12. Массивы (Arrays).

В программе переменными назначаются определенные значения. В процессе выполнения программы эти значения доступны в выражениях через идентификаторы переменных. В программах, рассмотренных до сих пор, не было слишком много переменных, и хотя мы имели возможность обрабатывать большие списки чисел, для этой цели мы не использовали различные переменные, а все числа представляли одной переменной, так как в программе не требовалось хранить значение каждой величины для последующей обработки. Например, в задаче суммирования чисел использовалась только одна переменная для представления вводимого числа, которая обновлялась при вводе каждого нового числа после того, как ее значение добавлялось к аккумулируемой сумме. Если бы все эти текущие значения потребовались для последующей обработки, мы бы не смогли их использовать.

Один из способов хранить списки чисел — это ввести отдельную переменную для каждого числа, но тогда мы не смогли бы использовать оператор цикла для обеспечения повторного ввода. Пусть мы хотим посчитать сумму шести чисел, и пусть их значения дальше по программе не понадобятся. Тогда код, который вводит эти числа и выводит результат суммирования, может иметь вид:

```
sum = 0;
For i := 1 to 6 Do
Begin Read(x);
    sum := sum+x;
End;
Writeln(' Сумма чисел = ', sum,'.');
```

Фрагмент легко расширить для случая *n* чисел, однако мы все-таки не имели бы возможность впоследствии восстановить их отдельные значения. Теперь решим тот же фрагмент, использовав отдельные переменные для представления каждого числа:

```
Var a,b,c,d,e,f:Real,
...
sum=0;
Read(a); sum = sum+a;
Read(b); sum = sum+b;
```

```
Read(c); sum = sum+c;
Read(d); sum = sum+d;
Read(e); sum = sum+e;
Read(f); sum = sum+f;
Writeln(' Сумма чисел = ', sum,'.');
```

Ясно, что такая запись кроме того, что слишком неудобна, она еще не дает возможности обобщить фрагмент для произвольного количества чисел. Вместо этого нам придется каждый раз добавлять новую переменную, когда нужно ввести побольше чисел. Легко представить, какие сложности могут возникнуть, если нужно будет подсчитать сумму 1000 чисел (1000 имен для переменных, и нужно помнить, какое имя соответствует какой именно переменной!)

Эти и подобные проблемы в Турбо-Паскале решаются с помощью структурированного типа данных, называемого *массивом (Array).*

12.1. Объявление и использование массивов.

Массив – это структура данных, в которой группе однотипных элементов-переменных назначается одно общее имя. Каждый элемент массива обладает своим собственным уникальным *индексом,* с помощью которого можно к нему обращаться. Индекс должен быть только целочисленного или другого порядкового типа, и он указывает на позицию элемента в массиве. Таким образом, массив является упорядоченным по индексу набором элементов.

Исходя из сложности структуры массива, он может быть одномерным (для представления, например, векторов), двумерным (для представления матриц), трех- и т.д. многомерным.

Массив как структурированный тип, определенный пользователем, может быть объявлен предварительно в разделе Type, где новому типу назначается имя-идентификатор, например,

```
Const n=5; m=6;

Type
    vector=Array [1..8] of Integer;
```

```
matrix=Array [1..n, 1..m] of Real;
lists=Array [1..100] of String[40];

Var
    v1,v2: vector;
    ml, m2:matrix;
    11,12:lists;
    c10:Array[1..10]of Char;
```

Здесь переменные *v1* и *v2* представляют группы из восьми целых чисел, переменные *m1* и *m2* – это действительные матрицы размерности *n—m,* переменные *l1* и *l2* – два списка из ста названий строкового типа, а *c10* – символьный массив из десяти элементов.

Количество элементов массива должно быть строго предопределено, т.е. индексный диапазон может быть либо конкретным числом, либо предварительно объявлен постоянной величиной, как в случае типа *matrix.* Использование констант для определения ранга массива дает уже известные преимущества при программировании, а именно, если вдруг понадобится изменить количество элементов, надо будет исправить только значение константы, иначе пришлось бы исправлять все вхождения фиксированного индекса-ранга. Заметим, что нельзя использовать в качестве ранга массива обыкновенную переменную, так как в момент объявления массива ее значение не будет известно компилятору.

Отметим два способа объявления массивов (это касается любого структурированного типа):

а) Сначала объявляется новый тип, а затем переменные данного типа *(vector, matrix, lists);*

б) Тип определяется при объявлении переменной (*c10*).

Оба способа допустимы, но первый способ считается хорошим стилем и дает преимущества в некоторых случаях, а иногда без предварительного объявления нельзя реализовать решение проблемы, например, при передаче параметров процедурам и функциям (см. ниже).

После объявления переменной типа массива, например *v*1, компилятор в оперативной памяти выделяет место для хранения соответствующих значений элементов, в данном случае 8x2=16 байтов, так как для представления одной целочисленной величины типа Integer требуется 2 байта памяти. Аналогично вычисляется количество зарезервированной памяти для других переменных.

К элементам массива можно обращаться с помощью идентификатора имени и индекса, заключенного в квадратные скобки. Например, если мы хотим присвоить пятому элементу массива *v*1 значение 10, мы должны написать:

```
vl[5]:=10;
```

Элемент массива в программе может использоваться в любом месте, где может стоять идентификатор переменной. Например,

° Считывание значения элементу с клавиатуры:

```
Read(vl[1]);
```

° Увеличение значения элемента:

```
vl[1]:=vl[1]+20;
```

° Элементы массива могут участвовать в формировании логических условий (программный фрагмент меняет местами первый и второй элементы массива с помощью дополнительной переменной r):

```
If(vl[1]>vl[2])Then
Begin
    r:=vl[1];
    vl[1]:=vl[2];
    vl[2]:=r
End;
```

° Следующий фрагмент находит среднее значение элементов массива *v*l:

```
average:=0;
sum:=0;
For i:=1 to 8 Do sum:=sum+vl[i];
average :=sum/8;
```

Сейчас, если количество элементов массива изменится, нам придется исправить все вхождения числа 8, этого можно было бы избежать, если бы ранг был определен с помощью константы.

° Следующий фрагмент обнуляет значения элементов массивов *m*1 и *m2:*

```
For i:=1 to n Do
For j:=1 to m Do Begin
   ml[i,j]:=0;
   m2[i,j]:=0;
End;
```

Программа-пример: «Больше среднего».

Даны положительные числа (скажем 200). Надо найти их среднее значение и посчитать количество тех элементов, которые больше среднего значения.

Так как все числа положительны, отрицательное число можно принять за признак конца ввода данных. Это именно тот случай, когда надо использовать массив, так как сначала должны быть введены данные и вычислено среднее значение, а затем каждое число надо сравнить со средним значением. Следовательно, вводимые числа нужно запомнить для их повторного использования,

Алгоритм будет иметь вид:

Инициализация:

Сумма *sum=0;*

Количество вводимых чисел *count=0;*

Количество *greater=0.*

Ввести значения элементов массива и просуммировать:

Ввести первое число *(number);*

Пока number > 0
{
Увеличить count:
Записать *number* в элемент массива → *arr[count]*
Добавить *number* к *сумме*,
Считать *number*;
}
Вычислить среднее значение:
average = sum/count;
Подсчитать элементы, которые больше среднего значения:
Для *index* от 1 до *count*
Если arr [index] >= average *Тогда Увеличить greater;*
Вывести результаты: *average, count* и *greater.*

Переменная *greater* представляет количество чисел, которые больше или равны среднему значению введенных чисел. Значение переменной *count* в начале устанавливается на ноль, хотя до цикла также вводится число, но этот недобор восстанавливается в конце цикла, при вводе отрицательного числа, которое в расчетах не будет участвовать, но значение счетчика увеличится.

Программный код:

```
Program greaters;
Const maxRange=200; { maximum no of elements in array }
Var number, sum, average: Real;
    index, greater, count: Integer;
    arr:Array[1.. maxRange] of Real;
BEGIN
    Sum:=0;
    greater; =0;
    count:=0;
    WriteLn('Enter number: (negative - to terminate!)');
    Read(number);
    While number>0 Do
    Begin
        count:= count+1
```

```
        sum:= sum + number;
        arr[count]:= number;
        Read(number)
    End;
    average:=sum/count;

    For index:=1 to count Do
    If arr[index]> average Then greater= greater+1;

    Writeln(' Number of values input is ', count);
    Writeln(' Number of values greater than average is', greater);
END.
```

Максимальное количество чисел, которые мы можем обработать — 200. Если мы введем меньшее количество чисел *n=50,* то они заполнят первые 50 позиций массива. В общем случае, принято устанавливать максимальное количество элементов для массива, которое может быть реализовано на практике, хотя фактически не все пространство будет использовано.

Программа-пример: Тестировка случайных чисел — подбрасывание кости.

Программа имитирует подбрасывание костей с помощью генерации случайных чисел от единицы до шести. Программа запрашивает количество подбросов и выдает распределение выпадов для каждой грани. Здесь массив используется для хранения информации о выпадании каждой грани кости — при ее подбрасывании определенное количество раз, т.е. первый элемент массива будет показывать — сколько раз выпала единица, второй элемент — двойка и т.д. Количество граней кости хранится в константе *maxSide,* так что при желании программу можно обобщить для произвольного количества граней. Здесь учитывется, что функция Random(n) генерирует случайные числа от нуля до n-1.

```
Program dice;
```

```
Const maxSide=6;
Var i,j,n:Integer;
    vipady:Array [1...maxSide] of Integer;
BEGIN
    { Ввод количества подбросов }
     Write(' Enter the number of trials: ');
     Read(n);
{ Обнуление исходного количества выпадов }
  For i:=l to n Do vipady[i]:=0;
  Randomize;
{ Имитация подбрасывания }
  For i:=l to n Do
  Begin
          j:=Random(5)+l;
      vipady[j]:= vipady[j]+l;
  End; {For}
{ Вывод результатов }
  Writeln(' The results:');
  For i:=l to maxSide Do Write(i:7);
  Writeln;
  For i:=l to maxSide Do Write(vipady[i]:7);
  Writeln;
END.
```

12.2. Массивы как параметры подпрограмм.

При передаче массива подпрограмме в качестве параметра должно соблюдаться следующее правило, а, именно, тип массива должен быть предварительно объявлен в разделе описания типов. Это означает, что не допускается, например, такая запись:

```
Procedure arrayExample (arr:Array[1...5] of Integer);
```

Вместо этого тип для параметра *arr* должен быть определен предварительно с назначением имени, индексного диапазона и базового типа:

```
Type intArray:Array [1..5] of Integer;
```

...

Procedure arrayExample(a: intArray);

Это единственное ограничение на массивы-параметры. Рассмотрим примеры их применения на задаче обработки квадратичных матриц.

Программа-пример: Сумма, разность и произведение двух квадратичных матриц.

Пусть заданы две квадратичные матрицы: *A* и *B* размерности *n*—*n*. Определим процедуры для их суммирования, вычитания и произведения, а результат представим в третьей матрице *C*. Сначала вспомним формулы для этих вычислений.

Сумма:

$$C_{ij} = \sum_{k=1}^{n} (A_{ik} + B_{kj})$$

Разность:

$$C_{ij} = \sum_{k=1}^{n} (A_{ik} - B_{kj})$$

Произведение:

$$C_{ij} = \sum_{k=1}^{n} (A_{ik} \cdot B_{kj})$$

Определим тип для программного представления матриц. Ранг матриц определим с помощью константы.

```
Const max=100;
Type maxArray=Array[1..max, 1..max] of Byte;

Var A,B,C: maxArray;
```

Параметрами процедур служат следующие величины:

n – фактическая размерность матриц;
A, B – входные матрицы;
C – выходная матрица-результат.

Теперь рассмотрим сами процедуры:

```
Procedure aPLUSb(n:Integer; A,B:maxArray; Var C:maxArray);
{ Суммирование двух матриц }
Var i,j:Integer; { Локальные переменные }
BEGIN
     For i:=1 to n Do
     For j:=1 to n Do
     C[i,j]:=A[i,j]+B[i,j]
END;

Procedure aMINUSb(n:Integer; A,B:maxArray; Var C:maxArray);
{ Вычитание двух матриц }
Var i,j:Integer; {Локальные переменные }
BEGIN
     For i:=1 to n Do
     For j:=1 to n Do
     C[i,j]:=A[i,j]-B[i,j]
END;

Procedure aTIMESb(n:Integer; A,B:maxArray; Var C:maxArray);
{ Произведение двух матриц }
Var i,j,k:Integer;
     r:Byte;     { Локальные переменные }
BEGIN
     For i:=1 to n Do
     For j:=1 to n Do
     Begin
               r:=0;
          For k:=1 to n Do r:=r+ A[i,k]*B[k,j];
               C[i,j]:=r
     End;
```

```
END;
```

Определим еще одну процедуру для вывода матриц на экран:

```
Procedure outputABC(n:Integer; A,B,C:maxArray);
{ Вывод трех матриц на экран }
Var i,j,k,l:Integer;
BEGIN
    1:=5;
    k:=n*l+5;
    Writeln('A':k Div 2, 'B':k, 'C':k);
    For i:=l to n Do
Begin
    For j:=l to n Do Write(A[i,j]:l);
    Write(' ':5);
    For j:=l to n Do Write(B[i,j]:l);
    Write(' ':5);
    For j:=l to n DoWrite(C[i,j]:l);
    Writeln;
      End;
     Writeln;
END; {outputABC }
```

Код главной программы может иметь такой вид:

```
Const max=l00;
Type maxArray=Array [1...max, 1...max] of Byte;
Var i,j,n:Integer;
    A,B,C: maxArray;



    {Описание процедур }

BEGIN
    Write('Введите размерность для матриц:');
    Read(n);
```

```
    Writeln('Определите элементы матрицы A:');
    For i:=l to n Do
    For j:=l to n Do
    Begin
        Write('A[',i,',j, ']=');
    Read(A[i,j])
    End;
Writeln('Определите элементы матрицы B:');
    For i:==l to n Do
    Forj:=l to n Do
Begin
    Write(B[',i,',j,']=');
    Read(B[i,j])
    End;
{ Вычисления}
    aPLUSb(n, A,B, C);
    Writeln('A + B = C:');
    outputABC(n,A,B,C);
    Writeln;
    aMINUSb(n, A,B, C);
    Writeln('A - B = C:');
    outputABC(n,A,B,C);
    Writeln;
    aTIMESb(n, A,B, C);
    WriteIn('A x B = C:');
    outputABC(n,A,B,C);
    Writeln;
END.
```

12.3. Алгоритм упорядочения одномерного массива.

Одним из важнейших механизмов программирования является упорядочение группы элементов по возрастанию или убыванию.

Существуют различные методы разной эффективности. Здесь мы рассмотрим один из них, который иногда называют методом «воздушного шара».

Пусть задан одномерный массив целых чисел *M(n), n=5,* который надо упорядочить по возрастанию. Рассмотрим самый «неупорядоченный» случай:

10 8 6 4 2

Суть метода заключается в том, чтобы последовательно, начиная с первых двух, попарно сравнивать элементы массива, если они расставлены на своих местах (т.е. в данном случае, первый элемент меньше или равен второму), перейти на следующую пару, иначе поменять элементы местами. Так продолжать до тех пор, пока не получим желаемого результата. При *n* придется провести *n–1* сравнений за один проход. Рассмотрим этот процесс по шагам:
I проход:

Номер шага	m_1	m_2	m_3	m_4	m_5
0	10	8	6	4	2
1	8	10	6	4	2
2	8	6	10	4	2
3	8	6	4	10	2
4	8	6	4	2	10

В результате первого цикла попарных сравнений наибольший элемент массива расположился на «своем» месте, но остальные элементы остались все еще неупорядоченными. Сейчас надо еще раз провести цикл попарных сравнений, и так продолжать до тех пор, пока все элементы массива не будут упорядочены. Рассмотрим второй проход попарных сравнений по шагам:

II проход:

Номер шага	m_1	m_2	m_3	m_4	m_5
0	8	6	4	2	10

1	6	8	4	2	10
2	6	4	8	2	10
3	6	4	2	8	10

Как видим, из-за того, что последний элемент был на «своем» месте, количество попарных сравнений пришлось провести три раза вместо четырех. Теперь предпоследний элемент также оказался на «своем» месте. На каждом следующем проходе количество попарных сравнений будет уменьшаться на единицу:

III проход:

Номер шага	m_1	m_2	m_3	m_4	m_5
0	6	4	2	8	10
1	4	6	2	8	10
2	4	2	6	8	10

IV проход:

Номер шага	m_1	m_2	m_3	m_4	m_5
0	4	2	6	8	10
1	2	4	6	8	10

Т.е. для n=5 элементов придется сделать n–1=4 проходов попарных сравнений, при этом, за первый проход — n–1=4 , за второй — n–2=3, за третий — n–3=2, и за четвертый — n–4=1 попарных сравнений, т.е. на i-том проходе — n–i сравнений.

Таким образом, алгоритм упорядочения по возрастанию можно записать с помощью двух структур повтора *Для*:

Для i= от 1 до n–1 Делать
{
Для j= от 1 до *n–i* Делать
{
Если m[j] >m[j+l]
Тогда Поменять местами m[j] и m[j+1]

}
}

Теперь можно уже написать непосредственно программный код:

```
Program sortArray;
Const n=5;
Type arr=Array [1..n] of Integer;
Var i,j,r:Integer;
    m:arr;

BEGIN
    { Генерация случайных элементов массива }
    Randomize;
    For i:=1 to n Do m[i]:=Random(1000);
    { Вывод элементов на экран }
    Writeln('Initial Array:');
    For i:=1 to n Do Write(m[i]:8);
    Writeln;
    { Сортировка}
    For i:=1 to n-1 Do
    For j:=1 to n-i Do
        If m[j]>m[j+1]Then
    Begin
        r:=m[j];
        m[j]:=m[j+1];
        m[j+1]:=r
End;
    { Вывод упорядоченного массива на экран }
    Writeln(' Sorted Array:');
    For i:=1 to n Do Write(m[i]:8);
    Writeln;
END.
```