и после ввода числа n создается массив требуемой длинны. Далее, с использованием цикла for поэлементно заполняется массив, при этом ввод элементов выполняется с использованием созданного выше объекта input в строчке input.nextInt().

При решении ряда задач вводить массивы "вручную", особенно если их размерность велика, не всегда удобно. Существуют, как минимум, два альтернативных решения.

Описание массива с заданными начальными элементами производится следующим образом:

```
float[] m={ 3.5, 2, -5, 4, 11.7 };
```

Как видно из примера, элементы массива перечисляются в фигурных скобках через запятую, их количество определяется общим количеством чисел в ряду (в данном случае 4), а индексы массива находятся в диапазоне 0…3.

Формирование массива из случайных значений уместно, если при решении задачи массив служит лишь для иллюстрации того или иного алгоритма, а конкретные значения элементов несущественны. Для того чтобы получить очередное случайное значение, используется стандартная функцияrandom() класса Math вызов которой осуществляется следующим образом: Math.random(). Данная функция возвращает число типа double в диапазоне 0…1. Например, оператор вида a[1]=Math.random(); запишет в a[1] случайное число из диапазона [0,1]. Если вам понадобится числов диапазоне 0..100 целого типа, то его можно получить вот так: a[1]=(int)(Math.random()*100); В этом примере иллюстрируется преобразование типов, при котором новый, требуемый тип заключается в круглые скобки, как в примере (int). Генерируемое функцией random() число преобразуется в целый тип и будет присвоено ячейке массива a[1].

Следует добавить, что генератор псевдослучайных чисел языка Java привязан к счетчику реального времени (микросхеме таймера компютера), таким образом смена основания генератора не требуется.

Приведем пример заполнения массива из 20 элементов случайными числами, лежащими в диапазоне от -10 до 10:

```
int[] a=newint[20];
        for (inti=0;i<20;i++)
        {
            a[i]=(int)(Math.random()*21-10);
            System.out.print(a[i]+" ");
        }
```

Еще более удобный путь - чтение элементов массива из текстового или численногоо файла. К массивам применимы все типовые алгоритмы, изученные в теме "Циклы". Приведем один пример, в котором вычисляется сумма s положительных элементов массива.

```
        Scanner input=new Scanner(System.in);
        int[] b=newint[5];
        floats;
        System.out.println("Введите 5 элементовмассива"); for
(inti=0;i<5;i++) b[i]=input.nextInt();
        s=0; for (inti=0;i<5;i++) if (b[i]>0) s=s+b[i];
```

```
//Выводмассиванаэкрантакжеделается с помощьюцикла for.
for (inti=0;i<5;i++) System.out.println(b[i]+" ");
System.out.println("Сумма положительных элементов: "+s);
```

Здесь 5 элементов массива b напечатаны в одну строку с разделителем в виде пробела. Для вывода одного элемента на одной строке можно было бы использовать функцию println() вместо print().

Существенно то, что если обработка массива осуществляется последовательно, по 1 элементу, циклы ввода и обработки зачастую можно объединить, как в следующем примере.

Найти арифметическое среднее элементов вещественного массива t размерностью 6 и значение его минимального элемента.

```
float[] b=newfloat[6];
floats=0,min=999999;
System.out.println("ВводB[6]");
for (inti=0;i<6;i++)
        {b[i]=input.nextInt();
        s=s+b[i]; if( b[i]<min) min=b[i];
        }
System.out.println("min= "+min+" s="+s/6.0);
```

Обратите внимание на выражение s/6.0. Язык Java, так же как и языки C++ и C# являются "более низкоуровневыми", в сравнении с языком Паскаль. В результате, при вычислении частного, если оба аргумента являются целыми числами, микропроцессор при делении использует целочисленную арифметику, потеряв дробную часть. Чтобы этого не происходило деление производят не нацелое число, а на дробное, которым, в частности, является число 6.0.

Теоретически в этой программе можно было бы обойтись и без массива - ведь элементы b[i] используются только для накопления суммы и поиска максимума, так что описание массива вполне можно было заменить описанием вещественной переменной b. Однако, в реальных задачах данные, как правило, обрабатываются неоднократно и без массивов обойтись трудно. Приведем пример учебной задачи, где использование массива дает выигрыш за счет уменьшения объема вычислений, выполняемых программой.

Задана последовательность Ti = max {sin(i), cos(i)}, i= -5, -4, ..., 5. Найти элемент последовательности, имеющий минимальное отклонение от арифметического среднего положительных элементов.

Здесь в первом цикле можно сформировать массив по заданному правилу и найти арифметическое среднее положительных элементов. Во втором цикле, когда среднее известно, можно искать отклонение. Без использования массива необходимо было бы считать элементы последовательности дважды.

```
float[] t=newfloat[10];
intk=0;
floats=0;
intcount=0;
for (inti=-5;i<5;i++)
{t[count]=(float)Math.sin(i);
if (t[count]<Math.cos(i)) t[count]=(float)Math.cos(i);
   if (t[count]>0){k++;s=s+t[count];};
    count++;
}
s=s/k;count=0;
floatot=1e30f;
for (inti=-5;i<5;i++) if (Math.abs(t[count]-s)<ot) {ot=Math.abs(t[count]-
s);count++;};
System.out.println("ot="+ot);
```

Распространена обработка в одной задаче сразу нескольких массивов. Приведем пример.

Координаты 10 точек на плоскости заданы массивами x={xi}, y={yi}, i=1, 2, ..., 10. Найти длину ломаной, проходящей через точки (x1, y1), (x2, y2), ..., (x10, y10), а также номер точки, лежащей дальше всего от начала координат.

При решении задачи используем формулу для нахождения расстояния между 2 точками на плоскости, заданными координатами (x1,y1) и (x2,y2): $r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Обозначим через r расстояние между текущей точкой и следующей, Len искомую длину ломаной, Dist -- расстояние от

текущей точки до начала координат, max -- максимальное из этих расстояний, Num -- искомый номер точки.

```
float[] x=newfloat[10],y=newfloat[10];
inti,num;
floatr,len,dist=0,max;
//найдем max привводеданных
max=0;//т.к. расстояниене м.б. <0
num=1; //наслучай, есливсеточки (0,0)
System.out.println("Введите координаты 10 точек");
for (i=0;i<10;i++)
 {x[i]=input.nextFloat();y[i]=input.nextFloat();
 dist=(float)Math.sqrt (x[i]*x[i] + y[i]*y[i]);
if (dist>max){  max=dist; //запомнилиновоерасстояние
num=i;    //и номерточки
            }
}
System.out.println("Номер точки="+num+" расстояние="+dist);
len=0;//длиналоманой - суммадлинсторон
for(i=0;i<9;i++)
{//у 10-й точкинетследующей!}
r= (float)Math.sqrt (Math.pow(x[i]-x[i+1],2)+ Math.pow(y[i]-y[i+1],2));len=len+r;
}
System.out.println("Длиналоманой="+len);
```

Приведем пример задачи формирования массива по правилу.

Задан массив x из 8 элементов. Сформировать массив y по правилу

$$y_i = \begin{cases} 4x_i, & \text{если } i - \text{четное} \\ \cos(2x_i), & \text{если } i - \text{нечетное} \end{cases}$$

и найти количество его положительных элементов.

```
Scanner input=new Scanner(System.in);
float[] x=newfloat[8];
float[] y=newfloat[8];
intk;
System.out.println("Введитемассив x из 8 эл.");
for (inti=0;i<8;i++)
{ x[i]=input.nextInt();
if  (i%2==0) y[i]=4*x[i];
elsey[i]=(float)Math.cos(2*x[i]);
```

```
}
k=0; System.out.println("Массив y");
for (inti=0;i<8;i++){    if (y[i]>0)  k=k+1;System.out.println(y[i]); }
System.out.println("K="+k);
```

Пример обработки многомерного массива

Как правило, при обработке многомерных массивов используются вложенные циклы, т.е. цикл по столбцам располагается внутри цикла по строкам.

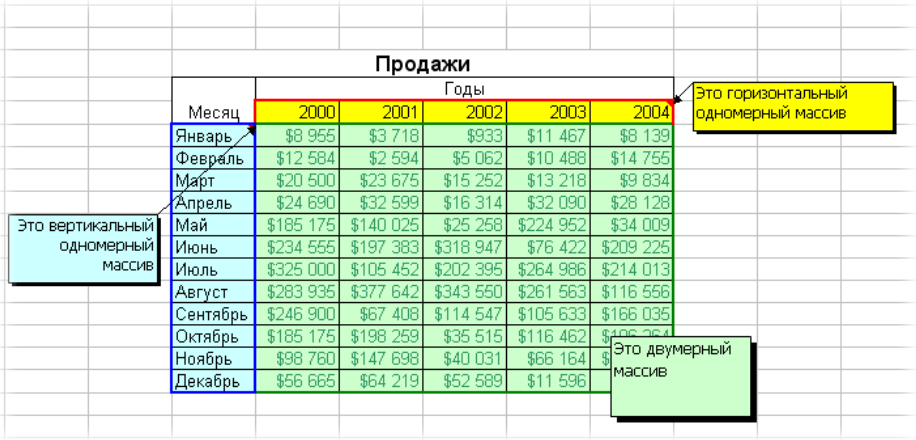
Дана матрица A(3,4), и вектор B (4), состоящие из целых чисел . Умножить матрицу A на вектор B .

```
Scannerinput=newScanner(System.in);
intm=3, n=4;
int[][] a =newint[m][n];// описаниематрицы
int[] b=newint[n];//описаниевектора
int[] c=newint[m];//описаниеC
inti,  j;

for (i=0;i<m;i++)//вводматрицы
{System.out.println("введите элементы "+ i+" -той строки");
for (j=0;j<n;j++) a[i][j]=input.nextInt();
}
System.out.println("введите элементы вектора");
for (j=0;j<n;j++) b[j]=input.nextInt(); // вводвектора

for(i=0;i<m;i++)
{c[i]=0;for(j=0;j<n;j++) c[i]=c[i]+a[i][j]*b[j];
}
for(i=0;i<m;i++)
{for (j=0;j<n;j++)System.out.println(a[i][j]);  System.out.println();
}
for (j=0; j<n;j++) System.out.println (b[j]+"    ");//выводмассива B
System.out.println();
for(i=0;i<m;i++)System.out.println(c[i]+"    ");// выводмассиваC
```

В программе элементы матрицы вводятся по строкам по одному с подтверждением клавишей Enter. А выводятся в общепринятом виде: каждая строка матрицы с новой строки экрана (цикл i по строкам внешний, а цикл j – внутренний)

Б. Обработка массивов в Excel.

В любой таблице Excel при желании можно найти один или несколько одномерных и многомерных массивов:



Месяц	2000	2001	2002	2003	2004
Январь	$8 955	$3 718	$933	$11 467	$8 139
Февраль	$12 584	$2 594	$5 062	$10 488	$14 755
Март	$20 500	$23 675	$15 252	$13 218	$9 834
Апрель	$24 690	$32 599	$16 314	$32 090	$28 128
Май	$185 175	$140 025	$25 258	$224 952	$34 009
Июнь	$234 555	$197 383	$318 947	$76 422	$209 225
Июль	$325 000	$105 452	$202 395	$264 986	$214 013
Август	$283 935	$377 642	$343 550	$261 563	$116 556
Сентябрь	$246 900	$67 408	$114 547	$105 633	$166 035
Октябрь	$185 175	$198 259	$35 515	$116 462	
Ноябрь	$98 760	$147 698	$40 031	$66 164	
Декабрь	$56 665	$64 219	$52 589	$11 596	

Формулы массива в Excel - это специальные формулы для обработки данных из таких массивов. Формулы массива делятся на две категории - те, что возвращают одно значение и те, что дают на выходе целый набор (массив) значений. Рассмотрим их на простых примерах...

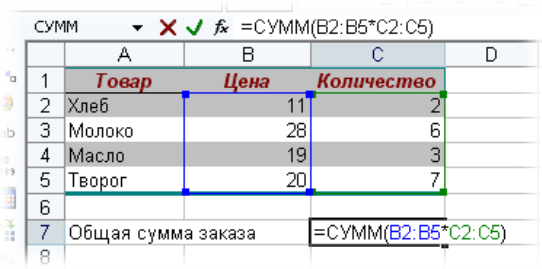
Пример 1. Товарный чек

	A	B	C
1	***Товар***	***Цена***	***Количество***
2	Хлеб	11	2
3	Молоко	28	6
4	Масло	19	3
5	Творог	20	7
6			
7	Общая сумма заказа		
8			

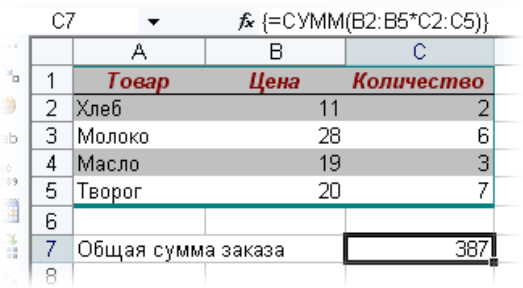
Задача: рассчитать общую сумму заказа. Если идти классическим путем, то нужно будет добавить столбец, где перемножить цену и количество, а потом взять сумму по этому столбцу. Если же применить формулу массива, то все будет гораздо красивее:

1. выделяем ячейку **C7**
2. вводим с клавиатуры **=СУММ(**
3. выделяем диапазон **B2:B5**

4. вводим знак умножения (**звездочка**)
5. выделяем диапазон **C2:C5** и закрываем скобку функции СУММ - в итоге должно получиться так:



6. чтобы Excel воспринял формулу как формулу массива гажимаемне Enter, как обычно, а **Ctrl + Shift + Enter.**



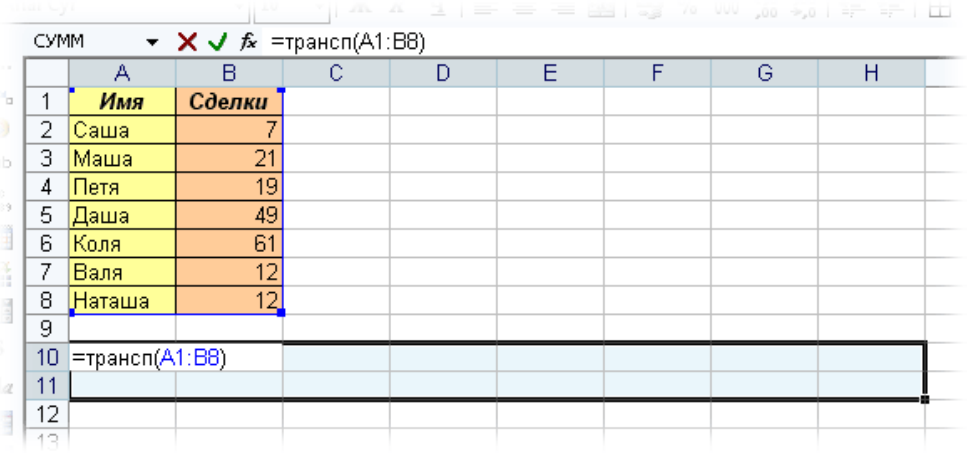
Т.е., Excel произвел попарное умножение элементов массивов B2:B5 и C2:C5 и образовал новый массив стоимостей (в памяти компьютера), а затем сложил все элементы этого нового массива.
Обратите внимание на фигурные скобки, появившиеся в формуле - отличительный признак *формулы массива*. Вводить их вручную с клавиатуры бесполезно - они автоматически появляются при нажатии **Ctrl + Shift + Enter.**

Пример 2. Транспонирование.
При работе с таблицами часто возникает необходимость поменять местами строки и столбцы, т.е. развернуть таблицу на бок, чтобы данные, которые раньше шли по строке, теперь располагались в столбцах и наоборот. В математике такая операция называется транспонированием. При помощи формулы массива и функции **ТРАНСП (TRANSPOSE)** это делается на раз.

Допустим, имеем двумерный массив ячеек, который хотим транспонировать.

	A	B
1	***Имя***	***Сделки***
2	Саша	7
3	Маша	21
4	Петя	19
5	Даша	49
6	Коля	61
7	Валя	12
8	Наташа	12

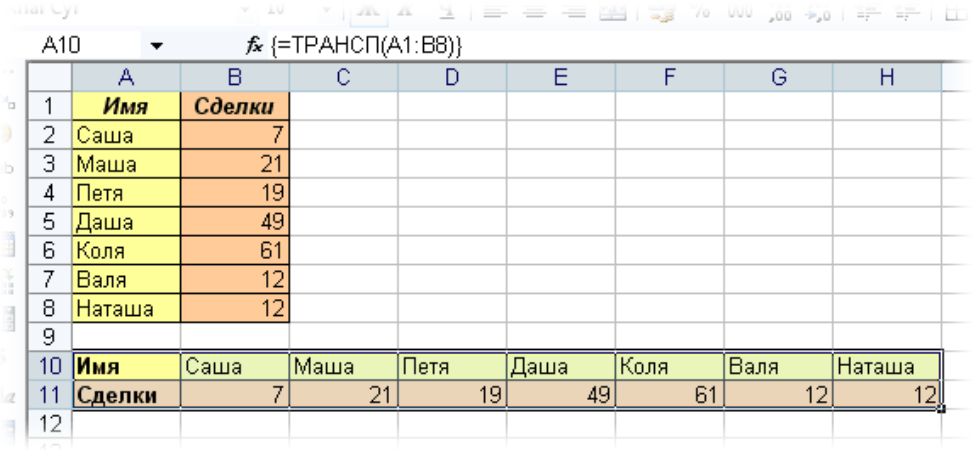
- Выделяем диапазон ячеек для размещения транспонированной таблицы. Поскольку исходный массив ячеек был 8 строк на 2 столбца, то надо выделить диапазон пустых ячеек размером 2 строки на 8 столбцов.
- вводим функцию транспонирования =ТРАНСП(
- в качестве аргумента функции выделяем наш массив ячеек A1:B8



СУММ ✕ ✓ fx =трансп(A1:B8)

	A	B	C	D	E	F	G	H
1	***Имя***	***Сделки***						
2	Саша	7						
3	Маша	21						
4	Петя	19						
5	Даша	49						
6	Коля	61						
7	Валя	12						
8	Наташа	12						
9								
10	=трансп(A1:B8)							
11								
12								

Нажимаем клавиши**Ctrl + Shift + Enter** и получаем "перевернутый массив" в качестве результата:



A10 {=ТРАНСП(A1:B8)}

	A	B	C	D	E	F	G	H
1	***Имя***	***Сделки***						
2	Саша	7						
3	Маша	21						
4	Петя	19						
5	Даша	49						
6	Коля	61						
7	Валя	12						
8	Наташа	12						
9								
10	**Имя**	Саша	Маша	Петя	Даша	Коля	Валя	Наташа
11	**Сделки**	7	21	19	49	61	12	12
12								

Редактирование формулы массива

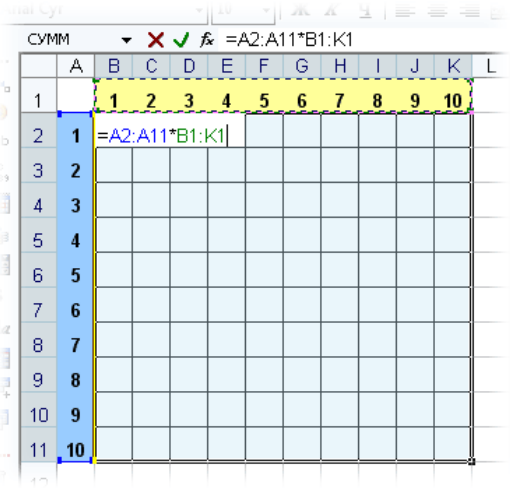
Если формула массива расположена не в одной ячейке, а в нескольких ячейках, то Excel не позволит редактировать или удалить одну отдельно взятую формулу (например в ячейке D10) и выдаст предупреждающее сообщение **Невозможно изменить часть массива.**

Для редактирования формулы массива необходимо выделить весь диапазон (A10:H11 в данном случае) и изменить формулу в строке формул (или нажав **F2**). Затем необходимо повторить ввод измененной формулы массива, нажав сочетание клавиш **Ctrl + Shift + Enter.**

Excel также не позволит свободно перемещать ячейки, входящие в формулу массива или добавлять новые строки-столбцы-ячейки в диапазон формулы массива (т.е. в диапазон A10:H11 в нашем случае)

Пример 3. Таблица умножения

При помощи формул массива она вся делается в одно движение:



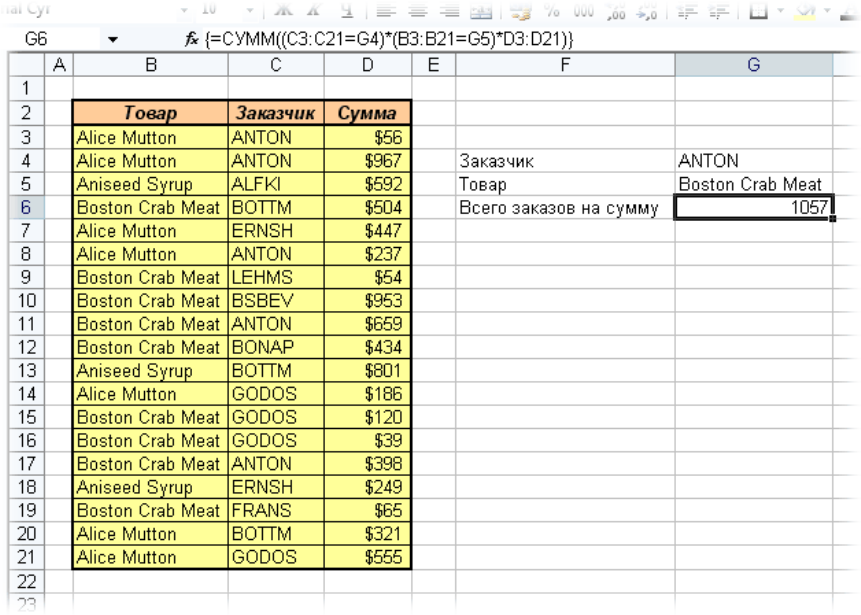
1. выделяем диапазон B2:K11
2. вводим формулу =A2:A11*B1:K1
3. нажимаем **Ctrl + Shift + Enter**, чтобы Excel воспринял ее как формулу массива

и получаем результат:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Пример 4. Выборочное суммирование

Посмотрите как при помощи одной формулы массива красиво и легко выбираются данные по определенному товару и заказчику:



G6 {=СУММ((C3:C21=G4)*(B3:B21=G5)*D3:D21)}

	A	B	C	D	E	F	G
1							
2		*Товар*	*Заказчик*	*Сумма*			
3		Alice Mutton	ANTON	$56			
4		Alice Mutton	ANTON	$967		Заказчик	ANTON
5		Aniseed Syrup	ALFKI	$592		Товар	Boston Crab Meat
6		Boston Crab Meat	BOTTM	$504		Всего заказов на сумму	1057
7		Alice Mutton	ERNSH	$447			
8		Alice Mutton	ANTON	$237			
9		Boston Crab Meat	LEHMS	$54			
10		Boston Crab Meat	BSBEV	$953			
11		Boston Crab Meat	ANTON	$659			
12		Boston Crab Meat	BONAP	$434			
13		Aniseed Syrup	BOTTM	$801			
14		Alice Mutton	GODOS	$186			
15		Boston Crab Meat	GODOS	$120			
16		Boston Crab Meat	GODOS	$39			
17		Boston Crab Meat	ANTON	$398			
18		Aniseed Syrup	ERNSH	$249			
19		Boston Crab Meat	FRANS	$65			
20		Alice Mutton	BOTTM	$321			
21		Alice Mutton	GODOS	$555			
22							

В данном случае формула массива синхронно пробегает по всем элементам диапазонов C3:C21 и B3:B21, проверяя, совпадают ли они с заданными значениями из ячеек G4 и G5. Если совпадения нет, то результат равенства ноль, если совпадение есть, то единица. Таким образом суммы всех сделок, где заказчик не ANTON и товар не Boston Crab Meat умножаются на ноль и суммируются только нужные заказы.

Простейшие операции с массивами

Часто при работе с таблицами возникает необходимость применить одну и ту же операцию к целому диапазону ячеек или произвести расчеты по формулам, зависящим от большого объема данных. **Массив** – это прямоугольный диапазон ячеек с однородными (однотипными) данными. Использование массивов дает возможность ввести одну формулу для выполнения вычислений сразу во многих ячейках. Формула массива может выполнить несколько вычислений, а затем вернуть одно или группу значений. Для работы с массивами есть специальные функции, кроме того, обычные функции для обработки массивов применяются особым образом. Рассмотрим **операцию умножения массива на число** . В качестве примера введите данные в диапазон A1:B2. Далее выделите на рабочем листе область такого же размера, как массив-множимое (например, D 1: E

2). В строке формул введите формулу = A 1: B 2*5 и закончите ввод нажатием сочетания клавиш < Ctrl >+< Shift >+< Enter >. Таким образом вы сообщите программе, что необходимо выполнить операцию над массивом. При этом Excel заключит формулу в строке формул в фигурные скобки {= A 1: B 2*5} **Эти скобки нельзя вводить вручную**, при этом возникнет сообщение об ошибке. При работе с массивами формула действует на все ячейки диапазона. Нельзя изменять отдельные ячейки в операндах формулы. Аналогично можно вычислить:
сумму (разность) массивов {= A 1: B 2+ D 1: E 2};
поэлементное умножение(деление) массивов {= A 1: B 2* D 1: E 2};
массив, каждый элемент которого связан посредством некоторой функции с соответствующим элементом первоначального массива {= sin (A 1: B 2)}.
Примером **специальной функции** для работы с массивом может служить функция транспонирования массива ТРАНСП. Введите в таблицу значения массива. Выделите целевой диапазон ячеек (т.е. диапазон, в который будет вставлен результат транспонирования). Целевой диапазон обязательно должен соответствовать по количеству строк и столбцов ожидаемому результату. Если выделите иное количество строк или столбцов, получите сообщение об ошибке. Затем выполните вставку функции ТРАНСП из категории **Ссылки и массивы**. Во втором окне введите вручную или укажите с помощью мыши исходный диапазон с данными. Завершите ввод нажатием комбинации клавиш < Ctrl >+< Shift >+< Enter >. В Excel могут быть созданы формулы массивов, в которых сами формулы содержат массивы значений, используемых в расчетах. Например, в ячейках B2:F2 находятся данные 1, 2, 3, 4, 5; и 2, 4, 6, 8, 10 – данные, находящиеся в ячейках B3:F3. Тогда результаты попарного сложения могут быть помещены в диапазон B4:F4 с помощью формулы: = {1;2;3;4;5} +{2;4;6;8;10}. После ввода формулы и нажатия сочетания клавиш <CTRL+SHIFT+ENTER> Excel поместит в каждую ячейку выделенного диапазона формулу: {={1;2;3;4;5}+{2;4;6;8;10}}. **Формулы массива** редактируются так же, как и другие формулы.

Нужно дважды щелкнуть по содержащей формулу ячейке, которую необходимо редактировать. В поле ввода можно внести в формулу необходимые изменения. По окончании редактирования обязательно нажать сочетание клавиш <Ctrl+Shift+Enter>. Только в этом случае все изменения в формуле будут внесены во все ячейки, содержащие данную формулу. Действия, выполняемые Excel при обработке формул массивов, могут рассматриваться как математические операции над матрицами. Результат вычислений с матрицами приводит к созданию новых матриц. Если правильно составить формулу, то Excel позволяет производить вычисление и обработку диапазонов разных размеров.

В Excel имеются следующие специальные функции для работы с матрицами: МОБР – обратная матрица; МОПРЕД – определитель матрицы; МУМНОЖ – матричное произведение двух матриц; и уже знакомая функция ТРАНСП – транспонированная матрица.

Во всех случаях при работе с матрицами перед вводом формулы надо выделить область на рабочем листе, куда будет выведен результат вычислений. В качестве примера решим систему линейных уравнений с двумя неизвестными, матрица коэффициентов которой записана в ячейки D 1: E 2 , а свободные члены – в ячейки G1:G2 . Напомним, что решение системы линейных уравнений, записанной в матричном виде **AX=B** , где **A** – матрица коэффициентов, **B** – столбец (вектор) свободных членов, **X** – столбец (вектор) неизвестных, имеет вид **X=A -1 B** , где **A -1** - матрица, обратная по отношению к матрице **A** . Выделим под вектор решений диапазон H1:H2 и введем формулу **{=МУМНОЖ(МОБР(D 1: E 2);G1:G2)}.**

Другой пример - решение системы линейных уравнений ***A 2 X=B*** , где

$$A = \begin{pmatrix} 7 & 2 \\ 1 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

С учетом, что элементы матрицы **A** введены в ячейки A1:B2 , элементы матрицы свободных членов B – в диапазон D 1: D 2 , а в диапазон F 1: F 2 поместим элементы вектора решения, формула будет иметь вид **{=МУМНОЖ(МОБР(МУМНОЖ(A1:B2;A1:B2)); D 1: D 2)}** Результатом вычисления приведенных примеров является

массив. При вычислении следующих выражений результатом является одно число. Пример – вычислить квадратичную форму $z = X^T AX$, где символ (Т) – операция транспонирования матрицы. Матрица **А** введена в диапазон A1:B2 , вектор **X** – в диапазон D1:D2 . Для вычисления в ячейку F1 введем формулу **{=МУМНОЖ(МУМНОЖ(ТРАНСП(D 1: D 2);A1:B2); D 1: D 2)}** Хотя результатом является число, обязательно нажмите комбинацию клавиш <Ctrl+Shift+Enter>. Решение квадратичной формы $z = Y^T A^T AY$, где $A = \begin{pmatrix} 7 & 2 \\ 1 & 4 \end{pmatrix}, \quad Y = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$Матрица **А** введена в диапазон A1:B2 , вектор **Y** – в диапазон D1:D2 . Для вычисления в ячейку F1 введем формулу нужно ввести формулу:**{=МУМНОЖ(ТРАНСП(D 1: D 2);МУМНОЖ(ТРАНСП(A1:B2); МУМНОЖ(A1:B2; D 1: D 2)))} .**

ЗАДАНИЕ:

А. Составить и отладить программу на языке высокого уровня.

Перед выполнением задания изучите теоретический материал.

1. Напишите фрагмент программы для заполнения элементов массива датчиком случайных чисел в диапазоне (sin(N), cos(N)*2), где N – порядковый номер в студенческой группе.
2. Создайте и распечатайте в виде таблицы квадратную матрицу размерностью k=mod(N,4)+3.
3. Создайте массив A(3), сформированный датчиком случайных чисел на интервале (-50,113). Из элементов массива А кратных N сформируйте массив В.
4. Составьте по следующей программе блок-схему и поясните ее функциональное назначение.

For i:=1 to 40 do begin
A[i]:=random(100);
Writeln('number', i, ' element array', a[i]

End;
For i:=1 to 20 do begin
P:=a[i]; a[i]:=a[40-i+1]; a[40-i+1]:=p
End;
Выполните трассировку программы.
5. Напишите и отладьте программу вычисления средних значений столбцов положительных элементов в квадратной матрице.
6. Напишите и отладьте программу преобразования матрицы в вектор.
7. Напишите и отладьте программу преобразования вектора в матрицу.
8. Составьте алгоритм замены в матрице строк – содержащей элемент с максимальным значением и содержащей минимальное значение (предполагается, что эти элементы единственные).
9. Составьте алгоритм и программу преобразования предложения в квадратную символьную матрицу, каждым элементом которой является два символа из предложения, если они являются буквами, три – если среди них встречается хотя бы одна цифра, четыре – если нет цифр и есть знак препинания.

Б. **В электронной таблице EXCEL.**
Перед выполнением задания изучите теоретический материал и просмотрите видиоурок в Интернет по адресу:
http://www.titorov.ru/index.php/distant/inform-tecnology/535-excel
1. Создать массив целых чисел с помощью датчика случайных чисел в заданном интервале – массив X (10 элементов).
2. Создать матрицу действительных чисел в диапазоне [-1,1] – матрица H (10*10 элементов).
3. С помощью командной строки: определить минимальный четный элемент в массиве X, определить сумму элементов массива H, отсортировать каждый столбец матрицы H по возрастанию, отобразить каждую строчку массива H в виде различных диаграмм, построить точечные графики зависимости массива X от каждого столбца матрицы H и определить соответствующие функциональные

зависимости линий тренда, транспонируйте матрицу H, перемножьте массивы X и H, найдите определитель матрицы H.

Составьте отчет, включающий в себя результаты выполненной работы и краткие ответы на контрольные вопросы.

Контрольные вопросы.

1. Опишите каким образом можно осуществить ввод линейного массива в «строчку» и-или в«столбец»?
2. Каким образом задать трехмерный массив?
3. Опишите способы заполнения массивов.
4. Перечислите основные алгоритмы обработки массивов.
5. Приведите алгоритм в виде блок-схемы, словесного описания, графа, языке высокого уровня, выполняющий формирование из одномерного массива двух одномерных – в один входят отрицательные элементы, в другой – положительные.
6. Опишите способы выводы матрицы на печать.
7. В чем особенности организации программ для обработки массивов произвольных размеров?
8. Составьте алгоритм и программу формирования квадратной матрицы по спирали:

1	2	3	4	5	6
20	21	22	23	24	7
19	28	27	26	25	8
18	29	30	31	32	9
17	36	35	34	33	10
16	15	14	13	12	11

ЛАБОРАТОРНАЯРАБОТА №2.ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ И В ЭЛЕКТРОННЫХ ТАБЛИЦАХ.

Краткие теоретические сведения.

С помощью компьютера можно решать весьма разнообразные задачи обработки текстов: от составления платежных ведомостей до автоматической верстки газет. Для того, чтобы компьютер мог обрабатывать тексты, он должна уметь оперировать не только с числами, но и со словами.

Будем полагать, что текст- это произвольная последовательность символов некоторого алфавита.*Алфавитом может служить любое множество символов, например (0, 1,2, ...), (А, Б, В,...), (A, B, C,...).*

Строкой символов, или символьной (строковой, текстовой) константой, будем называть последовательность символов, заключенных в апострофах.

Строка символов может состоять из одного или нескольких символов, а также не содержать ни одного символа (пустая строка, или строка нулевой длины). Максимальная длина текстовой строки 255 символов.

В Turbo Pascal 7.0 для работы с символами используются два типа переменных:*символьный тип и строковый тип данных.*

1. Символьный тип данных (Char).

Описание: идентификатор char, (var x: char).

Диапазон значений: значением переменной этого типа может быть любой символ – это буквы, цифры, знаки препинания и специальные символы. Каждому символу алфавита соответствует индивидуальный числовой код от 0 до 255.

В Turbo Pascal 7.0 значения для переменных типа char задаются в апострофах: sh := ''; a := '3'; summa :='G'.*

2. Строковый тип данных (string).

Как правило, одно целое число или один символ занимают в памяти ЭВМ два байта. В то же время для изображения символа достаточно одного байта. С целью экономии памяти машины при использовании символьных данных в языке Паскаль введено понятие строки. ***Строкой называется последовательность символов определенной длины.*** *Элементы строки хранятся по два в двух байтах памяти ЭВМ.*

Переменные типа ***string*** *могут быть объявлены следующим образом:*

var *s1: string[30]; s2: string.*

Число 30 означает максимально возможное количество символов строки s1.

В языке программирования Java для работы с символами используется тип данных char и два класса, с именами String и StringBuilder. Класс String позволяет отображать строки на экране, но не имеет механизма манипулирования символами внутри строки.

Класс StringBuilder представляет собой миощный инструмент обработки строк, но не имеет средств отображения на экране.

1. Символьный тип данных (char).

Описание: идентификатор char, (charx;).

Диапазон значений: значением переменной этого типа может быть любой символ – это буквы, цифры, знаки препинания и специальные символы. Данный тип данных занимает 2 байта в памяти, таким образом каждому символу алфавита соответствует индивидуальный числовой код от 0 до 65535. Это позволяет хранить символы всех алфавитов на земле и все символы, которые только необходимы. В связи с этим кодировку символьного типа в языке Java называют юникодом, от английского слова Unicode, что означает "универсальная кодировка".

В Java значения для переменных типа char задаются в апострофах: sh = ''; a = '3';*

summa ='G'.

2. КлассString

Объекты класса ***String*** *могут быть объявлены следующим образом:*

Strings1,s2;

Или так: ***Strings1="Привет студентам";***

Как видно из примера, количество символов, хранимых строкой не указывается, поскольку строка в языке Java представляет собой динамический объект переменной длинны с диапазоном памяти от нуля до физической памяти ЭВМ. Иными словами, вновь созданная строка по умолчанию является пустой, но в процессе работы ее длинна может увеличиваться до любых значений. Следует помнить, что как и в стандартных массивах, номер первого элемента строки начинается всегда с нуля.щ

2. Класс StringBuilder

Объекты класса StringBuilder объявляются следующим образом:

StringBuilder sb=new StringBuilder("Привет");

Эта структура данных может хранить неограниченное количество символов, позволяет их обрабатывать и в любой момент массив хранимых символов может быть передан в класс String:

String st=sb.toString();

В данном примере создается строка st, которой передается содержимое объекта sb.

II. Стандартные функции для работы с символьными величинами

1. Операция сложения символьных величин.

Операция сложения позволяет строить из двух символьных строк третью, состоящих из символов первой, за которой следуют символы второй. Обозначается эта операция знаком "+".Описываем строковые переменные.*var s1, s2, s3: string;*

Присваиваемое значение строки заключается в апострофы. Присвоим первым двум следующие значения, а третья будет равна их склеиванию:

s1: = ' Тишеводы, ';s2:= 'нижетравы';s3:=s1+''+s2;

Строка *s3* имеет значение 'Тише воды, ниже травы'.

В языке Java используется сходный подход. Операция сложения строк понимается как их объединение, что иллюстрирует следующий пример:

String s1="Иванов",s2="Вася",s3;

s3=s1+' '+s2.

Строка s3 имеет значение "Иванов Вася". Обратите внимание, что объявления строк в Java могут производиться в любом месте программы и в процессе объявления им можно присваивать начальные значения. Вместо единичного символа пробела, заключенного в одинарные кавычки, можно было использовать строку с единичным символом, в этом случае пример выглядел бы так: s3=s1+" "+s2.

2. Длина строки

Под длиной строки понимается количество введенных символов, но она не может превышать максимально возможной длины (в описательной части). Это значение можно определить при помощи функции, результат которой целое число, равное количеству символов.

s1:='12345';
s2:= 'Семеро одного не ждут';
k1:=Length(s1);k2:=Length(s2).
В результате значения целых переменных будут равны: kl=5, k2=21.

Данный пример в языке Java будет представлен следующим образом:

```
String s1="12345";
String s2= "Семероодногонеждут";
intk1=s1.length(); intk2=s2.length();
```

3. Копирование

Функция copy(str,n,m*) в Turbo Pascal 7.0* – копируют ***m*** символов строки **str**, начиная с **n**-го символа, при этом исходная строка не меняется. Можно результат этой функции присваивать другой строке или сразу выводить его на экран.

s1:='паровоз';s2:='123456';s3:=copy(s1, 5, 3);
writeln(s3);writeln(copy(s2, 3, 2));
Значения переменной s1=’воз’. А на экране будут выведены следующие строки: воз и 34.

В языке Java для копирования части строк используется метод replace(начало, конец). При этом "начало" определяет начальный индекс копирования, "конец" - конечный индекс копирования минус один. Например, в этом примере:

```
Strings1;
Strings2= "Семеро одного не ждут";
s1=s2.substring(1, 3);
System.out.println(s1);
```

на экране будет напечатано слово "ем", поскольку первый символ копирования равен единице (а строки начинаются с нуля), а последний 3-1=2, скопированы будут символы с индексами 1 и 2.

4. Удаление

В Turbo Pascal 7.0 для этого используется процедура **Delete(str, n,m)**, которая вырезает из строки **str** **m**символов, начиная с **n**-го. таким образом сама строка изменяется.

Данфрагментпрограммы:

s:='123456'; delete(s, 3, 2);writeln(s);

После выполнения этих операторов из строки будут удалены два символа, начиная с третьего, то есть строка будет такой: *s = '1256'*.

В языке Java удаление символов в строке производится с использованием вспомогательного класса StringBuilder, с последующей передачей результата в String. Для примера, вышеописанный пример будет иметь следующий вид:

```
Strings="123456";
StringBuilder sb=new StringBuilder(s);
sb.delete(2, 4);
s=sb.toString();
System.out.println(s);
```

Во второй строке программы создается объект sb класса StringBuilder, которому в качестве начального з-начения передается строка. Далее, используется метод delete(2, 4), который удаляет символы, начиная с позиции 2 (не забывайте про нумерацию с нуля), по позицию 4, но не включая ее. Порсле изменения содержимого строки результат передается в переменную s и печатается на экране. Текст результата будет такой: *1256*

5. Замена (Вставка)

В Turbo Pascal 7.0 это можно сделать, применяя процедуру ***Insert(s1,s2,n)*** *– вставка строки s1 в строку s2, начиная с* ***n****-го символа, при этом первая строка остается такой же, как и была, а вторая получает новое значение.*

s1;='34':s2:='1256'; insert (s1, s2, 3);

В результате выполнения данной процедуры строка будет такой *s2='123456'*.

На языке Java вставка символов используется через вспомогательный класс StringBuilder. Вышеописанныйпримерприметследующийвид:

```
String s1="34";
String s2="1256";
StringBuilder sb=new StringBuilder(s2);
sb.insert(2, s1);
s2=sb.toString();
System.out.println(sb);
```

6. Числа и строки

Надо заметить, что число 25 и строка 25 – это не одно и то же. Для работы с числами и строками в Turbo Pascal 7.0 применяются две процедуры.

Str(n,s1) – переводит числовое значение n в строковое и присваивает результат строке s1, причем можно переводить как целые числа, так и вещественные.

n:=12;
str(n,s1);
- после выполнения s1 ='12';

Существует обратная операция, переводящая строковое значение в числовое.

Функция val(s, n, k) – переводит строковое значение в числовое, если данная строка действительно является записью числа (целого или вещественного), то значение k=0, а n – это число, иначе k будет равно номеру символа, в котором встречается первое нарушение записи числа n.

val('1234',n,k) ***n=1234****, k=0;*

В языке Java также существует перевод строковой информации в численную и наоборот. Для перевода чиленных данных в строковые

можно использовать операцию соединения пустой строки и числа, компилятор при этом правильно преобразует типы:

```
intn=25;
String s=""+n;
System.out.println(s);
```

Во втором случае можно использовать метод valueOf класса String:

```
intn=25;
String s=String.valueOf(n);
System.out.println(s);
```

И в первом и во втором случаях на экран будет выведена строка "25"

Обратное преобразование возможно с использованием соответствующих численному типу классов. Следующий пример иллюстрирует перевод строки в численный тип с использованием класса Integer:

```
String s="25";
intn=Integer.valueOf(s);
System.out.println(n);
```

7. Функции преобразования типов

Иногда в программах возникает необходимость по коду определить символ и, наоборот, по символу определить его код. Для этого используют функцию: ***CHR(x)***.

Эта функция возвращает символ, соответствующий ASCII-коду числа *x*.

```
for i = 0 to 255 do
writeln( i,' ', chr(i));
```

Для определения кода по символу используют функцию **ORD**.

```
readln(s); writeln(ord(s));
```

В языке Java преобразование типа char в численный тип производится с использованием оператора преобразования типа, который выглядит следующим образом:

переменная1=(новый тип)переменная2;

Следующий пример иллюстрирует сказанное:

```
for (inti=0;i<65535;i++)
{
 charc=(char)i;//символ с принялзначениекода i
 intn=c;//число n принялозначениесимвола c
 System.out.println("Код "+i+" имеет символ "+(char)c);
}
```

III. Решение задач на обработку текстовой информации

Пример 1.

Составить программу, определяющую по введенному с клавиатуры символу его код.

```
Programm prim1;
Var s: char;
Begin
Writeln('введитесимволсклавиатуры');Readln(s);
Writeln('кодсимвола ',s,'=',ord(s));
Readln;
End.
```

Пример 2.

В три символьные переменные F, I, O ввести свои фамилию, имя, отчество. Сформировать из этих данных строку S, содержащую ваши фамилию и инициалы.

```
Program prim2
Var F, I, O, S : string;
Begin
Writeln('введите вашу фамилию');
Readln(F);
Writeln('введите ваше имя');
Readln(I);
Writeln('введите ваше отчество');
Readln(O);
S:=F+' '+copy(I,1,1)+'.'+copy(O,1,1)+'.';
Writeln('ваширеквизиты: ', S);
Readln;
End.
```

Пример 3.

Определить сколько цифр содержится в записи произвольного натурального числа.

```
Program prim3;
Var s: string;
x, k: integer;
Begin
Writeln('введитечисло');
Readln(x);
Str( x, s);
k:=length(s);
Writeln('вчисле ',k,' цифр');
Readln;
End.
```

Пример 4.

Переменные A и B содержат строки цифр. Найти сумму соответствующих чисел.

```
Program prim4;
Var A, B: string;
S, x, y, n, k: integer;
Begin
Writeln('введитепервоечисло');
Readln(A);
Writeln('введитевтороечисло');
Readln(B);
Val(A, x, n);
Val(B, y, k);
S:=x+y;
Writeln('суммачиселравна ',S);
Readln;
End.
```

Пример 5.

Распечатать заданное слово в одной строке с разрядкой (пробел после каждой буквы).

```
Program prim5;
Var s, x: string;
i: integer;
Begin
Writeln('введитеслово');Readln(s);
x:='';
For i:=1 to length(s) do beginx;=x+copy(s,i,1)+' '; End;
Writeln('получилосьслово ', x);
Readln;
End.
```

Пример 6.

Составить программу подсчета количества вхождений буквы "а" в заданном тексте.

```
Program prim6;
Var s: string;
i, k: integer;
Begin
Writeln('введитетекст');
Readln(s);
k:=0;
for i:=1 to length(s) do begin
if copy(s, i, 1)='a' then k:=k+1
end;
Writeln('количествобукв "а" втекстеравно ', k);
Readln;
End.
```

Пример 7.

Определить, какое из двух исходных слов длиннее и насколько.

```
Program prim7;
Var s1, s2: string;
L1, l2: integer;
Begin
Writeln('введите первое слово');Readln(s1);
Writeln('введите второе слово');Readln(s2);
l1:=length(s1);
l2:=length(s2);
if l1>l2 then writeln('первое слово длиннее второго на ',l1-l2,' символов')
elseif l1=l2 then writeln('слова одинаковой длины')
else writeln('первое слово длиннее второго на ',l2-l1,' символов');
Readln;
End.
```

Переменная типа char может принимать значения из определенной упорядоченной последовательности символов. Переменная этого типа занимает 1 байт и принимает одно из 256 значений кода ASCII (американский стандартный код для обмена информацией). Символы упорядочены в соответствии с их кодом, поэтому к данным символьного типа применимы операции отношения.

В программе вместо символа можно использовать его код, состоящий из # и номера кодируемого символа (например, #51). Обычно символы, имеющие экранное представление, записывают в явном виде, заключив в апострофы (например, 'A', 'b', '*'). Две стандартные функции позволяют поставить в соответствие данную последовательность символов множеству целых неотрицательных чисел (порядковым номерам символов последовательности).

Эти функции называются функциями преобразования:
ord(ch) – выдает номер символа (нумерация с нуля),
chr(i) – выдает i-ый символ из таблицы символов.

Пример. ord(H) выдает номер символа H в последовательности всех символов, используемых транслятором. chr(15) выдает 15-ый символ этой последовательности.

Кроме того, для символьных переменных применяются такие функции:

pred(ch) – возвращает предыдущий символ;
succ(ch) – возвращает следующий символ;
upcase(ch) – преобразует строчную букву в заглавную. Обрабатывает буквы только латинского алфавита.

В языке Java для вывода кода символа достаточно использовать операцию преобразования типа: charc=(char)65; или обратное преобразование: intn=(int)'A';

Также можно использовать процедуры inc и dec.

Строковый тип данных

Для обработки строковой информации в Турбо Паскаль введен строковый тип данных. Строкой в Паскале называется последовательность из определенного количества символов. Количество символов последовательности называется длиной строки.

Синтаксис:

```
var s: string[n];
var s: string;
```

n - максимально возможная длина строки - целое число в диапазоне 1..255. Если этот параметр опущен, то по умолчанию он принимается равным 255.

Строковые константы записываются как последовательности символов, ограниченные апострофами. Допускается формирование строк с использованием записи символов по десятичному коду (в виде комбинации # и кода символа) и управляющих символов (комбинации ^ и некоторых заглавных латинских букв).

Примеры:

```
'Текстовая строка'
#54#32#61
```

'abcde'^A^M

Пустой символ обозначается двумя подряд стоящими апострофами. Если апостроф входит в строку как литера, то при записи он удваивается.

Переменные, описанные как строковые с разными максимальными длинами, можно присваивать друг другу, хотя при попытке присвоить короткой переменной длинную лишние символы будут отброшены.

Выражения типа char можно присваивать любым строковым переменным.

В Турбо Паскаль имеется простой доступ к отдельным символам строковой переменной: i-й символ переменной st записывается как st[i]. Например, если st - это 'Строка', то st[1] - это 'С', st[2] - это 'т', st[3] - 'р' и так далее.

Над строковыми данными определена операция слияния (конкантенации), обозначаемая знаком +. Например:

```
a := 'Turbo';b := 'Pascal';c := a + b;
```

В этом примере переменная с приобретет значение 'TurboPascal'.

Кроме слияния над строками определены операции сравнения <,>,=,<>,<=,>=. Две строки сравниваются посимвольно, слева направо, по кодам символов. Если одна строка меньше другой по длине, недостающие символы короткой строки заменяются символом с кодом 0.

В системе Turbo Pascal имеется несколько стандартных процедур и функций, ориентированных на работу со строками. Например:

```
Length(s:string):integer
```

Функция возвращает в качестве результата значение текущей длины строки-параметра

```
n := length('Pascal'); {n будетравно 6}
Concat(s1,[s2,...,sn]:string):string
```

Функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина строки-результата превышает 255

символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации "+" и работает немного менее эффективно, чем эта операция.

Copy(s:string; index:integer; count:integer):string

Функция возвращает подстроку, выделенную из исходной строки s, длиной count символов, начиная с символа под номером index.

```
s := 'Система Turbo Pascal';
s2 := copy(s, 1, 7); {s2 будет равно 'Система'}
s3 := copy(s, 9, 5); {s3 будетравно 'Turbo'}
s4 := copy(s, 15, 6); {s4 будетравно 'Pascal'}
```

Delete(var s:string; index,count:integer)

Процедура удаляет из строки-параметра s подстроку длиной count символов, начиная с символа под номером index.

```
s := 'Система Turbo Pascal';delete(s,8,6); {s будет равно 'Система Pascal'}
```

Insert(source:string; var s:string;index:integer)

Процедура предназначена для вставки строки source в строку s, начиная с символа index этой строки.

```
s := 'Система Pascal';insert('Turbo ',s,9); {s будет равно 'Система Turbo Pascal'}
```

Pos(substr,s:string):byte

Функция производит поиск в строке s подстроки substr. Результатом функции является номер первой позиции подстроки в исходной строке. Если подстрока не найдена, то функция возвращает 0.

```
s := 'Система Turbo Pascal';x1 := pos('Pascal', s); {x1 будет равно 15}
x2 := pos('Basic', s); {x2 будетравно 0}
```

Str(X: *арифметическое выражение*; var st: string)

Процедура преобразует численное выражение X в его строковое представление и помещает результат в st.

Val(st: string; x: *числоваяпеременная*; var code: integer)

Процедура преобразует строковую запись числа, содержащуюся в st, в числовое представление, помещая результат в x. x - может быть как целой, так и действительной переменной. Если в st встречается недопустимый (с точки зрения правил записи чисел) символ, то преобразование не происходит, а в code записывается позиция первого недопустимого символа. Выполнение программы при этом не прерывается, диагностика не выдается. Если после выполнения

процедуры code равно 0, то это свидетельствует об успешно произошедшем преобразовании.

В дополнение приведем некоторые функции, связанные с типом char, но которые тем не менее часто используются при работе со строками.

Chr(n: byte): char

Функция возвращает символ по коду, равному значению выражения n. Если n можно представить как числовую константу, то можно также пользоваться записью #n.

Ord(ch: char): byte;

В данном случае функция возвращает код символа ch.

UpCase(c: char): char;

Если c - строчная латинская буква, то функция возвращает соответствующую прописную латинскую букву, в противном случае символ c возвращается без изменения.

Понятие множества в языке Паскаль основывается на математическом представлении о конечных множествах: это ограниченная совокупность различных элементов. При этом понятие множества встречается исключительнов языке Паскаль, подобная реализация полностью отсутствует в языках программирования Java, C# и C++. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элементов, составляющих множество, называется базовым типом.Множественный тип описывается с помощью служебных слов Set of, например:

typeM = SetofB;Здесь M - множественный тип, B - базовый тип.

Пример описания переменной множественного типа:

```
Type   M = Set of 'A'..'D';
Var   MS: M;
```

Принадлежность переменных к множественному типу может быть определена прямо в разделе описания переменных:

var C: Set of 0..7;

Константы множественного типа записываются в виде заключенной в квадратные скобки последовательности элементов или интервалов базового типа, разделенных запятыми, например:['A', 'C'] [0, 2, 7] [3, 7, 11..14]

Константа вида [ ] означает пустое подмножество. Количество базовых элементов не должно превышать 256. Инициализация величин множественного типа может производиться с помощью типизированных констант:constseLit: Setof 'A'..'D'= [];

Порядок перечисления элементов базового типа в константах безразличен.

Значение переменной множественного типа может быть задано конструкцией вида [T], где T - переменная базового типа. Например, вполне допустима конструкция:type T = set of char;

Множество включает в себя набор элементов базового типа, все подмножества данного множества, а также пустое подмножество. Так, переменная T множественного типа

```
var T: Set of 1..3;
```

может принимать восемь различных значений:

[ ] [1] [2] [3] [1,2] [1,3] [2,3] [1,2,3]

К переменным и константам множественного типа применимы операции присваивания(:=), объединения(+), пересечения(*) и вычитания(-):

['A','B'] + ['A','D'] даст ['A','B','D']
['A','D'] * ['A','B','C'] даст ['A']
['A','B','C'] - ['A','B'] даст ['C'].

Результат выполнения этих операций есть величина множественного типа.

К множественным величинам применимы операции:
тождественность (=), нетождественность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логический тип, например:

['A','B'] = ['A','C'] даст FALSE
['A','B'] <> ['A','C'] даст TRUE
['B'] <= ['B','C'] даст TRUE
['C','D'] >= ['A'] даст FALSE.

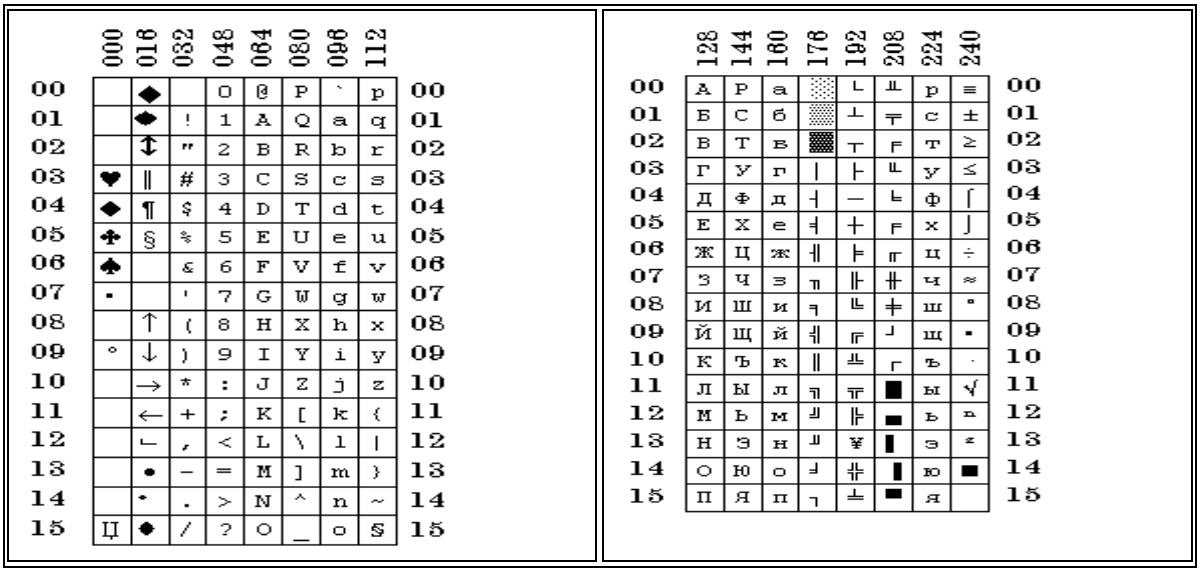
Кроме этих операций для работы с величинами множественного типа в языке ПАСКАЛЬ используется операция in, проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от знака операции. Результат выполнения этой операции - булевский. Операция проверки принадлежности элемента множеству часто используется вместо операций отношения, например:

'A' in ['A', 'B'] даст TRUE,
2 in [1, 3, 6] даст FALSE.

IBM-совместимые компьютеры обрабатывают 256 различных символов, каждый из которых кодируется одним байтом. Соответствие символов и байтов задается ***таблицей кодировки***, в которой для каждого символа указывается соответствующий байт.Символы с кодами от 0 до 127 построены по стандарту ASCII (American Standard Code for Information Interchange — Американский стандартный код обмена информацией, читается "аски"). Вторая половина таблицы (коды 128 ... 255) в нашей стране содержит русские буквы (кириллицу) и символы псевдографики. Для того, чтобы определить по этим таблицам код того или иного символа, нужно сложить номер строки с номером столбца, в которых он расположен. Так, код цифры 5 равен 05+048 = 053.

Символьная информация в алгоритмах и программах описывается данными двух типов: *символьным* и *литерным*. Они отличаются друг от друга тем, что значением символьной переменной является один символ, а литерной — строка символов.

Коды 0...127 (кодировка ASCII)	**Коды 128...255 (модифицированный альтернативный вариант)**



	000	016	032	048	064	080	096	112	
00		◆		0	@	P	`	p	00
01		◆	!	1	A	Q	a	q	01
02		↕	"	2	B	R	b	r	02
03	♥	‖	#	3	C	S	c	s	03
04	◆	¶	$	4	D	T	d	t	04
05	♣	§	%	5	E	U	e	u	05
06	♠		&	6	F	V	f	v	06
07	•		'	7	G	W	g	w	07
08		↑	(	8	H	X	h	x	08
09	°	↓	)	9	I	Y	i	y	09
10		→	*	:	J	Z	j	z	10
11		←	+	;	K	[	k	{	11
12		∟	,	<	L	\	l	|	12
13		•	-	=	M	]	m	}	13
14		•	.	>	N	^	n	~	14
15	Ц	●	/	?	O	_	o	⌂	15

	128	144	160	176	192	208	224	240	
00	А	Р	а	░	└	╨	р	≡	00
01	Б	С	б	░	┴	╤	с	±	01
02	В	Т	в	▒	┬	╥	т	≥	02
03	Г	У	г	|	├	╙	у	≤	03
04	Д	Ф	д	┤	─	╘	ф	⌠	04
05	Е	Х	е	╡	┼	╒	х	⌡	05
06	Ж	Ц	ж	╢	╞	╓	ц	÷	06
07	З	Ч	з	╖	╟	╫	ч	≈	07
08	И	Ш	и	╕	╚	╪	ш	°	08
09	Й	Щ	й	╣	╔	┘	щ	∙	09
10	К	Ъ	к	‖	╩	┌	ъ	·	10
11	Л	Ы	л	╗	╦	█	ы	√	11
12	М	Ь	м	╝	╠	▄	ь	ⁿ	12
13	Н	Э	н	╜	¥	▌	э	²	13
14	О	Ю	о	╛	╬	▐	ю	■	14
15	П	Я	п	┐	╧	▀	я		15

Типы данных, используемые для обработки символьной информации

Язык	Тип, ключевое слово	Примеры использования
	Литерный **лит**	t := "Литерная величина" s := "" (пустая строка)
Turbo Pascal	Символьный **Char**	a := ' f ' ; b := '+' ; c := '5' ; If a = ' ' then k := k + 1
	Литерный **String**	t := 'Литерная величина' ; f := ' ' ; (пустая строка)
QBasic	Литерный	t$:= "Литерная величина" f$:= "" (пустая строка)

Для данных символьного и литерного типов применимы ***операции сцепки*** (соединения, конкатенации) и ***сравнения*** (<, >, <=, >=, =, <>). Сравнивать можно строки разной длины. Сравнение осуществляется слева направо в соответствии с ASCII-кодами соответствующих символов. Так, строка "*стол*" меньше строки "*стул*", строка "*teacher*" больше строки "*pupil*" , а строка "*пар*" меньше строки "*парад*".

Функции и команды обработки строк QBasic

Функции

ASC (X$) Возвращает порядковый номер символа X$ в таблице кодов символов.

CHR$ (N) Возвращает символ с заданным порядковым номером N.

INSTR ([N ,] X$, Y$) Возвращает номер позиции строки X$, начиная с которой в ней размещается подстрока Y$. Если подстрока не найдена, то значение функции равно нулю. Поиск подстроки ведется с позиции N, а если N не задано, то с начала строки.

LEFT$ (X$, N) Возвращает подстроку, составленную из первых N символов строки X$.

LEN (X$) Возвращает количество символов в строке X$.

MID$ (X$, N [, M]) Возвращает подстроку, составленную из M символов строки X$, начиная с позиции N (если параметр M опущен, то возвращаются все символы, начиная с позиции N).

RIGHT$ (X$, N) Возвращает подстроку, составленную из последних N символов строки X$.

STR$ (N) Возвращает представление числа N в символьной форме.

VAL (X$) Возвращает представление символов строки X$ в числовой форме.

Операторы (функции)

**MID$ (X$, N , M) = Y$** Часть строки X$, начиная с позиции N, длиной M позиций заменяется на строку Y$. Длина X$ не изменяется.

**SWAP X$, Y$** Строки X$ и Y$ обмениваются своими значениями.

Пример 1. ***Определить количество слов в заданном тексте.***
Если слова в тексте разделены **одним пробелом**, то задача сводится к подсчету числа пробелов. Количество слов при этом равно числу пробелов плюс 1. Если же **число пробелов**между соседними словами **произвольное**, как обычно и бывает, то алгоритм усложняется. Рассмотрим оба варианта решения этой задачи.

Вариант 1. Слова в тексте разделены одним пробелом.

Тест

Данные	Результат
"Кот на крыше"	**N=3**

Turbo Pascal

```
Program Probel;
  Uses Crt;
 VarText  : String;  {заданныйнепустойтекст}
 i, Number : Integer; {Number — количествословвтексте}
 Letter    : Char;    {текущаябуква }
BEGIN ClrScr;
  WriteLn('Введитетекст :'); ReadLn(Text);
  Number:=1;
  For i:=1 to Length(Text) do {циклпобуквамтекста}
   begin
     Letter:=Text[i];
     If (Letter = ' ' ) then Number:=Number+1;
end;
  WriteLn('О т в е т : количество слов в тексте равно ', Number);
END.
```

Java

```
String text;
Scanner input=new Scanner(System.in);
System.out.println("Введитетекст");text=input.nextLine();
intnumber=1;
for (inti=0;i<text.length();i++)
{
        charletter=text.charAt(i);
        if (letter==' ') number++;
}
System.out.println("Ответ: количество слов в предложении "+number);
```

Вариант 2. Слова в тексте разделены произвольным количеством пробелов.

Тест

Данные	Результат
"Кот на крыше"	**N=3**

```
 Program KolSlov;
 Uses Crt;
 Var Text      : String;  {заданныйтекст}
     i, Number : Integer; {Number - количество слов в тексте}
     Flag      : Boolean;
     Letter    : Char;    {текущаябуква }
BEGIN
 ClrScr;
 WriteLn('Введитетекст :');
 ReadLn(Text);
 Number := 0; Flag := TRUE;
 For i := 1 to Length(Text) do {циклпобуквамтекста}
  begin
    Letter := Text[i];       {текущаябукватекста }
    If (Letter <> ' ') and Flag
      then Number := Number+1;
    Flag := (Letter=' ')     {(Letter=' ') — логическоевыражение,}
  end;                       {принимающее значения TRUE или FALSE }
 WriteLn;
 WriteLn('О т в е т : количество слов в тексте равно ', Number); ReadLn
END.
```

QBasic

```
CLS
PRINT "Введите текст, отделяя слова пробелами."
PRINT "Если в тексте есть запятые, заключите его в кавычки."
INPUT Text$ : PRINT
Number = 0 : Flag = 0
FOR i = 1 TO LEN(Text$)              'циклпобуквамтекста
   Letter$ = MID$(Text$, i, 1)       'текущаябукватекста
   IF (Letter$ <> " ") AND (Flag = 0) THEN Number=Number+1
   IF (Letter$ = " ") THEN Flag = 0 ELSE Flag = 1
NEXT i
```

```
PRINT "О т в е т : количество слов в тексте равно "; Number
END
```

Пример 7.2. ***Определить, является ли заданное слово "перевёртышем" (слово называется "перевёртышем", если совпадает с собой после переворачивания).***

Система тестов

N теста	Данные	Результат
1	Slovo = "казак"	Otvet = "Перевертыш"
2	Slovo = "коза"	Otvet = "Не перевертыш"

Turbo Pascal

```
Program TurnOver;
  Uses Crt;
  Var Slovo    : String;
      Dlina, i : Integer;
      Flag     : Boolean;
BEGIN
  ClrScr;
  Write('Введитеслово : ');  ReadLn(Slovo);
  Dlina:= Length(Slovo);
{Сравниваются пары букв: первая буква с последней, }
    {вторая буква с предпоследней и т.д. }
  i:=1;  Flag := TRUE;
  While (i <= Dlina/2) and Flag do       {циклдопервойнесовпавшей }
    begin                                {пары букв (если такая есть)}
      Flag := (Slovo[i]=Slovo[Dlina-i+1]);
      i := i+1
    end;
  WriteLn; Write( 'Ответ : слово ', Slovo);
  If Flag then WriteLn(' — перевертыш. ')
         else WriteLn(' — не перевертыш');
  ReadLn
END.
```

QBasic

```
CLS : INPUT "Введитеслово : ", SLOVO$
Dlina = LEN(SLOVO$)
  ' Сравниваются пары букв: первая буква с последней,
  ' вторая буква с предпоследней и т.д.
```

```
i = 1 : Flag = 0
WHILE (i<=Dlina/2) AND (Flag=0) 'цикл до первой несовпавшей пары букв
  Letter1$ = MID$(SLOVO$, i, 1)          'перваябуквапары
  Letter2$ = MID$(SLOVO$, Dlina-i+1, 1)  'втораябуквапары
  IF Letter1$ = Letter2$ THEN i=i+1 ELSE Flag=1
WEND
PRINT : PRINT "Ответ : слово "; SLOVO$;
IF Flag = 0 THEN PRINT " — перевертыш." ELSE PRINT " — не перевертыш."
END
```

Пример 7.3. ***В заданном тексте одно заданное слово везде заменить на другое заданное слово такой же длины.***

Тест

Данные			Результат
Текст	Слово1	Слово2	
"2sinx+siny"	"sin"	"cos"	"2cosx+cosy"

Turbo Pascal

(эта программа, использующая стандартную функцию Pos , не требует, чтобы длины заменяемого и вставляемого слов были одинаковыми)

```
Program Replace;
  Uses Crt;
  Var Text, Slovo1, Slovo2 : String;
      i, DlinaSlova, P     : Integer;
BEGIN ClrScr;
  Write('Введитестроку : '); ReadLn(Text);
  Write('Какое слово заменить ? '); ReadLn(Slovo1);
  Write('На какое слово заменить ? '); ReadLn(Slovo2);
  WriteLn; WriteLn('Ответ : ');
  WriteLn('Исходныйтекст: ', Text); DlinaSlova:=Length(Slovo1);
  DlinaSlova:=Length(Slovo1);
  P:=Pos(Slovo1,Text); {номерпозиции, скоторойвстроке Text     }
{в первый раз встречается подстрока Slovo1  }
  While P>0 do         {цикл продолжается до тех пор,пока подстрока}
                  {Slovo1 встречается в строке Text          }
   begin
     Delete(Text, P, DlinaSlova); {удаление подстроки Slovo1, начинаю-}
                          {щейся с позиции P, из строки Text  }
```

```
    Insert(Slovo2, Text, P);  {вставка подстроки Slovo2  }
                          { в строку Text с позиции P}
    P:=Pos(Slovo1, Text); {номер позиции, с которой подстрока Slovo1}
                      {встречается в строке Text в очередной раз}
end;
  WriteLn('Новыйтекст: ', Text);
  ReadLn
END.
```

QBasic

```
CLS : INPUT "Введитетекст : " , Text$
INPUT "Какое слово заменить ? " , Slovo1$
INPUT "На какое слово заменить ? " , Slovo2$
PRINT : PRINT "Ответ"
PRINT "Исходныйтекст : " ; Text$
DlinaText = LEN(Text$) : DlinaSlova = LEN(Slovo1$)
FOR i = 1 TO DlinaText-DlinaSlova+1
  IF MID$(Text$, i, DlinaSlova) = Slovo1$ THEN
      MID$(Text$, i) = Slovo2$ : i=i+DlinaSlova
  END IF
NEXT i
PRINT "Новыйтекст : " ; Text$
END
```

Пример 7.4. ***Заданную последовательность слов переупорядочить в алфавитном порядке (то есть выполнить лексикографическое упорядочение).***

Тест

Данные	Результат
Words=("стул", "гора", "яма", "стол")	Words=("гора", "стол", "стул", "яма")

Turbo Pascal

```
Program LexOrder;
  Uses Crt;
  Var Words       : Array[1..10] of String; {массивслов}
      Tmp         : String;       {Tmp — вспомогательная переменная}
      i, j, NWords : Integer;      {NWords — количествослов}
BEGIN
  ClrScr;
  Write('Количество слов в тексте — ');
  ReadLn(NWords);
```

```
  For i := 1 to NWords do
   begin Write(i, '-оеслово : ');
         ReadLn(Words[i])
   end;
  For i := 1 to NWords-1 do {лексикографическоеупорядочениеслов}
    For j := i+1 to NWords do
     If Words[i]>Words[j] then
       begin
         Tmp := Words[i]; Words[i]:=Words[j]; Words[j]:=Tmp
       end;
  WriteLn;  WriteLn('Ответ');
  WriteLn('Лексикографически упорядоченный массив слов:');
  For i := 1 to NWords do Write(Words[i], ' ');
  WriteLn; ReadLn
END.
```

QBasic

```
CLS : INPUT "Количествословвтексте — ", NWords
DIM Words(NWords) AS STRING
FOR i = 1 TO NWords
  PRINT i; "-оеслово " ; : INPUT Words(i)
NEXT i
FOR i = 1 TO NWords - 1     'лексикографическое упорядочение слов
  FOR j = i + 1 TO NWords
    IF Words(i) > Words(j) THEN SWAP Words(i), Words(j)
  NEXT j
NEXT i
PRINT : PRINT "Ответ"
PRINT "Лексикографически упорядоченный массив слов:"
FOR i = 1 TO NWords
  PRINT Words(i); " " ;
NEXT i
PRINT
END
```

Пример 7.5. ***Проверить, имеется ли в линейной записи заданной математической формулы баланс открывающих и закрывающих скобок.***

Система тестов

Номер теста	Проверяемый случай	Данные	Результат
1	При просмотре линейной записи слева направо первой встречается закрывающая скобка	"a)b+1("	"Нет баланса"
2	Первой встречается открывающая скобка, но число открывающих и закрывающих скобок не совпадает	"(a+b))"	"Нет баланса"
3	Есть баланс скобок	"(a+b/(c*d))"	"Есть баланс"

Turbo Pascal

```
Program Balance;
  Uses Crt;
  Var S              : String;
      Dlina, Flag, i : Integer;
BEGIN ClrScr;
  GotoXY(15, 5);
 Write('Введите линейную запись математической формулы :');
  GotoXY(32,7); ReadLn(S);
  i:=1; Flag:=0; Dlina:=Length(S);
  While (Flag>=0) and (i<=Dlina) do
    begin
      If S[i] = '(' then Flag:=Flag + 1;
      If S[i] = ')' then Flag:=Flag - 1;
      i:=i+1
    end;
  GotoXY(32, 9); WriteLn('Ответ');
  GotoXY(15,11);
  If Flag=0 then Write('Естьбаланс ') else Write('Нетбаланса ');
  WriteLn('открывающих и закрывающих скобок');
  ReadLn
END.
```

QBasic

```
CLS
INPUT "Введите линейную запись математической формулы :", S$
i = 1 : Flag = 0 : Dlina = LEN(S$)
WHILE Flag >= 0 AND i <= Dlina
```

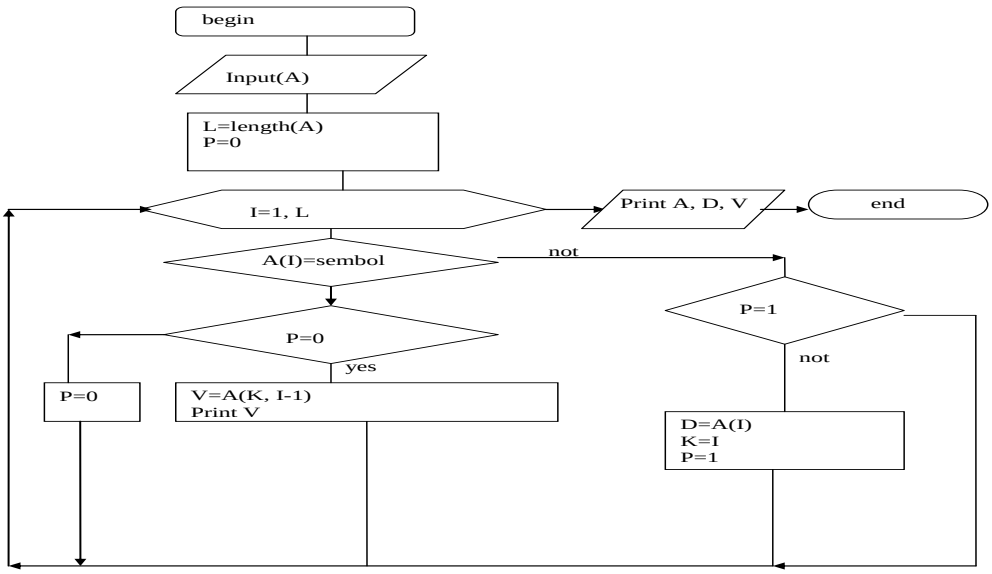
```
  IF MID$(S$, i, 1) = "(" THEN Flag = Flag + 1
  IF MID$(S$, i, 1) = ")" THEN Flag = Flag - 1
  i = i + 1
WEND
PRINT : PRINT "Ответ"
IF Flag = 0 THEN PRINT "Естьбаланс "; ELSE PRINT "Нетбаланса ";
PRINT "открывающих и закрывающих скобок"
END.
```

Задание

Изучите теоретический материал и презентации-видио уроки (файл «Символьная информация в ТР», сайты http://excelvideo.ru/,http://book.kbsu.ru/kbrinfo/kabardinfo/2_7/2_7.html; http://www.excel-eto-prosto.ru/ ?s=yc2&yclid=5959032698412038227).

1. Используя функции символьных переменных написать программу, которая из слова ИНФОРМАТИКА составит слово КИНО.
2. Используя функции символьных переменных, написать программу составления из слов ТЕРРИКОН, ОПЕРА, МЕДИЦИНА слово ТРИОД.
3. Написать и отладить программу, проверяющую – является ли это слово полидромом (читается одинаково «слева - направо» и «справа – налево)?
4. Выполнить пункт 3 в электронной таблице Excel, если каждая буква слова записана в отдельной ячейки.
5. Составить программу замены первого и последнего слова местами, если количество символов в слове максимальной длины больше 4.
6. Ввести текст в электронную таблицу. Замените каждое второе слово, если его длина меньше 3 символов, на мнимое число, действительная часть которого равна количеству символов в предложении без этого слова, мнимая – количеству символов в данном слове.
7. После выполнения п.6 сделайте все символы в предложении прописными.

8. В электронной таблице определите количество совпадений пар символов в двух различных предложениях.
9. Занесите в ячейку электронной таблице любое слово. Измените формат его представления на все имеющиеся модификации. Сравните результаты.
10. Занесите в ячейку целое число меньше 255. Переведите его в символьные форматы (символ, текст, денежный, время, дата).
11. Занесите в ячейку целое число меньше 255. Переведите его в символьные форматы (символ, текст, денежный, время, дата).
12. Занесите в ячейку целое число большее 300. Переведите его в символьные форматы (символ, текст, денежный, время, дата).
13. Выполните трассировку (проверка верификации) следующего алгоритма:



14. Оформите отчет: результаты выполнения заданий и ответы на контрольные вопросы.

Контрольные вопросы:

1. Что такое символьные и текстовые переменные? Как они задаются определяются на языках высокого уровня программирования, Ассемблере, электронных таблицах?
2. Какие стандартные функции определены для символьных переменных?

3. Что напечатается в результате выполнения программы:
a:=’aunkciy’; m:=legth(a)-2; writeln(m)?
4. Что напечатается в результате выполнения программы:
a:=’aunkciy’; b:=’signal’; c:=pos(b,a); m:=legth(a)+2; writeln(c,m-1)?
5. Перечислите и опишите не менее 5 функций, предназначенных для обработки символьной информации в электронной таблице.
6. Перечислите и опишите не менее 5 функций, предназначенных для обработки символьной информации в Паскале.
7. Перечислите и опишите не менее 5 функций, предназначенных для обработки символьной информации в Бейсике.
8. Перечислите и опишите не менее 5 функций, предназначенных для обработки символьной информации .
9. Какие режимы форматирования текстовой информации имеются в электронной таблице?

ЛАБОРАТОРНАЯ РАБОТА №3. ОТОБРАЖЕНИЕ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ ПРОГРАММНО И ИНСТРУМЕНТАРИЕМ EXCEL И MATHCAD

Цель работы: овладение практическими навыками отображения графической информации программными средствами и универсальными пакетами (на примере ExcelMat и MathCad.

Краткие теоретические сведения.

При обработке биомедицинской информации часто возникает задача отображения графической информации, которую можно условно разделить на статическую и динамическую. К статической информации относятся различные графики и диаграммы, к динамической – движущиеся объекты. В последнем случае используются различные мультимедийные средства.

Поскольку размеры экрана монитора и изображаемого объекта в реальности не идентичны, то на первом этапе с помощью линейных преобразований необходимо рассчитать коэффициенты масштабирования по осям ординат и абсцисс.

Графическая информация отображается как программным путем, так и с помощью инструментария универсальных

программных модулей типа MathCad, MatLab, Excel и т.п. В случае разработки программ используются языки программирования высокого уровня или специализированные языки и макросы универсальных программных модулей.

Построение графических объектов в Делфи.

1) //Очистка обозначеного сектора

```
procedure TForm1.Button1Click(Sender: TObject);
varx0,x1,y0,y1:integer;
begin
x0:=strtoint(Edit1.text);x1:=strtoint(Edit2.text);y0:=strtoint(Edit3.text);y1:=strtoint(E
dit4.text);
//Image1.Canvas.FillRect(Rect(0,0,305,21
//Image1.Canvas.Brush.Style:=BsClear;
 Image1.Canvas.Pen.Color:=clWhite;//Цвет контура
Image1.Canvas.Brush.Color:=clWhite;//Цвет Заливки
Image1.Canvas.rectangle(x0,y0,x1,y1);//Цвет линии
end;
```

//**Начало**

```
procedure TForm1.Button2Click(Sender: TObject);
begin
```

2) //Определение размера окна Image1 в пикселах X:(Image1.ClientWidth), Y:(Image1.ClientHeight)

```
label1.caption:='Размер окна: Высота '+inttostr(Image1.ClientHeight)+'   Ширина '
+inttostr(Image1.ClientWidth);
//
```

3) //Пример построения прямоугольника

```
Image1.Canvas.Pen.Color:=clRed;//Цвет линии
Image1.Canvas.rectangle(0,0,305,217);//Цвет линии
//
```

4**) //Пример построения прямоугольника с заккругленными углами (Roundrect)**

```
Image1.Canvas.Pen.Color:=clblue;//Цвет линии
Image1.Canvas.Roundrect(20,20,285,197,10,10);// Roundrect последние цифры
10,10 определяют степень закргленности прямоугольника
//
```

5**) //квадрат сиреневого цвета, с зеленым обводом (свойства: Pen.Width, Pen.Color, Brush.Color)**

```
 Image1.Canvas.Pen.Width:=3;//толщина линии контура
Image1.Canvas.Pen.Color:=clGreen;//Цвет контура линии
Image1.Canvas.Brush.Color:=clFuchsia;//Цвет Заливки
Image1.Canvas.rectangle(30,160,60,190);//Цвет линии
//
```

6) //квадрат сиреневого цвета, с зеленым обводом с штриховкой (свойства: Pen.Width, Pen.Color, Brush.Color)

```
 Image1.Canvas.Pen.Width:=3;//толщина линии контура
Image1.Canvas.Pen.Color:=clGreen;//Цвет контура линии
Image1.Canvas.Brush.Style:=bsFDiagonal;//Тип штриховки
Image1.Canvas.Brush.Color:=clFuchsia;//Цвет Заливки
Image1.Canvas.rectangle(220,140,270,190);//Цвет линии
//
```

7) //Построение треугольника из линий

```
 Image1.Canvas.Pen.Width:=3;//толщина линии контура
Image1.Canvas.Pen.Color:=clyellow;//Цвет контура линии
Image1.Canvas.Moveto(60,160);// Установка пера на начальную точку
Image1.Canvas.LineTo(160,20);// Линия Снизу вверх
Image1.Canvas.LineTo (230,160);//Линия Сверху вниз
Image1.Canvas.LineTo(60,160);//Горизонтальная нижняя линия
//
```

8) //Построение окружности или элипса

```
Image1.Canvas.Pen.Color:=clBlue;//Цвет контура
Image1.Canvas.Brush.Color:=clLime;//Цвет Заливки контура
Image1.Canvas.Brush.Style:=bsSolid;//Тип штриховки
Image1.Canvas.Pen.Width:=2;//толщина линии контура
Image1.Canvas.Ellipse(120,70,190,140);// Координаты прямоугольника в котором строится окружность
```

9) //Построение коричневой точки в центре окружности

```
 Image1.Canvas.Pen.Color:=clMaroon;//Цвет контура линии
Image1.Canvas.Pixels[153,105]:=clRed;//Установить красную точку в координатах x=150,y=110
//
```

10) //Созданиен надписи около окружности (стр 261)

```
Image1.Canvas.Brush.Style:=BsClear;//Установить бесцветный цвет фона
```

```
Image1.Canvas.Font.Size:=9;//Установка размера шрифта
Image1.Canvas.Font.Style:=[fsBold,fsUnderline];//установка стиля шрифтов
(Жирный, подчеркнутый)
Image1.Canvas.Font.Color:=clRed;//Установка цвета шрифта (Красный)
Image1.Canvas.TextOut(155,105,'Центр окружности');//Установка текста надписи
//
```

Построение графических объектов на языке Java

На языке Java создание графических изображений возможно в программах с графическим интерфейсом, у которых главной действующей единицей является форма (объект класса JFrame). Этот язык позволяет получить прямой доступ к форме и использовать богатую библиотеку 2D и 3D графики для создания приложений и игр. Для простоты рассмотрим простейший примепр программы с графическим интерфейсом, в произвольной части которой рисуется красная линия:

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
publicclassExample1extends JFrame{
publicvoid paint(Graphics g) {
super.paint(g);  // вызов метода рисования предка
Graphics2D g2 = (Graphics2D) g;
   Line2D lin = new Line2D.Float(100, 100, 250, 260);
g2.setColor(Color.RED);
g2.draw(lin);
   }
publicstaticvoid main(String []args){
      Example1 s=new Example1();
s.setVisible(true);
s.setSize(400,400);
   }
       }
```

Как видно из представленного примера, основной действующей единицей программы является класс Example1, наследуемый от стандартного класса формы JFrame. В главной программе (main) создается объект этого класса и задаются начальные размеры формы 400X400. Рисование происходит в методе paint(), в котором создается

объект g2 класса Graphics2D, который позволяет отображать двумерные объекты. Далее, создается объект lin с координатами концов 100,100 и 250, 260, который и отображается на форме

1) //Очистка обозначеного сектора

Осуществляется вызовом метода clearRect с координатами прямоугольной зоны, которая подлежит очистке. Сказанное иллюстрирует пример:

```
g2.clearRect(0, 0, 400, 400);
```

2) //Определение размера окна в пикселах:

Так как рисование производится на форме, то внутри класса формы доступ к размерам окна можно получить через указатель на форму (this) и вызовом метода getHeight() и getWidth().

```
inth=this.getHeight();
intw=this.getWidth();
```

3) //Пример построения прямоугольника

```
publicvoidpaint(Graphicsg) {
super.paint(g);  // Вызов метода рисования предка
Graphics2D g2 = (Graphics2D) g;
Rectangle2Dlin = newRectangle2D.Float(100, 100, 250, 260);
g2.setColor(Color.RED); // Задать красный цвет
g2.draw(lin);
```

Результат рисования представлен ниже

4**) //Пример построения прямоугольника с заккругленными углами (Roundrect)**

```
Graphics2D g2 = (Graphics2D) g;
RoundRectangle2D lin = new RoundRectangle2D.Float(100, 100, 250, 260, 40, 40);
g2.setColor(Color.RED);//Цветлинии
g2.draw(lin);//Зарисовка
```

Roundrect последние цифры 40,40 определяют размеры дуги, которая скругляет стороны прямоугольника

5) //закрашенный квадрат синего цвета

```
Graphicsg2 = g;
g2.setColor(Color.BLUE);//Цветквадрата
g2.fillRect(100, 100, 250, 260);
```

7) //Построение треугольника используя полигон

```
g2.setColor(Color.BLUE);//Цветлинии
int[] x={200,100,300};//создатьмассив x точек
int[] y={50,300,300};//создатьмассив y точек
g.fillPolygon(x,y,3); //рисовать полигон
```

8) //Построение окружности или элипса

```
g2.setColor(Color.BLUE);//Цветэллипса
g.fillOval(150, 150, 90, 150);//рисовать эллипс, центр в точке 150, 150, ширина
90 пикселей, высота 150
```

Построение графических движущих объектов на Паскале

Очень часто хочется "оживить" картинку, "заставить" что-нибудь двигаться. Как сделать это, не используя анимационные программы средствами модуля Graph? Движение простых объектов может быть имитировано с помощью некоторых не очень сложных приемов.

I способ. Имитация движения объекта на экране за счет многократного выполнения программой набора действий: нарисовать – пауза – стереть (нарисовать в том же месте цветом фона) – изменить координаты положения рисунка.

Перед началом составления программы надо продумать описание «двигающегося» объекта, характер изменения координат, определяющих текущее положение объекта, диапазон изменения и шаг.

II способ. Иллюзия движения создается при помощи специальных процедур и функций.

Функция ImageSize(x1, y1, x2, y2: integer) возвращает размер памяти в байтах, необходимый для размещения прямоугольного фрагмента изображения, где x1,y1 – координаты левого верхнего и x2, y2 – правого нижнего углов фрагмента изображения.

Процедура GetImage(x1, y1, x2, y2:integer, var Buf) помещает в память копию прямоугольного фрагмента изображения, где x1, ..,y2 –

координаты углов фрагмента изображения, Buf – специальная переменная, куда будет помещена копия видеопамяти с фрагментом изображения. Buf должна быть не меньше значения, возвращаемого функцией ImageSize с теми же координатами. *Процедура PutImage(x1, y1, x2, y2:integer, var Buf, Mode:word)* выводит в заданное место экрана копию фрагмента изображения, ранее помещенную в память процедурой GetImage. X, Y – координаты левого верхнего угла того места на экране, куда будет скопирован фрагмент изображения; Buf –специальная переменная, откуда берется изображение, Mode – способ копирования. Координаты правого нижнего угла не указываются, так как они полностью определяются размерами выводимой на экран копии изображения. Координаты левого верхнего угла могут быть любыми, лишь бы только копия уместилась в пределах экрана (если копия не размещается на экране, то она не выводится, и экран остается без изменений). Параметр Mode определяет способ взаимодействия размещаемой с уже имеющимся на экране изображением

Примеры

1 Разработка программы, реализующей движение по траектории графического объекта.

Исходные данные:
x,y – координаты стартовой точки, тип целый.
d,t – переменные для инициализации графического режима
Промежуточные данные:
a,b – переменные для построения линии моря, тип целый
y0 – координата для высоты положения линии моря, тип целый
Использование модулей:
1. *crt* включает в себя процедуры очистки (clrscr) и задержки экрана(readkey);
2. *graph* позволяет провести инициализация графического режима с помощью процедуры InitGraph; включает в себя процедуры и

функции, позволяющие вырисовывать графические объекты и применять к ним различные типы, стили и цвета оформления.

Алгоритмическая структура:

1. цикл прямого пересчета *for…to… do;*
2. цикл с предусловием *while … do*

Алгоритм программы:

1. Задание имени программы
2. Открытие модулей
3. Написание процедуры «море» с использованием цикла прямого пересчета
4. Инициализация графического режима
5. В цикле с предусловием произвести рисование волны и задание движения объекта по траектории
6. Рисование графического объекта
7. Задержка выполнения программы
8. Установка шага движения
9. Закрытие графического режима

Листинг программы:

```
program corablik;
uses Graph, Crt;
var d,t,x,y,y0,a,b: integer
procedure more(a,b:integer);
begin
  moveto(0,y0);  setcolor(blue);
  for a:=0 to 680 do   begin
    b:=y0-round(sin(a*pi/180)*30;    lineto(a,b);
   end;
end;
Begin
 d := Detect;  t:=2; InitGraph(d, t, '');
 y0 := 250;  x:=600;
   while x>=0 do
   begin
   cleardevice;
   more(a,b);   setcolor(white);
   y:=y0-40-round(sin(x*pi/180)*30);
   MoveTo(x - 60, y + 40);   LineTo(x - 40, y + 60);
```

```
  LineTo(x + 40, y + 60);   LineTo(x + 60, y + 40);
  LineTo(x - 60, y + 40);
  MoveTo(x + 35, y + 40);   LineTo(x + 35, y - 60);
  LineTo(x - 40, y + 40);   LineTo(x + 35, y + 40);
  delay(2500);
  x:=x-2;{шаг движения}
  end;
 CloseGraph;
end.
```

2 Разработка программы, реализующей перемещение по экрану окружности

Исходные данные:
x,y – начальные координаты центра окружности, тип целый.
r – радиус окружности, тип целый
d,t – переменные для инициализации графического режима
Промежуточные данные:
dx – величина перемещения по оси X, тип целый
dy – величина перемещения по оси Y, тип целый
Использование модулей:
1. *crt* включает в себя процедуры очистки (clrscr) и задержки экрана(readkey);
2. *graph* позволяет провести инициализация графического режима с помощью процедуры InitGraph; включает в себя процедуры и функции, позволяющие вырисовывать графические объекты и применять к ним различные типы, стили и цвета оформления.

Алгоритмическая структура:
1. цикл с постусловием *repeat … until*
2. условный оператор *if…then…[else]*

Алгоритм программы:
1. Задание имени программы
2. Открытие модулей
3. Инициализация графического режима
4. Рисование рамки вокруг экрана
5. Рисование окружности белого цвета

6. С помощью условного оператора If указывается смена направления движения при достижении края экрана и включение звукового экрана
7. Задержка выполнения программы
8. Рисование черной окружности
9. Расчет новых координат
10. Закрытие графического режима

Листинг программы

```
ProgramMultik;
UusesGraph, Crt;
Varx, y, dy, dx, r, d, : integer;
Begin
 d :=detect; t:=2;
 Initgraph(d,t,");
 Rectangle(0,0,GetMaxX,GetMaxY);
 x:=100; y:=100;      dx:=10;          dy:=10;          r:=15 ;
Repeat
  SetColor(15);       Circle(x,y,r);
  ify>=GetMaxY-radiusthen
     begindy:=-10; Sound(2000); end;
  ify<=radius         the     begindy:= 10; Sound(3000); end;
  ifx>=GetMaxX-radiusthe      begindx:=-10; Sound(5000); end;
  ifx<=radius         then         begindx:= 10; Sound(4000); end;
  Delay(1000);
  NoSound;
  SetColor(0);
 Circle(x,y,r);
  x:=x+dx; y:=y+dy;
 UntilKeyPressed;
 CloseGraph;                End.
```

Использование графического режима на Бейсике

Базовые понятия работы в графическом режиме на языке высокого уровня Бейсик представлены в презентации на сайте http://festival.1september.ru/articles/603216/.

Монитор может работать в нескольких режимах, которые отличаются друг от друга разрешающей способностью (т. е.

количеством точек по горизонтали и вертикали), и также количеством различных цветов. Существует много различных графических режимов, но для рисования картинок в Basic наиболее употребительны следующие: 320x200 4 CGA,EGA,VG 2 640x200 2 ACGA.EGA.VG 7 320x200 16 EGA, VGA 8 640x200 16 EGA.VGA 9 640x350 16 EGA, VGA 12 640x480 16 VGA 13* 320x200 256 VGA.

Для установки нужного графического режима в программе надо вначале написать инструкцию SCREEN, без нее рисовать будет нельзя. В простейшем случае она выглядит так: SCREEN <режим> - например, SCREEN 12.

Каждая точка экрана имеет свои координаты. Эти координаты измеряются от левого верхнего угла экрана (точка (0, 0)) вправо по горизонтали (по X) и вниз по вертикали (по Y). Большинство графических инструкций требует задания координат отдельных точек экрана и цвета рисования. Цвет рисования закодирован целыми числами от 0 до 15 следующим образом: 0 — черный (цвет фона), 8 — темно-серый, 1 — синий, 9 — голубой, 2 — темно-зеленый, 10 — ярко-зеленый, 3 — салатовый, 11 — ярко-салатовый, 4 — темно-красный 12 — ярко-красный, 5 — темно-вишневый, 13 — ярко-вишневый, 6 — коричневый, 14 — желтый, 7 — светло-серый, 15 — белый. Коды темного и яркого вариантов одного и того же цвета отличаются на 8.

Рассмотрим простейшие инструкции рисования.

PSET (X,Y) [,цвет] — рисуется одна точка в заданной позиции экрана (x,y). Если параметр цвет задан, то точка имеет этот цвет, иначе она будет белой.

CIRCLE (X,У),радиус [,цвет] — рисуется окружность с центром в точке экрана с координатами (X,Y), с заданным радиусом и цветом. Если цвет не задан, то окружность будет белой.

LINE (X1,Y1)—(X2,Y2) [.цвет] — рисуется прямая линия из точки с координатами (X1,Y1) в точку с координатами (X2,Y2). Если цвет не задан, то линия будет белой.

LINE (X1,Y1)—(X2,Y2) [,цвет] ,B — рисуется прямоугольник заданного цвета. Точки (XI,Y1) и (X2,Y2) задают две любые противоположные вершины этого прямоугольника.
LINE (X1,Y1)—(X2,Y2) [,цвет] ,BF — рисуется прямоугольник, закрашенный заданным цветом. Точки (XI,Y1) и (X2,Y2) определяют две любые противоположные вершины этого прямоугольника. Там, где в этих инструкциях стоят X, Y, XI, X2, Y1, Y2, цвет, радиус, могут быть записаны любые арифметические выражения, но буквы B и BF являются составной частью инструкции LINE и должны быть записаны именно так.
PAINT (X,У),цвет_закрашивания,цвет_границы эта инструкция закрашивает область экрана, ограниченную линиями заданного цвета, точка (X,Y) должна быть внутри этой области. При использовании этой инструкции возможны следующие три распространенные ошибки.
1. Точка (X,Y) находится не внутри, а вне требуемой области. Тогда будет закрашена внешняя по отношению к данной области часть экрана.
2. Закрашиваемая область не ограничена сплошной линией одного цвета или в инструкции PAINT неверно указан цвет границы. Тогда цвет «вырвется наружу» и закрасит весь экран.
3. Точка (X,Y) попадает на место экрана, имеющее цвет границы. Тогда закрашивание сразу прекратится.

Пример использования инструкций.

```
SCREEN 9
поставим несколько точек разного цвета
PSET (20,20), 12 PSET (30,20), 14 PSET (25,30), 11
нарисуем две окружности
CIRCLE (200,100), 50 ,1 CIRCLE (300,100), 70 ,5 REM
Нарисуем треугольник из трех линий
LINE (200,200)—(250,200), 10 LINE (250,200)—(225,250), 10 LINE (200,200)—(225,250), 10
нарисуем простой и закрашенный прямоугольники
LINE (20,200)—(70,250), 6, B LINE (20,270)—(70,300), 9, BF
```

закрасим нарисованный ранее треугольник синим цветом PAINT (225,220), 1, 10

Графика вExcel.

Выделяем ячейки с данными , которые надо разместить на диаграмме. Активизируется инструмент **Мастер диаграмм** (или «Вставка», «Диаграммы» на панели инструментов «Стандартная». Внимательно просматриваем все закладки каждого открывающиеся окна мастера создания диаграмм и выполняем необходимые для реализации цели функции.

Графика в MathCad.

Для построения графиков в Mathcad можно воспользоваться функцией **Вставка > График > Тип графика** или панелью инструментов **График**. Поддерживаются следующие типы графиков: двумерный ("X-Y график"); в полярных координатах ("Полярный график"); линии уровня ("Контурный график"); столбчатая диаграмма ("3D панели"); поверхность ("Поверхностный график"); векторный ("Векторное поле").

Анимация

С помощью анимации пользователь может сделать любую скучную презентацию Power Point более привлекательной, интересной и динамичной. Анимация - это визуальный или звуковой эффект, который может быть добавлен к тексту или графическому объекту слайда (текст или изображение в движении, выделение цветом, изменение шрифта и т.д.). Правильно используя анимацию Power Point пользователь может быть уверен, что информация будет лучше запоминаться. Самые распространенные такие эффекты: -Вход и выход текста; -Изменение глубина текста; -Добавление различных звуков.

Анимацию пользователь может добавить к отдельным объектам слайда, к самим страницам слайда или к группе слайдов. В презентации Power Point объекты могут появляться в определенном, заданном порядке, меняться во время демонстрации слайдов, а по окончанию анимации - исчезать. Еще можно указывать время показа для каждого объекта отдельно.

К каждому элементу презентации Power Point пользователь может установить несколько эффектов одновременно. Анимацию можно задавать как в режиме сортировки слайдов, так и в обычном режиме. Еще существуют схемы анимации - это последовательность уже готовых эффектов в Power Point, которые облегчают работу пользователя с презентацией.

Анимация создана для того, чтобы акцентировать внимание человека на конкретных элементах, но если ее очень много, то это отвлекает и попросту раздражает. Цель презентации - донести определенную информацию слушателю, а звук или анимация не должны отвлекать от самого доклада.

Коллекция анимаций находится на различных сайтах. Например, http://prezentacii.com/animacii-dlya-prezentaciy.html

Порядок выполнения работы.

1. Написать и отладить программу на языке высокого уровня, строящегося на экране объекта образа согласно одного из вариантов: 1 - пятиконечная звезда, 2 -усеченная призма, 3 -ряд окружностей при изменениях диаметра по геометрическому закону, 4 -ряд прямоугольных треугольников с изменяющимся по гармоническому закону периметром, 5 -лесенка, 6 -ступенчатая диаграмма (задается шаг и площадь прямоугольных элементов диаграммы).
2. Написать и отладить программу на языке высокого уровня построения движения окружности по фигуре Лиссажу с изменением цвета окружности по гармоническому закону (задаются: амплитуды, частоты и фазы изменения координат по гармоническому закону).
3. Написать и отладить программу отображения на экране параболы и касательной к ней в точке с заданными координатами.

4. Выполнить пункт 2 с выводом символьной информации на изображение (наименование осей, название графика, текущих координат движения шарика.
5. Выполнить п.2 (без движения – отображается только график движения - и изменения в цвете) и п.3 в Excel. Отобразить диаграммы заданного участка координат движения.
6. Выполнить п.2 (без движения – отображается только график движения - и изменения в цвете) п.3 в MathCad. Отобразить трехмерный график фигуры Лиссажу (третья координата изменяется по гармоническому закону).
7. В Excel в одном из столбцов значения любой гармонической функции, во втором столбце – производной данной функции, в третьем – производной данной функции полученных с помощью численных методов, в третьем – относительную разницу значений второго и третьего столбцов. Отобразить на одном листе три графика одинакового размера: функция, производные функции, разница значений. Сделать выводы.
8. Отобразить с помощью мультимедийных средств возникновение экстрасистолы.
9. Оформить отчет: описание программ (и алгоритмов к ним), скрин-шоты выполненных пунктов, краткое описание выполненных действий, выводы и краткие ответы на контрольные вопросы.

Примечания: п.8 – является заданием дополнительной сложности (за его качественное выполнение на экзамене (зачете) прибавляется до 6 баллов);вместо фигуры Лиссажу может быть предложена функция, отображающая движение точки по любой параметрически заданной кривой с изменяющимися во времени координатами или спирали;описание алгоритмов привести в словесной и графической формах, в случае последнего допускается «ручной», а не печатный вариант.

Контрольные вопросы.

1. Каким образом отображается символьная информация в графическом режиме на языках высокого уровня?

2. Каким образом задаются цвета в графическом и символьном режимах на языке высокого уровня?
3. Как вычисляются коэффициенты масштабирования (подобия) ?
4. Сколько и какие фигуры будут выведены на экран в результате выполнения программы: «fori:=1 to 150 dobeginx:=random(640); y:=random(480); setcolor(random(15+i); Dec(i); end.»?
5. Какие формы диаграмм могут быть отображены в Excel?
6. Какие формы графиков могут быть отображены в MathCad?
7. Какие инструментальные средства используются для отображения динамических графических объектов? (Привести наименования и перечень базовых возможностей)

ЛАБОРАТОРНАЯ РАБОТА №4 . РАБОТА С ФАЙЛОВЫМИ СТРУКТУРАМИ

Цель работы: овладение практическими работы с файловыми структурами на языках высокого уровня.

Краткие теоретические сведения.

Информация на внешних носителях хранится в виде файлов. Работа с файлами является очень важным видом работы на компьютере. В файлах хранится все: и программное обеспечение, и информация, необходимая для пользователя. С файлами, как с деловыми бумагами, постоянно приходится что-то делать: переписывать их с одного носителя на другой, уничтожать ненужные, создавать новые, разыскивать, переименовывать, раскладывать в том или другом порядке и пр.

Файл - это информация, хранящаяся на внешнем носителе и объединенная общим именем.

Для прояснения смысла этого понятия удобно воспользоваться следующей аналогией: сам носитель информации (диск) подобен книге. Мы говорили о том, что книга - это внешняя память человека, а магнитный диск - внешняя память компьютера. Книга состоит из глав (рассказов, разделов), каждый из которых имеет

название. Также и файлы имеют свои названия. Их называют именами файлов. В начале или в конце книги обычно присутствует оглавление - список названий глав. На диске тоже есть такой список-каталог, содержащий имена хранимых файлов.
Каталог можно вывести на экран, чтобы узнать, есть ли на данном диске нужный файл.
В каждом файле хранится отдельный информационный объект: документ, статья, числовой массив, программа и пр. Заключенная в файле информация становится активной, т. е. может быть обработана компьютером, только после того, как она будет загружена в оперативную память.
Любому пользователю, работающему на компьютере, приходится иметь дело с файлами. Даже для того, чтобы поиграть в компьютерную игру, нужно узнать, в каком файле хранится ее программа, суметь отыскать этот файл и инициализировать работу программы.
Работа с файлами на компьютере производится с помощью файловой системы. **Файловая система** - это функциональная часть ОС, обеспечивающая выполнение операций над файлами.
Чтобы найти нужный файл, пользователю должно быть известно: а) какое имя у файла; б) где хранится файл .

Имя файла

Практически во всех операционных системах имя файла составляется из двух частей, разделенных точкой. Например:
myprog.pas
Слева от точки находится собственно имя файла (ту-prog). Следующая за точкой часть имени называется расширением файла (pas). Обычно в именах файлов употребляются латинские буквы и цифры. В большинстве ОС максимальная длина расширения - 3 символа. Кроме того, имя файла может и не иметь расширения. В операционной системе Windows в именах файлов допускается использование русских букв; максимальная длина имени - 255 символов.

Расширение указывает, какого рода информация хранится в данном файле. Например, расширение txt обычно обозначает текстовый файл (содержит текст); расширение pcx - графический файл (содержит рисунок), zip или гаг - архивный файл (содержит архив - сжатую информацию), pas - программу на языке Паскаль.
Файлы, содержащие выполнимые компьютерные программы, имеют расширения exe или com. Например, программа популярной игры "Тетрис" хранится в файле tetris.exe. Инициализация программы происходит путем записи ее в оперативную память и перехода работы процессора к ее исполнению.

Логические диски

На одном компьютере может быть несколько дисководов - устройств работы с дисками. Каждому дисководу присваивается однобуквенное имя (после которого ставится двоеточие), например A:, B:, C:. Часто на персональных компьютерах диск большой емкости, встроенный в системный блок (его называют жестким диском), делят на разделы. Каждый из таких разделов называется логическим диском, и ему присваивается имя C:, D:, E: и т. д. Имена A: и B: обычно относятся к сменным дискам малого объема - гибким дискам (дискетам). Их тоже можно рассматривать как имена дисков, только логических, каждый из которых полностью занимает реальный (физический) диск. Следовательно, A:, B:, C:, D: - это всё имена логических дисков.
Имя логического диска, содержащего файл, является первой "координатой", определяющей место расположения файла.

Файловая структура диска

Вся совокупность файлов на диске и взаимосвязей между ними называется **файловой структурой**. Различные ОС могут поддерживать разные организации файловых структур. Существуют две разновидности файловых структур: простая, или одноуровневая, и иерархическая - многоуровневая.
Одноуровневая файловая структура - это простая последовательность файлов. Для отыскания файла на диске

достаточно указать лишь имя файла. Например, если файл tetris.exe находится на диске A:, то его "полный адрес" выглядит так:
A:\tetris.exe
Операционные системы с одноуровневой файловой структурой используются на простейших учебных компьютерах, оснащенных только гибкими дисками.
Многоуровневая файловая структура - древовидный (иерархический) способ организации файлов на диске. Для облегчения понимания этого вопроса воспользуемся аналогией с традиционным "бумажным" способом хранения информации. В такой аналогии файл представляется как некоторый озаглавленный документ (текст, рисунок) на бумажных листах. Следующий по величине элемент файловой структуры называется **каталогом.** Продолжая "бумажную" аналогию, каталог будем представлять как папку, в которую можно вложить множество документов, т. е. файлов. Каталог также получает собственное имя (представьте, что оно написано на обложке папки).
Каталог сам может входить в состав другого, внешнего по отношению к нему каталога. Это аналогично тому, как папка вкладывается в другую папку большего размера. Таким образом, каждый каталог может содержать внутри себя множество файлов и вложенных каталогов (их называют подкаталогами). Каталог самого верхнего уровня, который не вложен ни в какой другой каталог, называется корневым каталогом.
В операционной системе Windows для обозначения понятия "каталог" используется термин "папка".
Графическое изображение иерархической файловой структуры называется деревом.
На рис. 1 имена каталогов записаны прописными буквами, а файлов - строчными. Здесь в корневом каталоге имеются две папки: IVANOV и PETROV и один файл fin.com. Папка IVANOV содержит в себе две вложенные папки PROGS и DATA. Папка DATA - пустая; в папке PROGS имеются три файла и т. д. На

дереве корневой каталог обычно изображается символом \.

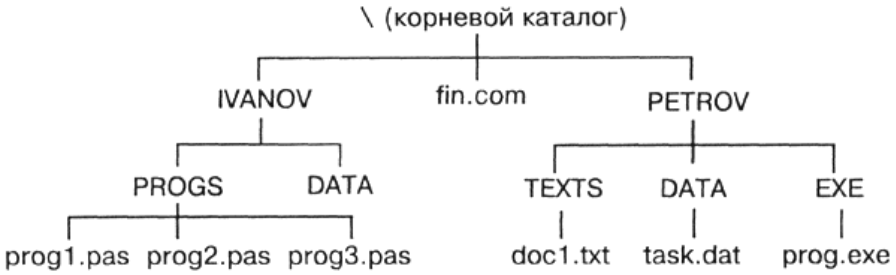


Рисунок 1. Пример иерархической файловой структуры

Путь к файлу

А теперь представьте, что вам нужно найти определенный документ. Для этого надо знать ящик, в котором он находится, а также "путь" к документу внутри ящика: всю последовательность папок, которые нужно открыть, чтобы добраться до искомых бумаг.

Второй координатой, определяющей место положения файла, является **путь к файлу на диске**. Путь к файлу - это последовательность, состоящая из имен каталогов, начиная от корневого и заканчивая тем, в котором непосредственно хранится файл.

Вот всем знакомая сказочная аналогия понятия "путь к файлу": "На дубе висит сундук, в сундуке - заяц, в зайце - утка, в утке - яйцо, в яйце - игла, на конце которой смерть Кощеева".

Последовательно записанные имя логического диска, путь к файлу и имя файла составляют **полное имя файла.**

Если представленная на рис. 1 файловая структура хранится на диске C:, то полные имена некоторых входящих в нее файлов в символике операционных систем MS-DOS и Windows выглядят так:

C:\fin.com

C:\IVANOV\PROGS\progl.pas

C:\PETROV\DATA\task.dat

Таблицаразмещенияфайлов

Сведения о файловой структуре Диска содержатся на этом же

диске в виде таблицы размещения файлов. Используя файловую систему ОС, пользователь может последовательно просматривать на экране содержимое каталогов (папок), продвигаясь по дереву файловой структуры вниз или вверх.

На рис. 2показан пример отображения на экране компьютера дерева каталогов на логическом диске E: (левое окно).

В правом окне представлено содержимое папки ARCON. ")то множество файлов различных типов. Отсюда, например, понятно, что полное имя первого в списке файла следующее:

E:\GAME\GAMES\ARCON\dos4gw.exe

Из таблицы можно получить дополнительную информацию о файлах. Например, файл dos4gw.exe имеет размер 254 556 байтов и был создан 31 мая 1994 года в 2 часа 00 мин.

Найдя в таком списке запись о нужном файле, применяя команды ОС, пользователь может выполнить с ним различные действия: инициализировать программу, содержащуюся В файле; удалить, переименовать, скопировать файл. Выполнять все эти операции вы научитесь на практическом занятии.

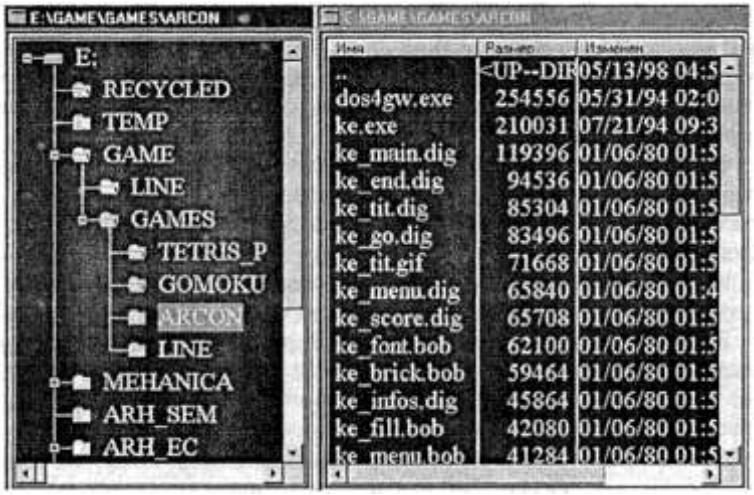


Рисунок 2. Дерево каталогов

Пользовательский интерфейс

Дружественный пользовательский интерфейс

А теперь познакомьтесь с новым для вас понятием

"пользовательский интерфейс".

Разработчики современного программного обеспечения стараются сделать работу пользователя за компьютером удобной, простой, наглядной. Потребительские качества любой программы во многом определяются удобством ее взаимодействия с пользователем.

Форму взаимодействия программы с пользователем называют **пользовательским интерфейсом**. Удобная для пользователя форма взаимодействия называется дружественным пользовательским интерфейсом.

Объектно-ориентированный интерфейс

Интерфейс современных системных и прикладных программ носит название объектно-ориентированного интерфейса. Примером операционной системы, в которой реализован объектно-ориентированный подход, является Windows.

Операционная система работает с множеством объектов, к числу которых относятся: документы, программы, дисководы, принтеры и другие объекты, с которыми мы имеем дело, работая в операционной системе.

Документы содержат некоторую информацию: текст, звук, картинки и т. д. Программы используются для обработки документов. Отдельные программы и документы неразрывно связаны между собой: текстовый редактор работает с текстовыми документами, графический редактор - с фотографиями и иллюстрациями, программа обработки звука позволяет записывать, исправлять и прослушивать звуковые файлы.

Документы и программы - это информационные объекты. А такие объекты, как дисководы и принтеры, являются аппаратными (физическими) объектами. С объектом операционная система связывает:

имя;
графическое обозначение;
свойства;
поведение.

В интерфейсе операционной системы для обозначения документов, программ, устройств используются значки (их еще называют пиктограммами, иконками) и имена. Имя и значок дают возможность легко отличить один объект от другого (рис. 3).

Рисунок 3. Имена и значки различных объектов в операционной системе Windows

С каждым объектом связан определенный набор свойств и множество действий, которые могут быть выполнены над объектом.

Например, свойствами документа являются его местоположение в файловой структуре и размер. Действия над документом: открыть (просмотреть или прослушать), переименовать, напечатать, скопировать, сохранить, удалить и др.

Контекстное меню

Операционная система обеспечивает одинаковый пользовательский интерфейс при работе с разными объектами. В операционной системе Windows для знакомства со свойствами объекта и возможными над ним действиями используется контекстное меню (рис. 4) (для вызова контекстного меню следует выделить значок объекта и щелкнуть правой кнопкой мыши).

Меню - это выводимый на экран список, из которого пользователь может выбирать нужный ему элемент.

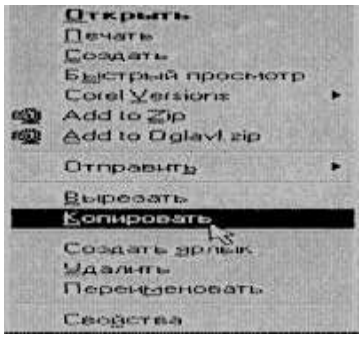
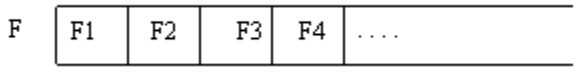


Рисунок 4. Контекстное меню документа

В меню на рис. 4 все пункты, кроме последнего, относятся к действиям, которые можно выполнить с документом. Выбор нужного пункта меню производится с помощью клавиш управления курсором или манипулятора (например, мыши). Если выбрать пункт меню "Свойства", то на экран будет выведен список свойств данного объекта.

Работа с файлами на языке высокого уровня (на примере Pascal-Паскаль)

Существенной особенностью всех рассмотренных до сих пор значений производных типов является наличие в них конечного, наперед заданного числа компонент. Так, в значении многомерного массива это число можно определить, зная количество компонент по каждому измерению, а в значении записи это число определяется количеством и типом полей. Таким образом, заранее, еще до выполнения программы, по этому описанию можно выделить необходимый объем памяти машины для хранения значений переменных этих типов. Но существует определенный класс задач и определенные ситуации, когда количество компонент (пусть даже одного и того же из известных уже типов) заранее определить невозможно, оно выясняется только в процессе решения задачи. Поэтому возникает необходимость в специальном типе значений, которые представляют собой произвольные последовательности элементов одного и того же типа, причем длина этих последовательностей заранее не определяется, а конкретизируется в процессе выполнения программы. Этот тип значений получил название **файлового типа**. Условно **файл в Паскале** можно изобразить как некоторую ленту, у которой есть начало, а конец не фиксируется. Элементы файла записываются на эту ленту последовательно друг за другом:



где F – имя файла, а F1, F2, F3, F4 – его элементы. Файл во многом напоминает магнитную ленту, начало которой заполнено записями, а конец пока свободен. В программировании существует несколько разновидностей файлов, отличающихся методом доступа к его компонентам:**файлы последовательного доступа** и **файлы произвольного доступа.**

Простейший метод доступа состоит в том, что по файлу можно двигаться только последовательно, начиная с первого его элемента, и,

кроме этого, всегда существует возможность начать просмотр файла с его начала. Таким образом, чтобы добраться до пятого элемента файла, необходимо, начав с первого элемента, пройти через предыдущие четыре. Такие файлы называют **файлами последовательного доступа**. У последовательного файла доступен всегда лишь очередной элемент. Если в процессе решения задачи необходим какой-либо из предыдущих элементов, то необходимо вернуться в начало файла и последовательно пройти все его элементы до нужного.

Файлы произвольного доступа Паскаля позволяют вызывать компоненты в любом порядке по их номеру.Важной особенностью файлов является то, что данные, содержащиеся в файле, переносятся на внешние носители. Файловый тип Паскаля – это единственный тип значений, посредством которого данные, обрабатываемые программой, могут быть получены извне, а результаты могут быть переданы во внешний мир. Это единственный тип значений, который связывает программу с внешними устройствами ЭВМ.

Любой файл имеет три характерные особенности. Во-первых, у него есть имя, что дает возможность программе работать одновременно с несколькими файлами. Во-вторых, он содержит компоненты одного типа. Типом компонентов может быть любой тип Паскаля, кроме файлов. Иными словами, нельзя создать «файл файлов». В-третьих, длина вновь создаваемого файла никак не оговаривается при его объявлении и ограничивается только емкостью устройств внешней памяти.

Файловый тип или переменную файлового типа в Паскале можно задать одним из трех способов:

Type <имя_ф_типа>=file of<тип_элементов>;
<имя_ф_типа>=text;
<имя_ф_типа>=file;

Здесь <имя_ф_типа> – имя файлового типа (правильный идентификатор); File, of – зарезервированные слова (файл, из); <тип_элементов> – любой тип Паскаля, кроме файлов.

Пример описания файлового типа в Паскале

```
Type
   Product= record
      Name: string;
      Code: word;
   End;
   Text80= file of string[80];
Var
   F1: file of char;
   F2: text;
   F3: file;
   F4: Text80;
   F5: file of Product;
```

В зависимости от способа объявления можно выделить три вида файлов Паскаля:

- типизированные файлы Паскаля(задаются предложением file of..);
- текстовые файлы Паскаля(определяются типом text);
- нетипизированные файлы Паскаля(определяются типом file).

Следует помнить, что физические файлы на магнитных дисках и переменные файлового типа в программе на Паскале – объекты различные. Переменные файлового типа в Паскале могут соответствовать не только физическим файлам, но и логическим устройствам, связанным с вводом/выводом информации. Например, клавиатуре и экрану соответствуют файлы со стандартными именами Input, Output.

Как известно, каждый тип данных в Паскале, вообще говоря, определяет множество значений и множество операций над значениями этого типа. Однако над значениями файлового типа Паскаля не определены какие-либо операции, в том числе операции отношения и присваивания, так что даже такое простое действие, как присваивание значения одной файловой переменной другой файловой переменной, имеющей тот же самый тип, запрещено. Все операции могут производиться лишь с элементами (компонентами) файлов. Естественно, что множество операций над компонентами файла определяется типом компонент.

Переменные файлового типа используются в программе только в качестве параметров собственных и стандартных процедур и функций.

Основные процедуры и функции для работы с файлами:

1.До начала работы с файлами в Паскале необходимо установить связь между файловой переменной и именем физического дискового файла:

Assign(<файловая_переменная>, <имя_дискового_файла>)

Следует помнить, что имя дискового файла при необходимости должно содержать путь доступа к этому файлу, включая имя дисковода. При этом имя дискового файла – строковая величина, т.е. должна быть заключена в апострофы. Например:

Assign (chf, 'G:\Home\ Student\ Lang\ Pascal\ primer.dat').

Если путь не указан, то программа будет искать файл в своем рабочем каталоге и по указанным путям в autoexec.bat.

Вместо имени дискового файла можно указать имя логического устройства, каждое из которых имеет стандартное имя:

CON – консоль, т.е. клавиатура-дисплей;

PRN – принтер. Если к компьютеру подключено несколько принтеров, доступ к ним осуществляется по именам LPT1, LPT2, LPT3.

Не разрешается связывать с одним физическим файлом более одной файловой переменной.

2.После окончания работы с файлами на Паскале, они должны быть закрыты.

Close(<список файловых переменных>);

При выполнении этой процедуры закрываются соответствующие физические файлы и фиксируются сделанные изменения. Следует иметь в виду, что при выполнении процедуры close связь файловой переменной с именем дискового файла, установленная ранее процедурой assign, сохраняется, следовательно, файл можно повторно открыть без дополнительного использования процедуры assign.

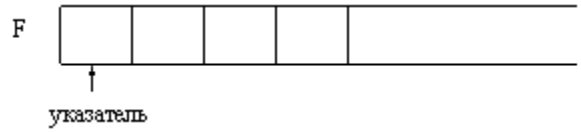
Работа с файлами заключается, в основном, в записи элементов в файл и считывании их из файла. Для удобства описания этих

процедур введем понятие указателя, который определяет позицию доступа, т.е. ту позицию файла, которая доступна для чтения (в режиме чтения), либо для записи (в режиме записи). Позиция файла, следующая за последней компонентой файла (или первая позиция пустого файла) помечается специальным маркером, который отличается от любых компонент файла. Благодаря этому маркеру определяется конец файла.

3.Подготовка к записи в файл Паскаля

Rewrite(<имя_ф_переменной>);

Процедура Rewrite(f) (где f – имя файловой переменной) устанавливает файл с именем f в начальное состояние режима записи, в результате чего указатель устанавливается на первую позицию файла. Если ранее в этот файл были записаны какие-либо элементы, то они становятся недоступными. Результат выполнения процедуры rewrite(f); выглядит следующим образом:

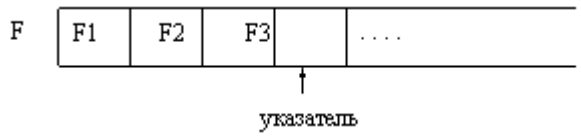


4.Запись в файл Паскаля

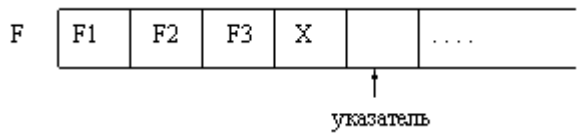
Write(<имя_ф_переменной>, <список записи>);

При выполнении процедуры write(f, x) в ту позицию, на которую показывает указатель, записывается очередная компонента, после чего указатель смещается на следующую позицию. Естественно, тип выражения x должен совпадать с типом компонент файла. Результат действия процедуры write(f, x) можно изобразить так:

Состояние файла f до выполнения процедуры



Состояние файла f после выполнения процедуры



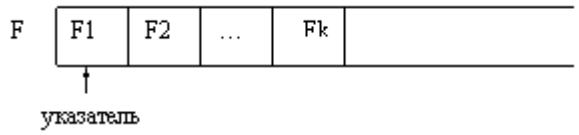
Для типизированных файлов выполняется следующее утверждение: если в списке записи перечислено несколько выражений, то они

записываются в файл, начиная с первой доступной позиции, а указатель смещается на число позиций, равное числу записываемых выражений.

5.Подготовка файла к чтению Паскаля

Reset(<имя_ф_переменной>);

Эта процедура ищет на диске уже существующий файл и переводит его в режим чтения, устанавливая указатель на первую позицию файла. Результат выполнения этой процедуры можно изобразить следующим образом:



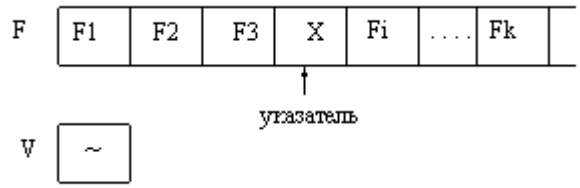
Если происходит попытка открыть для чтения не существующий еще на диске файл, то возникает ошибка ввода/вывода, и выполнение программы будет прервано.

6.Чтение из файла в Паскале

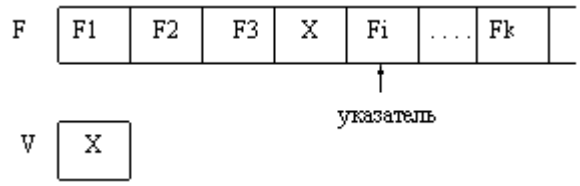
Read(<имя_ф_переменной>,<список переменных>);

Рассмотрим результат действия процедуры read(f, v):

Состояние файла f и переменной v до выполнения процедуры:



Состояние файла f и переменной v после выполнения процедуры:



Для типизированных файлов при выполнении процедуры read() последовательно считывается, начиная с текущей позиции указателя, число компонент файла, соответствующее числу переменных в списке, а указатель смещается на это число позиций.

В большинстве задач, в которых используются файлы, необходимо последовательно перебрать компоненты и произвести их обработку.

В таком случае необходимо иметь возможность определять, указывает ли указатель на какую-то компоненту файла, или он уже вышел за пределы файла и указывает на маркер конца файла.

7.Функция определения достижения конца файла в Паскале

Eof(<имя_ф_переменной>);

Название этой функции является сложносокращенным словом от end of file. Значение этой функции имеет значение true, если конец файла уже достигнут, т.е. указатель стоит на позиции, следующей за последней компонентой файла. В противном случае значение функции – false.

8.Изменение имени файла в Паскале

Rename(<имя_ф_переменной>, <новое_имя_файла>);

Здесь новое_ имя_ файла – строковое выражение, содержащее новое имя файла, возможно с указанием пути доступа к нему.

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был открыт.

9.Уничтожение файла в Паскале

Erase(<имя_ф_переменной>);

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был открыт.

10.Уничтожение части файла от текущей позиции указателя до конца в Паскале

Truncate(<имя_ф_переменной>);

11.Файл Паскаля может быть открыт для добавления записей в конец файла

Append(<имя_ф_переменной>);

Типизированные файлы Паскаля. Длина любого компонента типизированного файла строго постоянна, т.к. тип компонент определяется при описании, а, следовательно, определяется объем памяти, отводимый под каждую компоненту. Это дает возможность организовать прямой доступ к каждой компоненте (т.е. доступ по порядковому номеру).

Перед первым обращением к процедурам ввода/вывода указатель файла стоит в его начале и указывает на его первый компонент с номером 0. После каждого чтения или записи указатель сдвигается к

следующему компоненту файла. Переменные и выражения в списках ввода и вывода в процедурах **read() и write()** должны иметь тот же тип, что и компоненты файла Паскаля. Если этих переменных или выражений в списке несколько, то указатель будет смещаться после каждой операции обмена данными на соответствующее число позиций.

Для облегчения перемещения указателя по файлу и доступа к компонентам типизированного файла существуют специальные процедуры и функции:

fileSize(<имя_ф_переменной>) – функция Паскаля, определяющая число компонентов в файле;

filePos(<имя_ф_переменной>) – функция Паскаля, значением которой является текущая позиция указателя;

seek(<имя_ф_переменной>,n) – процедура Паскаля, смещающая указатель на компоненту файла с номером n. Так, процедура seek(<имя_ф_переменной>,0) установит указатель в начало файла, а процедура seek(<имя_ф_переменной>, FileSize(<имя_ф_переменной>)) установит указатель на признак конца файла.

Текстовые файлы Паскаля. Текстовые файлы предназначены для хранения текстовой информации. Именно в таких файлах хранятся, например, исходные тексты программ. Компоненты текстовых файлов могут иметь переменную длину, что существенно влияет на характер работы с ними. Доступ к каждой строке текстового файла Паскаля возможен лишь последовательно, начиная с первой. К текстовым файлам применимы процедуры assign, reset, rewrite, read, write и функция eof. Процедуры и функции seek, filepos, filesize к ним не применяются. При создании текстового файла в конце каждой записи (строки) ставится специальный признак EOLN(end of line – конец строки). Для определения достижения конца строки существует одноименная логическая функция EOLN(<имя_ф_переменной>), которая принимает значение true, если конец строки достигнут.

Форма обращения к процедурам write и read для текстовых и типизированных файлов одинакова, но их использование принципиально различается.

В списке записываемых в текстовый файл элементов могут чередоваться в произвольном порядке числовые, символьные, строковые выражения. При этом строковые и символьные элементы записываются непосредственно, а числовые из машинной формы автоматически преобразуются в строку символов.

- текстовые файлы удобнее для восприятия человеком, а типизированные соответствуют машинному представлению объектов;
- текстовые файлы, как правило, длиннее типизированных;
- длина текстовых файлов зависит не только от количества записей, но и от величины переменных.

Так, в типизированном файле числа 6, 65 и 165 как целые будут представлены одним и тем же числом байт. А в текстовых файлах, после преобразования в строку, они будут иметь разную длину. Это вызывает проблемы при расшифровке текстовых файлов. Пусть в текстовый файл пишутся подряд целые числа (типа byte): 2, 12, 2, 128. Тогда в файле образуется запись 2122128. При попытке прочитать из такого файла переменную типа byte программа прочитает всю строку и выдаст сообщение об ошибке, связанной с переполнением диапазона.

Но, вообще-то, такой файл не понимает не только машина, а и человек.

Чтобы избежать этой ошибки, достаточно вставить при записи в файл после каждой переменной пробел. Тогда программа при каждом чтении берет символы от пробела до пробела и правильно преобразует текстовое представление в число.

Кроме процедур read и write при работе с текстовыми файлами используются их разновидности readln и writeln. Отличие заключается в том, что процедура writeln после записи заданного списка записывает в файл специальный маркер конца строки. Этот признак воспринимается как переход к новой строке. Процедура

readln после считывания заданного списка ищет в файле следующий признак конца строки и подготавливается к чтению с начала следующей строки.

Пример решения задачи с файлами Паскаля

Пусть нам необходимо сформировать текстовый файл с помощью Паскаля, а затем переписать из данного файла во второй только те строки, которые начинаются с буквы «А» или «а».

Пояснения: нам понадобятся две файловые переменные f1 и f2, поскольку оба файла текстовые, то тип переменных будет text. Задача разбивается на два этапа: первый – формирование первого файла; второй – чтение первого файла и формирование второго.

Для завершенности решения задачи есть смысл добавить еще одну часть, которая в задаче явно не указана – вывод на экран содержимого второго файла.

Пример процедуры Assign Паскаля

```
Program primer;
Var f1,f2:text;
   I,n: integer;
   S: string;
Begin
{формируем первый файл}
   Assign(f1, ‘file1.txt’); {устанавливаем связь файловой переменной с
физическим файлом на диске}
   Rewrite(f1);  {открываем файл для записи}
   Readln(n) {определим количество вводимых строк}
   for i:=1 to n do
   begin       readln(s); {вводим с клавиатуры строки}
      writeln(f1,s); {записываем последовательно строки в файл}
   end;
   close(f1); {заканчиваем работу с первым файлом, теперь на диске существует
файл с именем file1.txt, содержащий введенные нами строки. На этом
программу можно закончить, работу с файлом можно продолжить в другой
программе, в другое время, но мы продолжим}
{часть вторая: чтение из первого файла и формирование второго}
   Reset(f1); {открываем первый файл для чтения}
   Assign(f2, ‘file2.txt’); {устанавливаем связь второй файловой переменной с
```

```
физическим файлом}
   Rewrite(f2); {открываем второй файл для записи}
 {Дальше необходимо последовательно считывать строки из первого файла,
проверять выполнение условия и записывать нужные строки во второй файл.
Для чтения из текстового файла рекомендуется использовать цикл по условию
«пока не конец файла»}
   While not eof(f1) do
   Begin
      Readln(f1,s);{считываем очередную строку из первого файла}
      If (s[1]='A') or (s[1]='a') then
      Writeln(f2,s); {записываем во второй файл строки, удовлетворяющие
условию}
   End;
   Close(f1,f2); {заканчиваем работу с файлами}
 {часть третья: выводим на экран второй файл}
   Writeln;
   Writeln('Второй файл содержит строки:');
   Reset(f2); {открываем второй файл для чтения}
   While not eof(f2) do {пока не конец второго файла}
   Begin
      Readln(f2,s);{считываем очередную строку из второго файла}
      Writeln(s); {выводим строку на экран}
   End;
End.
```

Работа с файлами на языке Java

Язык программирования Java представляет обширный механизм для доступа и манипуляцией файлами и элементами директорий (папок). Несмотря на огромное обилие операционных систем, данный язык представляет универсальный механизм работы с файлами для любых типов файловых систем: FAT16, FAT32, NTFS, EXT3 и.т.д. Программы на языке Java для доступа к файлам обычно используют классы, описанные в библиотеке java.io. В этом разделе мы рассмотрим толькор самые необходимые классы: File, FileReader, FileWriter, FileInputStream, FileOutputStream.

Класс "File" является полезным инструментом для получения информации о файлах и директориях компьютера. Объекты этого

класса сами по себе не открывают файлы и не извлекают из них информацию, в тоже время взаимодействуя с операционной системой они позволяют извлечь необходимую для пользователя информацию

Методы класса "File"	Описание
boolean canRead()	Возвращает true, если из файла может производиться чтение и false в противном случае
boolean canWrite()	Возвращает true, если имеется возможность производить запись в файл и false в противном случае
boolean exists()	Возвращает true, если указанный файл существует и false в противном случае
boolean isFile()	Возвращает true, если по указанному пути находится файл и false, если это директория
boolean isDirectory()	Возвращает true, если по указанному пути находится директория и false, если это файл
boolean isAbsolute()	Returns true if the arguments specified to the File constructor indicate an absolute path to a file or directory; false otherwise.
String getName()	Возвращает имя указанного файла
String getPath()	Возвращает путь (место расположения) файла
String getParent()	Возвращает имя директории, в которой файл находится
long length()	Возвращает размер (количество байт) указанного файла
long lastModified()	Возвращает дату и время последнего изменения файла
String[] list()	Возвращает список файлов, которые находятся в указанной директории

Example 1 illustrates how to use the class File to get information about the file. New componets used in the example are the class JFileChooser that displays a dialog (known as the JFileChooser dialog) that enables the user to easily select files or directories and the class JTextArea that allows to represent many linesa of text inside a frame.

Example 1. Test of "File" class using GUI application

```
import java.awt.*;
import java.io.File;
import javax.swing.*;

publicclassExample1extends JFrame
{
  JTextArea outputArea = new JTextArea();//1
```

```
   JScrollPane scrollPane=new JScrollPane( outputArea );//2
   JFileChooser fileChooser = new JFileChooser();//3

public Example1()
   {  this.setLayout(new BorderLayout());
this.add( scrollPane, BorderLayout.CENTER ); //4
this.setSize( 400, 400 ); // 5 set GUI size
this.setVisible( true ); //6 display GUI
fileChooser.setFileSelectionMode(//7
                JFileChooser.FILES_AND_DIRECTORIES );
intresult = fileChooser.showOpenDialog( this );//8
if ( result == JFileChooser.CANCEL_OPTION )//9
      System.exit( 1 );//10
      File name = fileChooser.getSelectedFile(); // получитьсписокфайлов
if ( name.exists() ) ;// Еслифайлсуществует, товывести о неминформацию
      String text="Информация о файле:\n";
text=text+name.getName()+" существует\n";
text=text+"Этофайл? "+name.isFile()+"\n";
text=text+"Этодиректория? "+name.isDirectory()+"\n";
text=text+"Длиннафайла: "+name.length()+"\n";
outputArea.setText(text);
if ( name.isDirectory() ) // output directory listing
         {  String[] directory = name.list();
outputArea.append( "\n\nСодержимоедиректории:\n" );
for ( String directoryName : directory )
outputArea.append( directoryName + "\n" );
        } // end else
      } // end outer if
// end FileDemonstration constructor

publicstaticvoid main( String[] args )
   {
      Example1 application = new Example1();

   } // конецглавнойпрограммы
} // конец класса
```

Внешний вид программы представлен на рисунке ниже:

При выборе папки C:/Windows программа выводит следующую информацию:
Информация о файле:
Windows существует
Это файл? false
Это директория? true
Длинна файла: 0

Содержимое директории:
activator.exe
addins
AppCompat
AppPatch
ARJ.PIF
assembly
atiogl.xml
atisamu32.dll
ativpsrm.bin
bfsvc.exe
иещесотнифайлов

Классы**FileInputStreamandFileOutputStream**предназначеныдляработ ысбинарнымифайлами (файлы, представленныекакмассивбайт),классы**FileReaderandFileWriter**позво ляютработатьстекстовымифайлами.Класс**FileReader**позволяет читать символы из текстового файла. Данный класс содержит следующие методы

read() - вернуть один символ из файла
ready() - возвращает true, если конец файла еще не достигнут
read(char[] cbuf) - читать все символы текстового файла в массив "cbuf"
skip(longn) - пропустить n символов
close() - закрыть файл

Класс**FileWriter**предназначен для записи в текстовые файлы. Содержит следующие методы:
append (charc) - добавить символ "c" в конец текстового файла
write(charc) - записать в файл символ "c"
write(char[] cbuf,intoff, intlen); записать в файл "len"символов из массива cbuf, используя смещение от его начала заданное параметром "off"

Класс**FileInputStream**читает информацию из файла побайтно. Класс содержит следующие методы:
available(); Возвращает число байт информации, которые можно прочитать из файла. При открытии файла возвращает длинну файла, в процессе чтения происходит сдвиг невидимого указателя и количество доступных байт уменьшается. Если этот параметр равен нулю, это значит, что достигнут конец файла.
read(); прочитать 1 байт информации из файла
read(byte[] b); Заполнить массив b данными из файла. Количество прочитанных байт будет равно длинне массива.
read(byte[] b, intoff, intlen); Прочитать "len" байт из файла и внести данные в массив "b", начиная с индекса, заданного числом "off".
skip(longn); Пропустить "n" байт, сместив указатель

Класс**FileOutputStream**записывает последовательность байт в файл. Данный класс содержит следующие методы:
close(); Закрыть файл.

flush(); Произвести запись накопленной информации в файл, но не закрывать его.
write(intb); Записать число b в файл.
write(byte[] b); Записать массив b в файл. Количество записанных байт будет равно длинне массива b.
write(byte[] b, intoff, intlen); Данный метод записывает "len" байт в файл, используя данные массива b, начиная с индекса off.

Пример программы, которая читает текст с клавиатуры и записывает его в файл

Порядок выполнения работы.

1. Изучите теоретический материал.
2. Создайте в текстовом редакторе таблицу. Скопируйте ее в Excel. Сделайте вывод.
3. Создайте текстовый файл в блокноте. Откройте его в Word. Сделайте вывод.
4. Создайте формулы и рисунок как объект в Word версии 10 и выше. Откройте его в версии ниже 10. Сделайте вывод.
5. Создайте рисунок в текстовом файле как объект. Откройте его в Paint. Сделайте вывод.
6. Напишите и отладьте программу, выполняющую следующие действия: осуществляется расчет значений заданной функции в определенных интервале и шагом изменения аргумента, результат запоминается в текстовом файле. Подумайте, как прочитать этот файл в Блокноте и Word.
7. Напишите программу на языке высокого уровня импортирования и экспортирования файла из электронной таблице в массив, определенном в оболочке языка высокого уровня.
8. Создайте и отладьте на языке высокого уровня программу, формирующую простейший текстовый файл и осуществляет его последовательное чтение, запись в котором состоит из переменных разных типов (например, ФИО, год рождения, номер медицинской карты, дата заполнения).
9. Научитесь выполнять основные файловые операции в используемой ОС (копирование, перемещение, удаление, переименование файлов).
9. Оформите отчет, включающий описание действий, полученных результатов, кратких ответов на не менее чем трех контрольных вопросов.

Контрольные вопросы.

1. Что называется файлом? Как задается структура файла? Как осуществляется поиск информации в файле?
2. Как осуществляется поиск файла в каталоге и на языке высокого уровня?
3. Почему файлы с разным расширением в имени несовместимы? Как можно решить эту проблему?
4. Что выполняют процедуры открытия и закрытия файла?
5. В чем заключается назначение файловых менеджеров?
6. Как можно определить наличие вируса в файле?
7. Каким образом можно защитить файл от несанкционированного доступа?
8. Что такое типизированный и нетипизированный файл?
9. Как можно разместить один файл на нескольких носителях информации (если он не помещается на один)?
10. Каким образом можно прочесть содержимое файла по байта?
11. Как можно найти файл по контексту, если имя файла неизвестно (в менеджере, операционной системе и на языке высокого уровня)?
12. Каким образом можно создать семантическую структуру файлов, связав их по определенным элементам?
13. Какую файловую структуру использует операционная система на ваших компьютерах (простую, многоуровневую)? Что это такое?
14 Что такое пользовательский интерфейс операционной системы?
15. Чем характеризуется объект-файл (с точки зрения объектно-ориентированного подхода)?
16. Каким образом можно узнать свойства объекта или выполнить действие над ним?

ЛАБОРАТОРНАЯ РАБОТА № 5. ИТЕРАЦИЯ И РЕКУРСИЯ: НЕОБХОДИМОСТЬ И ОРГАНИЗАЦИЯ

Цель работы: овладение навыками организации итерационных вычислительных процессов с использованием электронных таблиц и языков программирования высокого уровня.

Краткие теоретические сведения.

Сущность метода итераций заключается в построении рекуррентной последовательности чисел, сходящейся к решению, по формуле $x_{к+1} = u(x_к)$, к=0,1,2,..., где x_0=[a,b] -произвольная точка.

Итерация - это цикл, участок программного кода, который выполняется несколько раз подряд до тех пор, пока не выполнится определённое условие, или же до бесконечности. Итерации используются для того, чтобы повторить то или иное действие несколько раз, как для разных объектов, так и для одного и того же. Счётчик цикла - это переменная, которая указывает на то, сколько раз надо повторить данные действия, и позволяет подсчитать, какой по счёту заход (т. е. итерация) выполняется в данный момент.

Итерация (часто называема «простая итерация») часто используется при решении алгебраических уравнений. Методы решения систем линейных алгебраических уравнений классифицируют на прямые (точные) и итерационные. Прямые методы основаны на выполнении конечного числа арифметических операций, это, например, метод обратной матрицы, метод Гаусса, метод Гаусса-Жордана, метод прогонки для трехдиагональных матриц и т.д. Суть итерационных методов, в свою очередь, заключаются в том, чтобы за счет последовательных приближений получить решение системы, определяемое необходимой точностью.

Для итерационных методов можно выделить три последовательных этапа:

1. Приведение исходной системы вида $\bar{A} \times \bar{x} = \bar{b}$ к итерационной форме $\bar{x} = \bar{C} * \overline{x^0} + \bar{d}$.

2. Проверка условия сходимости.
3. Решение системы одним из методов.

Метод простых итераций

Условимся, что для общего вида систем выполняется тождество $m=n$, где m - количество уравнений в системе, n - количество неизвестных. Т.е. не имеет смысла решать недоопределенные ($m<n$) и переопределенные ($m>n$) системы, т.к. они могут быть сведены путем элементарных алгебраических преобразований к нормальным ($m=n$) системам линейных уравнений. Другими словами, если имеется «ненормальная» система, то прежде, чем использовать метод простых итераций, преобразуйте ее к нормальной.

Система линейных уравнений может быть записана в матричной форме, где A – матрица коэффициентов, b – вектор свободных членов, x – вектор неизвестных. Возьмем систему:

$$\begin{cases} 5x_1-4x_2-x_3=-2 \\ 4x_1+x_2-2x_3=8 \\ 3x_1+x_2-5x_3=10 \end{cases}$$

Ее матричная форма:

$$\begin{pmatrix} 5 & -4 & -1 \\ 4 & 1 & -2 \\ 3 & 1 & -5 \end{pmatrix} * \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} -2 \\ 8 \\ 10 \end{pmatrix}$$

Переход к итерационному виду осуществляется по следующим формулам:

$c_{ij} = -a_{ij}/a_{ii}$ $d_i = b_i/a_{ii}$, где i, j = 1,2,3…

Также следует отметить, что, несмотря на эти формулы, диагональные элементы новой матрицы обнуляются, хотя должны быть равны -1.

В итоге:

$$\begin{pmatrix} 0 & -0.25 & 0.5 \\ 0.2 & 0 & 0.2 \\ 0.6 & 0.2 & 0 \end{pmatrix} * \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -2 \end{pmatrix}$$

Некоторые элементы матрицы C будут больше единицы. А глядя на условие сходимости, становится понятно, что, если хотя бы один будет больше единицы, то условие не выполнится, и решение системы путем простых итераций не будет найдено.

Поэтому открываем учебник по линейной алгебре и вспоминаем элементарные матричные преобразования! Прежде чем следовать этапам итерационных методов, нужно привести исходную систему к виду, в котором все диагональные элементы были бы максимальными по модулю в своих строках. Только при таком виде матрицы коэффициентов можно надеется на выполнение условия сходимости.

Смотрим начальную систему. Видим, что третий элемент третьей строки по модулю больше других. Оставим его неизменным. Меняем местами первую и вторую строки. Теперь умножаем строку, ставшую первой, на -1 и складываем с новой второй. В итоге получаем:

$$\begin{pmatrix} 4 & 1 & -2 \\ 1 & -5 & 1 \\ 3 & 1 & -5 \end{pmatrix} * \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 8 \\ -10 \\ 10 \end{pmatrix}$$

Теперь при подстановке в формулы мы получим итерационную форму верно. К сожалению, это преобразование начальной системы к "благоприятному" виду - чистая аналитика, поэтому записать его в программный код очень сложно, а если даже и попытаться, то в некоторых случаях вероятно возникновение ошибок.

Переходим ко второму этапу: "Проверка условия сходимости" (формулу смотрите выше). Если система не проходит проверку, то приближения не будут сходиться к реальному решению, и ответ получен не будет. В этом случае можно попытаться получить другую "благоприятную" форму. Если условие сходимости выполнено, то стратегия метода простых итераций применима и осуществляется переход к третьему этапу.

В конечном счете, получаем систему линейных алгебраических уравнений в итерационной форме:

$$\begin{cases} x_1=0x^0_1-0.25x^0_2+0.5x^0_3+2 \\ x_2=0.2x^0_1+0x^0_2+0.2x^0_3+2 \\ x_3=0.6x^0_1+0.2x^0_2+0x^0_3-2 \end{cases},$$

где x_1, x_2, x_3 – приближения, получаемые на текущей итерации за счет приближений полученных на предыдущей итерации - x^0_1, x^0_2, x^0_3.
Итерационный процесс в методе простых итераций идет до тех пор, пока вектор приближений не достигнет заданной точности, т.е. пока не выполнится условие выхода:
$Max|x_i-x^0_i|<\varepsilon$, где ε – требуемая точность.
Далее представлен код, написанный в среде **PascalABC.Net**.

```
Program MetodProstIter; {метод простых итераций}
Var
n:integer; {количество переменных или количество уравнений, как кому удобно
- }
        {в любом случае они должны быть равны (m=n)}
A:array of array of real; {матрица коэффициентов}
b:array Of real; {вектор-столбец свободных членов}
C:array of Array of real; {матрица Якоби - итерационная форма матрицы A}
d:array of real; {итерационная форма вектора свободных членов}
Err:Boolean; {переменная, по значению которой после выполнения процедуры
проверки}
          {сходимости определяется соответствие-несоответствие условию
сходимости}
X:array of Real; {вектор неизвестных}

procedure InputA(var n:integer); {ввод матрицы A}
var
i,j:Integer;
begin
  SetLength(A,n); {именно эта встроенная процедура задает правую границу
массива}
              {в зависимоти от количества переменных}
  for i:=0 To n-1 Do
  begin
    SetLength(A[i],n); {многомерные массивы в PascalABC можно определять как
массивы массивов}
```

```
  end;
  for i:=0 To n-1 Do
  begin    for j:=0 To n-1 Do     read(A[i,j]);    writeln('');  end;
end;

procedure InputB(var n:integer); {вводвектора B}
var
i:Integer;
beginSetLength(b,n);for i:=0 To n-1 Do  readln(b[i]);end;

Procedure IterForm(A:array of Array of real;b:array of real;n:integer);
 {полученииеитерационнойформысистемы}

var i,j:Integer;
begin
  SetLength(C,n);  for i:=0 To n-1 Do    Setlength(C[i],n); end;
  SetLength(d,n);
  for i:=0 To n-1 Do  begin
    for j:=0 To n-1 Do    begin
      if i=j then       C[i,j]:=0      else       C[i,j]:=-A[i,j]/A[i,i];
    end;
    d[i]:=b[i]/a[i,i];
  end;
end;

Procedure ProverkaShodimosti(C:array of array of real;d:array of real;n:integer);
 {проверкасистемынасходимость}
var i,j:Integer;
summ:real;
begin
  summ:=0;  for i:=0 To n-1 Do    for j:=0 To n-1 Do      summ:=summ+C[i,j]*C[i,j];
  if SQRT(Abs(summ))>1 then  begin
    writeln('Даннаясистеманеудовлетворяетусловиюсходимости');    Err:=True;
  end  Else Err:=False;
end;

Procedure ProstIterMetode(C:array of array of real;d:array of real;n:integer);
 {собственно, самастратегияметодапростыхитераций}
var i,j:Integer;
```

```
X0:array of real;
delta:real;
E:array of real;
begin
  SetLength(X0,n);  SetLength(X,n);  Setlength(E,n);
  X0:=Copy(d);
  repeat
  begin    for i:=0 To n-1 Do    begin
      X[i]:=0;
      for j:=0 To n-1 Do      begin
        X[i]:=X[i]+C[i,j]*X0[j];
      end;
      X[i]:=X[i]+d[i];      E[i]:=Abs(X[i]-X0[i]);
    end;
    delta:=E[0];
    for i:=1 To n-1 Do     if delta<E[i] then delta:=E[i];
    X0:=Copy(X);
    end;
  Until delta<=0.000001;
  writeln('решениесистемыравновектору:');
  For i:=0 To n-1 Do    Writeln(X[i]);
end;

BEGIN
Err:=False;
writeln('Введитеколичествопеременных');Readln(n);
writeln('Введитематрицукоэффициентов');InputA(n);
writeln('введитематрицусвободныхчленов');InputB(n);
IterForm(A,b,n);ProverkaShodimosti(C,d,n);
if Err=False then ProstIterMetode(C,d,n);
END.
```

Нахождение корня уравнения методом итераций на языке высокого уровня:

```
Program Confirm_2;
function F(x:real):real;
beginF:=3*sin(sqrt(x))+0.35*x-3.8;end;
label m1, m2, m3;
var
eps,f1,f2,x1,x2,a,b:real;
```

```
begin
writeln('Нахождение корня уравнения 3*sin(sqrt(x))+0.35*x-3.8=0');
writeln('на интервале [7, 8] методом итераций');
writeln;
a:=7;b:=8;eps:=0.00001;{методитераций}
x1:=a; f1:=F(x1);
begin
m3: x2:=x1+eps;f2:=F(x2);if f1*f2 > 0 then goto m1 else goto m2;
m1: x1:=x2;f1:=f2; goto m3;
m2: writeln('по методу итераций: x = ', (x1 + x2)/2 :0:5);
end;
readln;
end.
```

Для вычисления итераций используют алгоритмы-рекурсии.

Реку́рсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в различных специальных областях знаний — от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.

Рассмотрим пример: рекурсионный способ нахождения факториала в паскале

Найти факториал в pascal можно также посредством вызова функции (с помощью рекурсии) .

```
function fact(x:byte):real;
begin if x=0 then fact:=1 else fact:=fact(x-1)*x; end;
```

В математике рекурсия имеет отношение к методу определения функций и числовых рядов: рекурсивно заданная функция определяет своё значение через обращение к себе самой с другими аргументами. При этом возможно два варианта:

Конечная рекурсивная функция. Она задаётся таким образом, чтобы для любого конечного аргумента за конечное число рекурсивных обращений привести к одному из отдельно определённых частных случаев, вычисляемых без рекурсии.

Бесконечная рекурсивная функция. Она задаётся в виде обращения к самой себе во всех случаях (по крайней мере, для некоторых из аргументов). Подобным образом могут задаваться бесконечные ряды, бесконечныенепрерывные дроби и так далее. Практическое вычисление точного значения здесь, естественно, невозможно, поскольку потребует бесконечного времени. Требуемый результат находится аналитическими методами. Тем не менее, если речь идёт не о получении абсолютно точного значения, а о вычислении заданного приближения искомого значения, то тут в некоторых случаях возможно прямое использование рекурсивного определения: вычисления по нему ведутся до тех пор, пока необходимая точность не будет достигнута.

Другим примером рекурсии в математике является *числовой ряд, заданный рекуррентной формулой*, когда каждый следующий член ряда вычисляется как результат функции от n предыдущих членов. Таким образом, с помощью конечного выражения (представляющего собой совокупность реккурентной формулы и набора значений для первых n членов ряда) может даваться определение бесконечного ряда.

С рекурсией тесно связана математическая индукция: она является естественным способом доказательства свойств функций на натуральных числах, рекурсивно заданных через свои меньшие значения.

В математике отдельно рассматривается класс так называемых «примитивно рекурсивных» функций. Определение примитивно рекурсивной функции также рекурсивно, оно задаёт набор примитивных функций и набор правил; функция является примитивно рекурсивной, если она может быть представлена как комбинация примитивно рекурсивных функций, образованная по этим правилам.

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (*простая рекурсия*) или через другие функции (*сложная* или*косвенная рекурсия)* Количество вложенных вызовов функции или процедуры называется глубиной

рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.Структурно рекурсивная функция на верхнем уровне всегда представляет собой команду ветвления (выбор одной из двух или более альтернатив в зависимости от условия (условий), которое в данном случае уместно назвать «условием прекращения рекурсии»), имеющей две или более альтернативные ветви, из которых хотя бы одна является *рекурсивной* и хотя бы одна — *терминальной*. Рекурсивная ветвь выполняется, когда условие прекращения рекурсии ложно, и содержит хотя бы один рекурсивный вызов — прямой или опосредованный вызов функцией самой себя. Терминальная ветвь выполняется, когда условие прекращения рекурсии истинно; она возвращает некоторое значение, не выполняя рекурсивного вызова. Правильно написанная рекурсивная функция должна гарантировать, что через конечное число рекурсивных вызовов будет достигнуто выполнение условия прекращения рекурсии, в результате чего цепочка последовательных рекурсивных вызовов прервётся и выполнится возврат.

Помимо функций, выполняющих один рекурсивный вызов в каждой рекурсивной ветви, бывают случаи «параллельной рекурсии», когда на одной рекурсивной ветви делается два или более рекурсивных вызова. Параллельная рекурсия типична при обработке сложных структур данных, таких как деревья. Простейший пример параллельно-рекурсивной функции — вычисление ряда Фибоначчи, где для получения значения n-го члена необходимо вычислить (n-1)-й и (n-2)-й.

Реализация рекурсивных вызовов функций в практически применяемых языках и средах программирования, как правило, опирается на механизм стека вызовов — адрес возврата и локальные переменные функции записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно. Оборотной стороной этого довольно простого по структуре

механизма является то, что на каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнение стека вызовов.

Вопрос о желательности использования рекурсивных функций в программировании неоднозначен: с одной стороны, рекурсивная форма может быть структурно проще и нагляднее, в особенности, когда сам реализуемый алгоритм по сути рекурсивен. Кроме того, в некоторых декларативных или чисто функциональных языках (таких как Пролог или Haskell) просто нет синтаксических средств для организации циклов, и рекурсия в них — единственный доступный механизм организации повторяющихся вычислений. С другой стороны, обычно рекомендуется избегать рекурсивных программ, которые приводят (или в некоторых условиях могут приводить) к слишком большой глубине рекурсии. Так, широко распространённый в учебной литературе пример рекурсивного вычисления факториала является, скорее, примером того, как *не надо* применять рекурсию, так как приводит к достаточно большой глубине рекурсии и имеет очевидную реализацию в виде обычного циклического алгоритма.

Имеется специальный тип рекурсии, называемый «хвостовой рекурсией» (структура рекурсивного алгоритма такова, что рекурсивный вызов является последней выполняемой операцией в функции, а его результат непосредственно возвращается в качестве результата функции). Интерпретаторы и компиляторы функциональных языков программирования, поддерживающие оптимизацию кода (исходного или исполняемого), автоматически преобразуют хвостовую рекурсию китерации, благодаря чему обеспечивается выполнение алгоритмов с хвостовой рекурсией в ограниченном объёме памяти. Такие рекурсивные вычисления, даже если они формально бесконечны (например, когда с помощью рекурсии организуется работа командного интерпретатора, принимающего команды пользователя), никогда не приводят кисчерпанию памяти. Однако далеко не всегда стандарты языков программирования чётко определяют, каким именно

условиям должна удовлетворять рекурсивная функция, чтобы транслятор гарантированно преобразовал её в итерацию. Одно из редких исключений — язык Scheme (диалект языка Lisp), описание которого содержит все необходимые сведения.

Теоретически, любую рекурсивную функцию можно заменить циклом и стеком. Однако такая модификация, как правило, бессмысленна, так как приводит лишь к замене автоматического сохранения контекста в стеке вызовов на ручное выполнение тех же операций с тем же расходом памяти. Исключением может быть ситуация, когда рекурсивный алгоритм приходится моделировать на языке, в котором рекурсия запрещена.

Доказательство корректности программ[править | править вики-текст]

В отличие от явно-циклических программ, для доказательства корректности рекурсивных нет необходимости искусственно вводить инвариант. Аналитическое доказательство корректности рекурсивной функции сводится к методу математической индукции, то есть к доказательству следующих утверждений:

1. Корректность использования рекурсивного обращения. Доказывается, что результат, вычисляемый в любой рекурсивной ветви функции, будет верным при условии, что соответствующие рекурсивные вызовы, в свою очередь, вернут верный результат.

2. Корректность всех терминальных ветвей. Доказывается, что все терминальные ветви возвращают верные значения. Как правило, это доказывается элементарно, так как терминальные ветви обычно никаких вычислений не содержат.

3. Достижимость терминальной ветви для любого корректного набора параметров после конечного числа рекурсивных вызовов. Доказывается, что изменение параметров вызова функции, которое производится при рекурсивном обращении, через конечное число рекурсивных вызовов приведёт к одному из наборов параметров, для которых задана терминальная ветвь.

Из суммы первого и второго утверждений следует, что в случае достижения терминальной ветви (а это значит — во всех случаях, когда вычисление функции окажется конечным) функция вернёт правильный результат. Третье положение доказывает, что конечным будет любое вычисление. Следовательно, любой вызов функции с корректными параметрами вернёт правильный результат (с очевидной оговоркой — если глубина рекурсии не окажется настолько большой, что вызовет переполнение памяти).

Примерырекурсий

```
void func(int n, long x, long y) {
	if (x>=deg(n)+1 && y<=deg(n)) {	if (n>1) {cout << "WHITE";	return;
	}
		else{	cout << "BLACK";			return;		}
					}
	if (n % 2 == 0 && x<=deg(n) && y>=deg(n)+1) {	cout		<<
"BLACK";return;}
	if (n % 2 != 0 && x<=deg(n) && y>=deg(n)+1) { cout << "WHITE";return;}
	if (n==1 && (x==1 && y==1 || x==2 && y==2)) {	cout		<<
"BLACK";return;}
	if (n>1) {if (x>=deg(n)+1 && y>=deg(n)+1) {func(n-1,x-deg(n),y-deg(n));}
		elsefunc(n-1,x,y);}}
void func(long l, long r) {if (r==l) {	return;	}else {	long mid = (r-l)/2;for
(long i=l; i<=l+mid; i++) {long temp = ab[i];	ab[i]=ab[r+l-i];ab[r+l-i]=temp;}
		func(l, l+mid);func(l+mid+1, r);	}
}
int func(int a, int b) {	int l, r, value, k=0; if (a > b) {	return 0;	}
	if (b-a == 0) {	return ans[a];	}
	else { value=ans[a];	for (int i=a+1; i<=b; i++) {	if	(ans[i]>value)
{value=ans[i];}
					}
		for (int i=a; i<=b; i++) {if (ans[i]==value) {k=i;	break;}
		}		l=func(a,k-1);		r=func(k+1,b);		return
r+value-l;
	}
}
```

Классическим примером бесконечной рекурсии являются два поставленные друг напротив друга зеркала: в них образуются два коридора из уменьшающихся отражений зеркал.

Другим примером бесконечной рекурсии является эффект самовозбуждения (положительной обратной связи) у электронных схем усиления, когда сигнал с выхода попадает на вход, усиливается, снова попадает на вход схемы и снова усиливается. Усилители, для которых такой режим работы является штатным, называются автогенераторы.

Рекурсивная модель — новое направление когнитивной психологии, предназначенное для описания функционирования психики в самых различных ситуациях социального взаимодействия; в частности, рекурсивная модель призвана моделировать то, как субъект оценивает ситуации, принимает решения, переосмысливает прошлый опыт и т.п.

Для рекурсивной модели характерна гибкость описания ситуаций, т.к. её конструкты, в отличие от конструктов большинства психологических теорий, строго не фиксированы, они создаются каждый раз под конкретную ситуацию согласно определённым правилам.

Рекурсивные процессы всё более и более широко применяются в современной науке. В частности, А. В. Анисимов, развивая идеи рекурсивности при создании систем искусственного интеллекта, открыл новое направление в лингвистике, в основу которого положен алгоритмический анализ и рекурсивный синтез структуры речи.

Несмотря на то, что в лингвистике благодаря рекурсии удалось получить важные результаты, в психологии рекурсия до сих пор систематически не применялась. Таким образом, рекурсивная модель функционирования психики является пионерской.

При разработке рекурсивной модели создан особый, специфический язык, который представляет собой язык современной научной психологии, приспособленный к нуждам рекурсивной модели и дополненный специальным графическим языком.

Рекурсивная модель является инструментом, который открывает для людей фрактально-рекурсивную психологическую реальность, она поможет изменить психику и даже, до некоторой степени, сознание людей. Кроме того, она будет служить основой при разработке психокоррекционных техник, направленных на привлечение дополнительных психических ресурсов к решению психологических проблем, а также на уточнение, преобразование и организацию прошлого опыта.

Рекурсивная модель способна стимулировать активность человека, направленную на выявление скрытых возможностей и опасностей ситуации, и, кроме того, на поиск путей удовлетворения актуализированных потребностей, в том числе, на достижение стоящих перед человеком целей.

Графические рекурсивные алгоритмы

Ситуация, когда алгоритм вызывает себя в качестве вспомогательного, называется рекурсией (это слово происходит от латинского recursio -- возвращение). Рекурсивными бывают подпрограммы-процедуры и подпрограммы-функции. В рекурсивных подпрограммах в теле процедуры или функции имеется вызов этих же подпрограмм. Процесс обращения к себе самому может длиться бесконечно. Для обеспечения остановки устанавливают барьер в виде некоторого условия.

Пример 1. Опишем алгоритм рисования окружности радиуса r с центром в точке (x,y) с четырьмя окружностями вокруг. При этом необходимо знать расстояния (r1) от точки(x,y) до центров окружностей окружения (радиусы орбиты), которые, очевидно равны.

```
procedure krug(x,y,r,stepN,stepMax:integer);
begin {вычисляем координаты центра i-й окружности};
krug(x+r,y,r div 2);krug(x,y+r,r div 2);krug(x-r,y,r div 2);krug(x,y-r,r div 2);
Arc(x,y,r,0,360); {рисуем окружность с центром в точке (x,y) и радиусом r}
end;
```

Таким образом, в описанной процедуре осуществляется вызов ее же самой (рекурсия) в качестве вспомогательной.Если осуществить вызов этой же процедуры из основной программы с начальными параметрами, то рекурсивные вызовы никогда не закончатся и

программа будет работать бесконечно. Для того, чтобы этого не случилось, можно ввести в качестве аргумента процедуры некоторую величину stepN, которая при каждом новом вызове процедуры будет увеличиваться на 1, а в тело процедуры что его операторы должны выполняться только при stepMax>stepN, то есть это условие должно играть роль своеобразной «заглушки», ограничивающей число вызовов.

Ниже приведена программа, полностью решающая поставленную задачу.

В основной программе запрашиваются число уровней stepMax. Центр самой окружности располагается в центре экрана.

```
uses crt,graphABC;
Var stepMax, x, y, r : integer;
procedure krug(x,y,r,stepN,stepMax:integer);
begin
if stepMax>stepN thenbegin {задаём координаты окружностей а так же их размеры}
krug(x+r,y,r div 2, stepN+1, stepMax);krug(x,y+r,r div 2, stepN+1, stepMax);
krug(x-r,y,r div 2, stepN+1, stepMax);krug(x,y-r,r div 2, stepN+1, stepMax);
end;
Arc(x,y,r,0,360); {получаем окружноть}
end;
begin {начало основной программы}
read(stepMax);
x:=700; {начальные координаты}y:=1000;
r:=90; {начальный радиус}SetWindowSize(700,700); {размер графического окна}
krug(WindowWidth div 2, WindowHeight div 2{ Размещаем рисунок в центре графич окна
}, r, 0, stepMax);
end.
```

Изменяя в процессе демонстрации работы программы stepmax можно получать множество привлекательных рисунков.

Фракталы

Пример 1. Кривые Гильберта

Этот узор состоит из суперпозиции пяти кривых. Три наложенные друг на друга кривые имеют форму H_1, H_2 и H_3.H_{i+1} получается соединением четырех экземпляров H_i вдвое меньшего размера, повернутых соответствующим образом и "стянутых" вместе тремя прямыми линиями. H_1 можно считать состоящей из четырех вхождений пустой H_0, соединенных этими же тремя линиями. Кривая H_i называется кривой Гильберта i-го порядка в честь ее первооткрывателя Д.Гильберта.Каждая кривая H_i состоит из четырех вдвое меньших H_{i-1}, то процедура для рисования Hi будет включать четыре обращения для рисования H_{i-1}, соответствующим образом повернутых и уменьшенных вдвое. Обозначим эти четыре части через A, B, C и D.

Это кривые Гильберта 1-го порядка.

Соединительные прямые будем обозначать стрелками, указывающими в соответствующем направлении. Будем предполагать, что направление задается целым параметром i как i·45 градусов.Тогда получаем:

Для построения A, B, C, D получается следующая схема рекурсий:

Кривые Гильберта 2-го порядка.

Для рисования линии будем использовать процедуру:

Line(i, s: integer), где i - направление, s - длинаотрезка.

```
procedure Line( i, s: integer);
beginx:=i*45/180*pi;LineTo(PenX{текущиекоординаты} + Round(s*cos(x)), PenY
+ Round(s*sin(x)));
end;
```

Используя эту процедуру, с помощью рекурсивных обращений напишем процедуру, соответствующуюсхемеA:

```
Procedure A(i, s: integer);
beginif i>0 thenbegin
D(i-1,s); Line(4,s);A(i-1,s); Line(6,s);A(i-1,s); Line(0,s);B(i-1,s);
end;
end;
```

Аналогично составляются процедуры для B, C, D. Процедура A инициируется главной программой по одному разу аждой из кривых

Гильберта. Главная программа определяет начальную точку кривой, т.е. исходные координаты (x0,y0) и начальное значение длины отрезка u (желательно, чтобы u=2n).

Эта программа рисует кривые Гильберта 5-го порядка.

```
Uses Crt, Graphabc;
Var i,x0,y0,u : integer;
x : Real;
procedure Line( i, s: integer);
begin
x:=i*45/180*pi;
LineTo(PenX+ Round(s*cos(x)), PenY + Round(s*sin(x)));
END;
Procedure B(i, s: integer);forward;Procedure C(i, s: integer);forward;
Procedure D(i, s: integer);forward;Procedure A(i, s: integer);
begin
if i>0 then
beginD(i-1,s); Line(4,s);A(i-1,s); Line(6,s) ;A(i-1,s); Line(0,s);B(i-1,s);
end;
end;
Procedure B(i, s: integer);
begin
if i>0 thenbeginC(i-1,s); Line(2,s);B(i-1,s); Line(0,s);B(i-1,s); Line(6,s);A(i-1,s);
end;
end;
Procedure C(i, s: integer);
begin
if i>0 thenbegin
B(i-1,s); Line(0,s);C(i-1,s); Line(2,s);C(i-1,s); Line(4,s);D(i-1,s);
end;
end;
Procedure D(i, s: integer);
begin
if i>0 thenbegin
A(i-1,s); Line(6,s);D(i-1,s); Line(4,s);D(i-1,s); Line(2,s);C(i-1,s);
end;
end;
begin
SetWindowSize(700,540);
write (' ','КРИВЫЕГИЛЬБЕРТА');
```

```
begin
x0:=400; y0:=350;{координаты начала рисования} u:=128{длина отрезка}; i:=1;
while i<=5 {задаемпорядок} do
begin
MoveTo(x0,y0); {задаем координаты пера}
A(i,u); i:=i+1; u:=u div 2;
x0:=x0+(u div 2); y0:=y0+(u div 2);
end;
end;
end.
```

Кривые Серпинского

Аналогично, путем наложения друг на друга нескольких кривых, получается рисунок из кривых Серпинского. Главное отличие кривой Серпинского от кривой Гильберта в том, что первая кривая замкнута. Значит, основная рекурсивная схема должна давать разомкнутую кривую, четыре части которой соединяются линиями, не принадлежащими самому рекурсивному образу. Четыре составляющих образа обозначим через A, B, C, D

Соединительные прямые будем обозначать стрелками, указывающими в соответствующем направлении. Будем предполагать, что направление задается целым параметром i как i·45 градусов.

Основной образ кривых Серпинского задается программой:

```
beginif i>0 thenbegin
A(i-1,s); Line(7,s);B(i-1,s); Line(0,2*s);D(i-1,s); Line(1,s);A(i-1,s);
end;
end;
```

Аналогично получаются процедуры для B, C, D. Главная программа строится по образу S. Ее задача - установить начальные значения для координат рисунка и задать единичную длину линий.

Эта программа рисует кривые Серпинского 4-го порядка.

```
Uses Crt, Graphabc;
Var x0,y0,u,i : integer;
X : Real;
```

```
procedure Line( i, s: integer);
begin
x:=i*45/180*pi;
LineTo(PenX{текущиекоординаты} + Round(s*cos(x)), PenY + Round(s*sin(x)));
end;
Procedure B(i, s: integer);forward;Procedure C(i, s: integer);forward;
Procedure D(i, s: integer);forward;procedure A(i,s: integer);
begin
if i>0 thenbeginA(i-1,s); Line(7,s);B(i-1,s); Line(0,2*s);D(i-1,s); Line(1,s);A(i-
1,s);end;
end;
Procedure B(i, s: integer);
begin
if i>0 thenbeginB(i-1,s); Line(5,s);C(i-1,s); Line(6,2*s);A(i-1,s); Line(7,s);B(i-
1,s);end;
end;
Procedure C(i, s: integer);
begin
if i>0 thenbeginC(i-1,s); Line(3,s);D(i-1,s); Line(4,2*s);B(i-1,s); Line(5,s);C(i-
1,s);end;
end;
Procedure D(i, s: integer);
begin
if i>0 thenbeginD(i-1,s); Line(1,s);A(i-1,s); Line(2,2*s);C(i-1,s); Line(3,s);D(i-
1,s);end;
end;
begin
SetWindowSize(900,1000);
write (' ', 'КРИВЫЕСЕРПИНСКОГОЧЕТВЁРТОГОПОРЯДКА');
x0:=120; y0:=700; {координаты начала рисования} u:=13; i:=1;
for i:=1 to 4 do beginMoveTo(x0,y0);
A(i,u); Line(7,u);B(i,u); Line(5,u);C(i,u); Line(3,u);D(i,u); Line(1,u);
                        end;
end.
```

Приведем пример простой программы square, строящей на экране компьютера изображение вписанных друг в друга квадратов и использующей для этого рекурсивный вызов функции.

Анализ изображения: на рисунке большой квадрат со вписанными в него квадратами меньшего размера. Причём, каждый вписанный квадрат имеет свои координаты, которые вычисляются по координатам описанного квадрата. Всего изображено 6 уровней квадратов. Для того, чтобы составить программу построения этого изображения, можно:

описать функцию изображения одного квадрата со вписанным квадратом меньшего размера.

для изображения каждого квадрата следующего уровня использовать эту же функцию, только с другими значениями параметров которые будут вычисляться автоматически в той же функции рисования квадратов.

Опишем алгоритм рисования:

Сначала вычерчивается первый четырёхугольник (квадрат). После этого вызывается функция Kvad строящая внутри этого четырехугольника (квадрата) малый квадрат так, что вершины его лежат точно на серединах сторон большого квадрата. В обращение включен целочисленный аргумент n, определяющий глубину рекурсии.

Для рисования квадратов будем использовать линии определенной длины. При этом надо задавать координаты каждой линии (x1,y1), (x2,y2). Для этого будем использовать параметры графического окна:

```
xc:=WindowWidth div 2; {возвращаем ширину графического окна}
yc:=WindowHeight div 2; {возвращаем высоту - // - // - }
ax:=xc-a; ay:=yc-a; {находим координаты для первой линии}
bx:=xc+a; by:=yc-a; {находим координаты для второй линии}
cx:=xc+a; cy:=yc+a; {находим координаты для третьей линии}
dx:=xc-a; dy:=yc+a; {находим координаты для четвертой линии}
```

где a - длина линии.

```
function Kvad(xc,yc,a:integer; ax,ay,bx,by,cx,cy,dx,dy,m:real; n:integer) :integer;
begin
if n=0 then Kvad:=0
else
begin {координаты линий округляем}
line(round(ax),round(ay),round(bx),round(by));
line(round(bx),round(by),round(cx),round(cy));
```

```
line(round(cx),round(cy),round(dx),round(dy));
line(round(dx),round(dy),round(ax),round(ay));
dec(n); {уменьшаем n на 1}
```

Рекурсивный вызов функции и вычисление новых значений производится так:

```
Kvad:=Kvad(xc, yc, a, ax+(bx-ax)*m, ay+(by-ay)*m, bx+(cx-bx)*m, by+(cy-by)*m,cx+(dx-cx)*m, cy+(dy-cy)*m, dx+(ax-dx)*m, dy+(ay-dy)*m, m, n);
end;
end;
```

m - вспомогательная переменная, в данном случае используется для нахождения середины стороны.

Таким образом, в описанной функции мы осуществляем вызов ее же самой в качестве вспомогательной.

Если осуществить вызов этой же функции из основной программы с начальными параметрами, то рекурсивные вызовы никогда не закончатся и программа будет работать бесконечно. Для того, чтобы этого не случилось, можно ввести в качестве аргумента функции некоторую величину n, которая при каждом новом вызове процедуры будет уменьшаться на 1 (применяем dec(n)), а в тело функции что его операторы должны выполняться только при n<>0, то есть это условие должно играть роль своеобразной «заглушки», ограничивающей число вызовов.

Рекурсия может быть использована для получения линейного рисунка, например «дерева», например:

```
Uses Crt, Graphabc;
Var x, fi : Real;
i, x0, y0, u, divisor : integer;
```

Для рисования линии будем использовать процедуру: Line(i, s: integer), где i - направление, s - длина отрезка.

```
procedure Line( i, s: integer);
begin
x:=(i*fi-90)*pi/180;
LineTo(PenX{текущиекоординаты}+ Round(s*cos(x)), PenY + Round(s*sin(x)));
{рисуемлиниюподопределеннымуглом}
end;
procedure A(i,s,k, kmax: integer); //k - шагрекурсии, kmax - порядокрекурсии
var cX, cY: integer; //координаты разветвления
```

```
begin
Line(i,s); //рисуемножкуветви
if k<kmax thenbegin
cX:= PenX; //запоминаемкоординатыветвления
cY:= PenY;A(i,s div divisor,k+1,kmax); //рисуемсреднююветвь
MoveTo(cX,cY);A(i-1,s div divisor,k+1,kmax); //рисуемлевуюветвь
MoveTo(cX,cY);A(i+1,s div divisor,k+1,kmax); //рисуемправуюветвь
end
elsebegin
Line(i,s); //если рекурсия окончена, дорисовываем ветку
end;
end;
begin
read(i); //порядокрекурсии
SetWindowSize(700,640);
x0:=330; //начальные координаты
y0:=500;
u:=243; //длина самой крупной ножки
divisor:=3; //во сколько раз длина ветки на следующем шаге меньше, чем
на предыдущем.
fi:=70; //угол между ветвями
MoveTo(x0,y0); //переходим к начальным координатам
A(0,u,0,i); //рисуемветку
end.
```

Разработаем программу, строящую на экране компьютера изображение описанных кругов.

```
uses crt,graphABC,events;
var
stepMax,x,y,r:integer;
```

напишем процедуру krug которая рисует окружность и описывает её шестью окружностями меньщего размера:

```
procedure krug(x,y,r,stepN,stepMax:integer);
var nr:integer;
begin
if stepMax>stepN then
begincircle(x,y,r); {рисуем основной круг}
```

Вычисляем центр каждого круга:

```
krug(x+round(2*r),y, r div 3, stepN+1, stepMax); {круг с права}
```

```
krug(x-round(2*r),y, r div 3, stepN+1, stepMax); {кругслева}
krug(x+round(2*r*cos(PI/3)),y-round(2*r*sin(PI/3)),r div 3, stepN+1, stepMax);
{верхнийправыйкруг}
krug(x-round(2*r*cos(PI/3)),y-round(2*r*sin(PI/3)),r div 3, stepN+1, stepMax);
{левыйверхнийкруг}
krug(x+round(2*r*cos(PI/3)),y+round(2*r*sin(PI/3)),r div 3, stepN+1, stepMax);
{нижнийправыйкруг}
krug(x-round(2*r*cos(PI/3)),y+round(2*r*sin(PI/3)),r div 3, stepN+1, stepMax);
{нижнийлевыйкруг}
end;
end;
begin
write('[фигуру какого порядка следует нарисовать]','_');read(stepMax);
clearwindow;
write('[введите желаемый радиус]','_'); read(r);
clearwindow;
writeln (' ','на экране окружности ' ,stepMax, ' го',' порядка');
x:=350; {начальные координаты}
y:=400;SetWindowSize(700,700);
krug(x,y,r,0,stepMax); {вызываем процедуру krug }
end.
```

Дерево рекурсивных вызовов

Проанализируем пример 3:

Для получения окружност со вписанными окружностями (второго порядка) требуется сделать следующие действия:

Шаг 1.

```
procedure krug(x,y,r,stepN,stepMax:integer);
var nr:integer;
begin
if stepMax>stepN thencircle(x,y,r);
end;
```

Шаг 2.

Вычислим радиус для следующей окружности: nr:=r div 3;

Вызываем в процедуре krug ещё одну процедуру krug и передаём ей новые параметры:

krug(x,y,nr, stepN+1, stepMax); получаем:

окружность в центре которого ещё одна окружность.

Шаг 3.

Добавляем в основную процедуру krug вторую процедуру krug со следующими параметрами:
krug(x,y+2*nr,nr, stepN+1, stepMax); передвигаемся по игрику
Шаг 4.
Добавляем третью процедуру krug : krug(x,y-2*nr,nr, stepN+1, stepMax);
Шаг 5.
Добавляем четвертую процедуру krug:
Для нахождения центра круга будем использовать тригонометрические функции.
krug(x+2*round(nr*sin(2*PI/3)),y+nr,nr, stepN+1, stepMax);
Шаг 6.
Аналогично добавляем пятую процедуру krug:
krug(x-2*round(nr*sin(2*PI/3)),y+nr,nr, stepN+1, stepMax);
Шаг 7.
Добавляем шестую процедуру krug:
krug(x+2*round(nr*sin(2*PI/3)),y-nr,nr, stepN+1, stepMax);
Шаг 8.
Добавляем седьмую процедуру krug:
krug(x-2*round(nr*sin(2*PI/3)),y-nr,nr, stepN+1, stepMax);
получаем конечный результат:
«Заглушка» срабатывает когда stepN становится больше stepMax
При каждом повторе процедуры krug, stepN увеличивается на единицу.

Решение нелинейного уравнения методом итерации в Excel

Задание: на отрезке [a,b] требуется найти **корень нелинейного уравнения методом итерации** в табличном процессоре Excel с использованием **циклических ссылок.**

$x-x^3+1=0 \quad a=1 \quad b=2$

Решение:

Найдем корень нелинейного уравнения **в табличном процессоре Excel методом итерации** с использованием циклических ссылок. Для включения **режима циклических вычислений в Excel**2003 в меню Сервис/Параметры/вкладка Вычисления следует поставить флажок Итерации и флажок выбора вида ведения

вычислений: автоматически. В MS Excel 2010 следует зайти в меню Файл/Параметры/Формулы и поставить флажок в поле "Включить итеративные вычисления".

Используем формулу:

$$X_{n+1} = X_n - \frac{f(X_n)}{M}, \quad n = 0, 1, 2, \ldots,$$

M – максимальное значение производной на промежутке (по модулю). Найдем M, для этого вычислим

f’(1)=-2 f’(2)=-11

Т. к. значение производных <0, возьмем функцию с противоположным знаком: f(x)=-x+x3-1.

M=11

В ячейку A7 введем значение a =1, в ячейку B7 введем формулу расчета текущего значения x: =ЕСЛИ(B7=0;A7;B7-(-B7+СТЕПЕНЬ(B7;3)-1)/11)

В ячейку C7 введем формулу для контроля значения f(x): =B7-СТЕПЕНЬ(B7;3)+1

Получим корень уравнения x=1,375

4			
5	Метод итерации		
6	***Хнач***	***Хтек***	***F(Хтек)***
7	1,000	1,325	0,000
8			

В A7 введем начальное приближение = 2, получим корень x=1,375

Получен тот же результат, значит корень на данном промежутке один.

4			
5	Метод итерации		
6	***Хнач***	***Хтек***	***F(Хтек)***
7	2,000	1,325	0,000
8			

Порядок выполнения работы.

1. Изучить теоретический материал.
2. Методом простых итераций найти корень уравнения (задается преподавателем). Решение реализовать на языке высокого уровня и в электронной таблице.

3. Поснить работу алгоритма вычисления факториала с помощью рекурсии.
4. Построить кривые Серпинского и предложить их применение в математической биологии и медицине.
5. Преложить метод реализции п.4 в MatCad и Excel.
6. Оформить отчет, включающий: результаты выполнения работы (алгоритмы, программные коды, скри-шоты) и ответы на контрольные вопросы (не менее трех).

Контрольные вопросы:
1. Что называется итерацией?
2. Что называется рекурсией?
3. Для чего применяются кривые Серпинского?
4. Какова роль фракталов в природе?
5. Какие физиологические процессы можно отнести к рекурсивным?
6. Что такое метод дихатомии?
7. Каким образом можно применять рекурсию при вычисления ряда (например, Тейлора для гармонической функции)?
8. Какой итерационный процесс будет бесконечным: расходящийся, сходящийся?
9. Как представить процесс дифференциальной диагностики заболевания в виде итарации (рекурсии)?
10. Как реализуется итерация в логическом базисе с помощью логических функций и переменных?
11. Приведите примеры семантических рекурсий (например: «у попа была собака, он ее любил…»)
12. Как осуществляется рекурсивный диагностический процесс?

Литература
1. Булавин, Леонид Анатольевич. Компьютерное моделирование физических систем [Текст] : учебное пособие / Л. А. Булавин, Н. В. Выгорницкий, Н. И. Лебовка. - Долгопрудный : Интеллект, 2011. - 349 с

2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники [Текст] : учебное пособие / Л. Г. Гагарина, А. А. Петров. - М. : Форум, 2011. - 368 с.
3. Рекурсивное построение детерминистических фракталов [Электронный ресурс] : методические указания к выполнению лабораторной работы по дисциплине «Компьютерные технологии в научных исследованиях» для студентов направления подготовки 222900.68 / Юго-Зап. гос. ун-т ; сост.: А. М. Стороженко, И. А. Шабанова. - Электрон. текстовые дан. (297 КБ). - Курск : ЮЗГУ, 2015. - 10 с.
4. Златопольский Д.М. Рекурсия. Информатика, 1996
5. Головешкин В.А., Ульянов М.В. Теория рекурсии для программистов М.,Физматлит : 2006 296 с
6. Федер Е. Фракталы Издательства Едиториал УРСС, Ленанд Серия Синергетика: от прошлого к будущему, 2014, 264 с.
7. Построение изображений с помощью рекурсии http://festival.1september.ru/articles/646874/
8. Информационные сайты: http://web-pascal.narod.ru/stat/recurs.htm,http://it.kgsu.ru, http://ru.wikipedia.org, http://tvd-home.ru/recursion, http://pascal.helpov.net

ЛАБОРАТОРНАЯ РАБОТА №6. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Цель работы: овладение навыками имитационного моделирования, включая анимацию, для решения медико-биологических задач

Краткие теоретические сведения.

Модель — это создаваемое человеком подобие изучаемого объекта (макет, изображение, схема, карта, словесное описание, математическое представление и т.п.). Метод моделирования состоит в исследовании объекта, явления или процесса путем построения моделей и их изучения. Модель всегда проще реального объекта, но она позволяет выделить главное, не отвлекаясь на детали. Необходимость моделирования объясняется принципиальной невозможностью исследования многих объектов или большой ресурсоемкостью их изучения.

Различают биофизические, физические, электрические, ситуационные, информационные, математические и другие модели.

Информационная модель — модель объекта, процесса или явления, в которой представлены информационные аспекты моделируемого объекта, процесса или явления. Среди информационных моделей особое место занимают модели представления знаний. Математическая модель — приближенное описание объекта, явления или процесса с помощью математической символики. Эта модель представляет собой систему математических соотношений: формул, функций, уравнений, систем уравнений, описывающих те или иные стороны изучаемого объекта, явления или процесса. Математическое моделирование — мощное средство познания, прогнозирования и управления. Анализ математической модели помогает проникнуть в суть изучаемого объекта или явления.

Математические модели строятся на основе данных эксперимента или умозрительно, описывают гипотезу, теорию или закономерность того или иного феномена и требуют дальнейшей проверки на практике. Различные варианты проводимых экспериментов выявляют границы применения математической модели и создают

условия для ее дальнейшей коррекции. Математическое моделирование часто позволяет предвидеть характер изменения исследуемого процесса в условиях, трудно воспроизводимых в эксперименте, а в отдельных случаях позволяет предсказать ранее неизвестные явления и процессы.

Процесс математического моделирования принято делить на несколько этапов.

1. *Постановка задачи.* Необходимо отметить, что построение модели подразумевает наличие у специалиста хорошего уровня знаний предметной области, в рамках которой осуществляется моделирование. В постановку задачи входят определение цели исследования, выделение объекта исследования, определение параметров исследуемого объекта, выявление взаимосвязей между параметрами. Этап завершается записью модели в математическом виде.

2. Проведение модельных экспериментов. На этом этапе осуществляется решение прямой задачи, для которой предназначена математическая модель, т. е. получение выходных данных для дальнейшего сопоставления с результатами наблюдений изучаемых явлений. Исследователь сознательно изменяет условия функционирования модели, регистрирует ее «поведение» в разных условиях. Важная роль при проведении модельных экспериментов принадлежит вычислительной технике. Именно она обеспечивает возможность обсчета многочисленных модельных экспериментов. Итогом второго этапа моделирования является множество результатов модельных экспериментов.При математическом моделировании разных процессов и явлений может использоваться один и тот же математический аппарат. Это упрощает задачу моделирования, дает возможность выбора из полученных вариантов.

3. *Оценка реализованной модели.* Выясняют, удовлетворяет ли созданная математическая модель критерию практики, т.е. согласуются ли результаты наблюдений с теоретическими (гипотетическими, модельными) данными в пределах заданной точности. Достижение такого результата означает, что положения, лежащие в

основе модели, правильны и модель пригодна для исследования выбранного объекта или явления.

4. *Анализ модели на основе накопленных данных об изучаемом объекте, модернизация первоначально построенной модели.* С получением новых научных данных знания об исследуемом объекте уточняются, и наступает момент, когда результаты, получаемые на основании существующей модели, перестают им соответствовать. Возникает необходимость уточнения данной модели или построения новой. Между моментами построения исходной и последующей моделей проходят разные промежутки времени в зависимости от сути изучаемого явления, уровня и скорости исследования данной предметной области, характера полученных новых знаний и данных.

В медицине модели применяются для исследования структур, функций и процессов на разных уровнях организации живого организма: атомарно-молекулярном, субклеточном, клеточно-тканевом, органно-системном, организменном, биоценотическом.

В медицине, как и в биологи, используются в большинстве случаев биологические, физико-химические, математические модели. Исторически сложилось, что в медицине до сих пор широко распространены словесные описания объектов и процессов (например, заболеваний), а в последние десятилетия все чаще применяются информационные модели.

Биологические модели в медицине применяются для воспроизводства на лабораторных животных заболеваний или состояний, встречающихся у человека. Таким образом, в эксперименте исследуются механизмы возникновения заболевания, его этиология, патогенез, течение, изучаются варианты воздействия на протекание болезни, сравнивается эффективность применения различных лечебных пособий. В эксперименте, например, моделируются ишемические нарушения и гипертоническая болезнь, злокачественные новообразования и генетические заболевания, инфекционные процессы и др.

Для реализации биологических моделей экспериментальным животным вводят токсины, заражают их микробами, перевязывают

сосуды, исключают из пищи определенные вещества, помещают в искусственно создаваемую среду обитания и др. Подобные экспериментальные модели применяются в нормальной и патологической физиологии, генетике, фармакологии, хирургии, реаниматологии. Физико-химические модели имитируют сложные акты поведения, например формирование условного рефлекса.

Удачным следует признать опыт построения электронных схем, моделирующих биоэлектрические потенциалы в нервной клетке и синапсе на основе данных электрофизиологических исследований.

В настоящее время в медицине самое широкое распространение получили математические модели. Они используются практически во всех ее областях. Математические модели применяются для изучения сложных физиологических процессов, диагностики патологических состояний, исследования взаимодействия систем организма в норме и патологии, при изучении эпидемических процессов, в клинической иммунологии, фармакокинетике.

Из математических моделей, известных в физиологии, следует упомянуть модель возбуждения нервного волокна, предложенную А.Ходжкином и А.Хаксли.

Модель сердечной деятельности Ван дер Пола и Ван дер Марка, основанная на теории релаксационных колебаний, позволила предсказать возможность особого нарушения сердечного ритма, впоследствии обнаруженного у человека.

Ярким примером использования математической модели для обобщения накопленных экспериментальных знаний является модель кровообращения Ф. Гродинза. Построением и исследованием моделей кровообращения, применяющихся в практике российской сердечно-сосудистой хирургии, занимается В.А.Лищук.

В медицинской информатике широко используется моделирование, особенно часто математическое и информационное. Математические модели используются для расчета клинически значимых показателей при обработке сигналов и изображений, для описания заболеваний и состояний при вычислительной диагностике

и прогнозировании. Информационное моделирование все чаще применяется при описании деятельности ЛПУ и их подразделений.

Когда осуществляют процессный анализм в ЛПУ с последующим выходом на планирование эксперимента и принятие управленческих решений, то всегда наталкиваемся на одну и ту же проблему разработки и апробации модели управления процессов обеспечения КМП в условиях реального функционирования ЛПУ. Одна из важных особенностей управления качеством производства медицинских услуг — принципиальная невозможность проведения реальных работ по управлению КМП до завершения эксперимента. Возможным выходом является использование имитационных моделей. Сущность метода имитационного моделирования состоит в построении так называемой имитационной модели исследуемого ЛПУ или его подразделения и целенаправленном экспериментировании с разрабатываемой моделью управления для получения ответов на те или иные вопросы. Говоря о методе имитационного моделирования, как правило, имеют в виду метод, ориентированный на применение вычислительной техники, хотя в принципе могут использоваться любые средства, включая лист бумаги и карандаш.

Другой важный аспект - использование имитационных моделей в процессе эксплуатации информационных технологий управления для принятия управленческих решений по качеству. Такие модели создаются в процессе проектирования, чтобы их можно было непрерывно модернизировать и корректировать в соответствии с изменяющимися условиями работы пользователей. Эти же модели могут быть использованы для обучения персонала ЛПУ перед вводом в действие разработанных технологий в эксплуатацию или для проведения деловых игр.

Принципиальные возможности метода имитационного моделирования весьма велики, он позволяет при необходимости исследовать системы любой сложности и назначения с любой степенью детализации. Ограничениями являются лишь мощность используемой ПЭВМ и трудоемкость подготовки сложного

комплекса программ. Методы имитационного моделирования развиваются в основном в направлении исследования степени подобия имитационных моделей реальным системам и разработки типовых методов и приемов создания имитационных моделей.

Имитационное моделирование в медицине используется в основном по следующим направлениям:

1. при исследовании сложных внутренних и внешних взаимодействий ЛПУ с целью оптимизации их функционирования. Для этого на модели ЛПУ изучают закономерности взаимосвязи переменных, вносят в модель ЛПУ изменения и наблюдают их влияние на поведение системы производства медицинских услуг;
2. для прогнозирования поведения ЛПУ в будущем на основе моделирования развития самого ЛПУ и его внешней среды;
3. в целях обучения персонала ЛПУ, которое может быть двух типов:

- первый тип - индивидуальное обучение оператора, управляющего неким технологическим процессом или медицинской аппаратурой;
- второй тип - обучение группы персонала, осуществляющей коллективное управление сложным объектом по производству медицинских услуг.

Рассмотрим простейшие примеры математического имитационного моделирования.

1. Вычисление площади неизвестной фигуры на примере определения числа Пи.

Исходя из предположенийполагаем, что площадь круга известного радиуса равна квадрату радиуса умноженного на некоторую величину. Эта величина экспериментально определяется следующим образом. На листе бумаги чертится круг радиусом R (чем больше, тем лучше), а около него описывается квадрат со стороной 2R . Далее на рисунок нангосится случайным образом несколько сот точек в случайных местах. На следующем этапе подсчитывается количество течек: общее – N и попавших в круг – N0. Тогда отношение N0/N – позволяет определить площаджь круга в зависимости от площади квадрата, которая известна.

Заметим, что таким образом можно определить площадь любой фигуры, находящейся в квадрате с известными сторонами –например, площадь клетки, листа, капли крови, отпечатка пальцев, пятна лакмусовой бумажки на определенную концентрации химического вещества. Такой способ называется методом Монте-Карло.

Имитационная модель описанного процесса легко формализуется и следовательно может быть реализована с помощью ПЭВМ, имеющей фон Нейманскую архитектуру и последовательную организации выполнения программного кода. Алгоритм в этом случае выглядит следующим образом:

1. Задаемся определенным радиусом R.
2. Задаемся количеством «точек» N. Количество точек, попавших в круг считаем равным 0.
3. С помощью датчика случайных чисел генерируем координаты точки x и y [0,2R].
4. Проверяем: если координаты точки попали в круг с координатами центра (R,R) и радиусом R, то увеличиваем количество точек, попавших в круг на 1.
5. Процедуры 3,4 повторяем, пока общее количество случайным образом сгерерированных точек не превысит N.
6. Находим число Пи.

2. Игра «Жизнь».

Многие процессы, интересующих медиков (например, динамика возрастного состава в регионе) может быть с определенной точностью описано математической моделью. Самая простая модель известна под назнванием «Игра Жизнь».

Предполагается наличие прямоугольного клетчатого поля, в каждой клетке которого может «жить» существо. Если клетка пустая – то в ней никто не живет. Модель задается двумя параметрами: начальной конфигурацией (размером поля и расположением живых существ) и определенными «биологическими законами», регулирующими жизнь популяции существ. На каждой итерации осуществляется последовательный просмотр всех клеток с некоторой

начальной (координаты клетки выбираются исследователем или случайным обрзом) с применением к ним биологических законов.

В качестве типовых законов, предлагаются, например, следующие:

1. Если выбранная клетка пуста, а в соседней с ней клетках находится более двух существ, то внутри клетки появляется существо («размножение»).
2. Если выбранная клетка непуста, а в соседних с ней клетках живет меньше трех или больше четырех существ, то клетка очищается (существо в ней погибает от одиночества или перенаселения).
3. Если правила 1 и 2 не выполняются, то ничего с клеткой не происходит.

(Под соседними полразумеваются восемь окружающих клеток, за грницей ореала – прямоугольного поля – существ нет).

3. Модель «хищник» - «жертва».

В системе хищник-жертва ситуация моделируется следующим образом. В случае конкурирующих популяций исчезновение одной означает выигрыш для другой в борьбе за дополнительные ресурсы. Обозначим через С численность популяции хищника, N– популяцию жертвы. Наиболее популярная модель, отражающая колебания численности имеет вид:

$$N_{i+1} = N_i + (r \cdot N_i - a \cdot N_i \cdot C_i) \cdot \Delta t,$$
$$C_{i+1} = C_i + (-q \cdot C_i - a \cdot f \cdot N_i \cdot C_i) \cdot \Delta t.$$

Согласно первому уравнению при С=0 численность жертв быстро растет со скоростью r , поскольку модель не учитывает внутривидовой конкуренции. Скорость роста числа жертв $(\frac{\Delta N}{\Delta t})$ уменьшается тем больше, чем чаще происходят встречи особей видов (тогда а - коэффициент эффективности поиска).

Второе уравнение показывает, что в отсутствии жертв численность хищников быстро убывает со скоростью q: положительное слагаемое в правой части уравнения компенсирует

эту убыль, f – коэффициент эффективности перехода пищи к потомству хищников.

4. Внутривидовая конкуренция в популяции с дискретным размножением.

Для популяций с дискретным размножением (некоторые виды растений, насекомые) поколения дифференцирпованно разнесены во времени и особи разныз покалений вместе не сосуществуют. Численность подобной популяции характеризуется числом N_t , а время t- дискретная величина – можно, в первом приближении, считать номером популяции. Тогда одна из моделей межвидовой конкуренции может быть описана уравнением:

$$N_{t+1} = \frac{N_t \cdot R}{1 + (a * N_t)^b}$$

где R – скорость воспроизводства попляции в отсутствии внутривидовой конкуренции (математически это соответствует a=0);a – параметр, характеризующий интенсивностьвнутривидовой конкуренции, приb=1 осуществляется выход численности популяции на стационарное значение при любых значениях других параметров модели.

Знаменатель в уравнении отражает наличие конкуренции, делающей скорость роста тем меньше, чем больше численность популяции. Ланная модель описывает четыре вида эволюции:

1. мнотонное установление стационарной численности популяции;
2. колебательное установление стационарной численности популяции;
3. устойчивые предельные циклы изменения численности популяций;
4. случайные изменения численности популяции без наличия явных закономерностей.

5. Внутривидовая конкуренция в популяции с непрерывным размножением.

В данном случае численность популяции N(t) является непрерывной функцией во времени. В начале эволюционного процесса численность популяции невилика, а ее удельная скорость не зависит от численности: $\frac{1}{N} \cdot \frac{\Delta N}{\Delta t} = r$ - скорость роста численности популяции в отсутствии конкуренции. Далее, по мере роста численности, скорость роста начинает уменьшаться и при достижении определенного критического значения К обращаетя в ноль. Таким образом, в первом приближении, математическая модель имеет вид:

$N_{i+1} = N_i + r \cdot N_i \cdot (\frac{K-N_i}{K}) \cdot \Delta t$.

Порядок выполнения работы.

1. Изучите теоретические сведения.
2. Составьте блок-схему алгоритма и отладьте программу, определяющую число Пи. Постройте график зависимости числа Пи от количества точек N в двух вариантах: учитывающеьм и неучитывающем попадание координат случайной точки на окружность. Сделайте выводы.
3. Разработайте последовательность действий (и апробируйте ее) для экспериментального определения числа Пи в электронной таблице Excel.
4. Составьте и отладьте программу «игра Жизнь» на языке высокого уровня и предложите ее реализацию в Excel. Сравните полученные Вами результаты с программным продуктом, предложенном на сайте http://www.nature.air.ru/models/models.htm.

5а. Проведите с помощью MatCad и Exel моделирование динамики в системе хищник-жертва при значениях параметров модели: r=8, a=0.15, q=2, f=0.5, N0=175, C0=45. Сделайте выводы.

6а. Изучите функционирование аналогичной программы, представленной на сайте http://www.nature.air.ru/models/models.htm.

7а. Исследуйте зависимость амплитуд колебаний численности хищников от амплитуд колебаний численности жертв в зависимости

от параметров a и f. Значения остальных параметров фиксируйте по своему усмотрению.

5б. Составьте и отладьте программу моделирующую внутривидовую конкуренция в популяции с дискретным размножением на языке высокого уровня и предложите ее реализацию в Excel.

6б. Проведите с помощью MatCad и Exel моделирование в четырех видах эволюции с R=2 (1 и 2 режимы) и R=2 (3 и 4 режимы). На фазовой плоскости (b, R) найдите границы зон, разделяющих разные режимы эволюции изучаемой системы. Сделайте выводы.

7в. Составьте и отладьте программу моделирующую внутривидовую конкуренцию в популяции с непрерывным размножением при нескольких значениях входящих в модель параметров на языке высокого уровня и предложите ее реализацию в Excel. Сделайте выводы.

7в. Постройте (или найдите в информационных источниках) альтернативную модель внутривидовой конкуренции и исследуйте предсказываемые режимы.

Примечание: обучающийся самостоятельно выбирает подпункты «а», «б» и «в», исходя из номеразачетной книжки (определяется отстаток от деления последние цифры на 3 и прибавляется 1 – полученное число пределяет соответствующую букву).

8. Изучите интерфейс и функциональные возможности программ, представленных на сайте http://www.nature.air.ru/models/models.htm. Результаты изучения представьте в табличном виде.

9. Оформите отчет, включающий результаты выполненных действий и презентации к минидокладу на тему «Роль имитационного моделирования в медико-биологических исследованиях».

10. Придумайте контрольные вопросы (не менее 5) в виде мини тестов по указанной в п.9 тематике. (Конкретное гносеологическое насыщение вопросов – на Ваше усмотрение).

Литература

1. Имитационное моедлирование в биологии. Презентация. http://900igr.net/prezentatsii/biologija/Modelirovanie-v-biologii/Modelirovanie-v-biologii.html