Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Вятский государственный гуманитарный университет»

Дополнительная подготовка школьников по дисциплине «Информатика и информационные технологии»

Учебный модуль
Массивы и строки

Е. В. Разова

Киров
2011

СОДЕРЖАНИЕ

1. Массивы

1.1. Основные понятия. Виды массивов

Массивы являются одним из наиболее востребованных объектов разделов вычислительной математики. Примерами прикладных задач, при решении которых используются массивы, являются расчеты электрических цепей, статическая обработка результатов наблюдений, задачи линейного и не линейного программирования и многие другие.

Для решения задач обработки больших наборов данных недостаточно простых величин (переменных числовых, логических, символьных типов), поэтому в языках программирования используются составные типы – сложные структуры данных.

Пусть имеется совокупность данных, описывающих какое-либо свойство окружающего мира в различные моменты времени – например, среднедневную температуру воздуха в течение недели. Использовать величину простого числового типа нельзя – в каждый момент времени она может иметь единственное значение, т. е. можно сохранить значение температуры только в определенный день. Вводить несколько разных простых величин нецелесообразно – их может оказаться слишком много (данные о температуре нужны не за неделю, а за год), и работать с ними неудобно. Гораздо эффективнее обозначить имеющиеся данные одним именем и пронумеровать их значения – полученная структура данных и называется массивом.

Большие объемы однотипных данных представляются средствами языка программирования в виде так называемых регулярных типов или массивов.

Массив (*array*) – именованная совокупность фиксированного количества пронумерованных однотипных данных.

Массив характеризуется именем, а также количеством и типом значений. Отдельный элемент массива характеризуется именем (общим для всех элементов массива), номером и значением.

Обращение к элементу массива осуществляется по его номеру, который указывается за именем массива в квадратных скобках.

Пусть массив *T* имеет следующий вид:

Номер	1	2	3	4	5	6	7
Элемент	–3	4.2	0	3.5	7.2	–2.1	0.8

Тогда T [4] = 3.5 ; T [1] = –3 ; T [7] = 0.8 .

Вместо слова «номер» часто используется термин «индекс», потому что в некоторых языках программирования, например в Pascal, в качестве индексов можно использовать не только натуральные числа (номера), но и величины других порядковых (обязательно порядковых!) типов.

На практике нередки ситуации, когда одного индекса недостаточно для однозначного описания элемента массива.

Пример. Если вы стоите пятым в очереди, ваше положение однозначно описывает единственный номер 5, но на билете в кинотеатр требуется указать два номера – номер ряда и номер места. Театральный билет содержит уже три индекса: место, ряд, ярус (партер, амфитеатр, балкон), а в билете на стадион индексов еще больше: место, ряд, трибуна, ярус…

Количество индексов, однозначно определяющих элемент массива, называется *размерностью*.

Обычно выделяют одномерные (линейные) и двумерные (прямоугольные) массивы. Однако можно работать с массивами и большей размерности.

Пример.

Одномерный массив

	1	2	3	4	5
A	5	–6	7	**2**	6

Обращение к выделенному элементу A[4]

Двумерный массив

		1	2	3	4	5
B	1	0	1	1	0	1
	2	0	1	0	1	1
	3	1	0	1	**0**	1

Обращение к выделенному элементу B[3,4]

PascalABC.Net, так же как и FreePascal, поддерживает массивы двух категорий: статические массивы и динамические массивы. Статические, в отличие от динамических массивов, характеризуются фиксированным размером.

1.2. Способы описания массива

Для задания статического массива требуется указать его имя (идентификатор), размерность (вектор, матрица, трехмерный массив и т.д.),

число элементов по каждому измерению и способ индексации элементов в каждом измерении (тип индекса). При объявлении статического массива в явном или косвенном виде указываются конкретные границы изменения каждого индекса.

Конструкция объявления переменной типа массив имеет следующий вид:

<имя типа> : array[<тип(ы) индекса(ов)>] of <тип элемента>;

Одномерный массив:

Var <имя переменной> : Array [n1..n2] Of <тип элементов>;

где *Array* – служебное слово (набор, массив),
N1 – номер первого элемента массива – величина порядкового типа (необязательно числового),
N2 – номер последнего элемента массива – величина порядкового типа (необязательно числового),
Of – служебное слово («из»).

Двумерный массив:

Var<имя переменной> : Array [n1..n2, m1..m2] Of<тип элементов>;

где *N1..N2* – диапазон номеров строк– величины порядкового типа (необязательно числового),
M1..M2 – диапазон номеров столбцов – величины порядкового типа (необязательно числового).

Конструкция задает размерность массива и способ индексации.

Примеры:

```
A : Array[1..10] of Real;
T : Array[-100..100] of Integer;
CubeOfChar : array['a'..'z'] of Char;
```

Конструкция описания массива через предварительное описание пользовательского типа имеет вид:

Type<имя типа>=<определение типа>;

Имя типа – это произвольный идентификатор. Определение типа – синтаксическая конструкция, индивидуальная для различных способов объявления типов данных.

Конструкция описания одномерного массива через предварительное описание пользовательского типа данных:

Type<имя типа>=Array [n1..n2] Of<тип элементов>;
Var<имя переменной> :<имя типа>;

Пример.

```
Type OdMas = Array[1..30] Of Integer;
Var A : OdMas;
```

Стоит отметить, что число элементов массива задается как число элементов порядкового типа, выбранного в качестве диапазона типа индекса. Это означает, что размер памяти, занимаемой массивом, фиксируется во время компиляции и не может быть изменен при исполнении программы – так называемое *статическое распределение памяти*.

Вторая категория массивов – динамические массивы, объявление которых не содержит указания о границах изменения индексов. Для объявления динамического массива используется стандартная конструкция `array` без указания диапазона индексов:

<Имя переменной> : Array of<Тип элемента>;

Примеры объявлений динамических массивов:

```
Sequence : Array of Real;
MatrixOfInt : Array of Integer;
```

Так как при объявлении динамического массива не указывается число элементов массива, то компилятор выделяет 4 байта. Фактическое выделение памяти под динамические массивы производится только во время работы программы путем использования процедуры `SetLength`. Например, `SetLength(Sequence,100)` выделяется память для хранения 100 элементов для динамического массива `Sequence`.

Важно отметить индексы для динамических массивов всегда отсчитываются от 0.

Возникает вопрос, что будет, если вызвать снова процедуру выделение памяти для этого же динамического массива?

`SetLength (Sequence,110)` – в этом случае, значения ранее вычисленных элементов сохраняются, а всем добавляемым элементам присваиваются нулевые значения.

`SetLength (Sequence,90)` – в этом случае, сохраняются значения начальных 90 элементов, оставшиеся 10 будут безвозвратно потеряны.

1.3. Операция присваивания массива

Единственная операция, которую можно выполнять над массивами – операция присваивания одному массиву значения другого массива:

`A := B`,

где `A` и `B` – массивы одинаковой размерности и типа элементов.

Пример.

```
Type my_array = Array [1..100] of integer;
Var a: my_array;
    b: my_array;
    c: array [1..100] of real;
    d,e: array [1..100] of integer;
```

В приведенном примере фрагменты массивов *a, b* совместимы между собой по операции присваивания. Массив *c* не совместим ни с одним из представленных массивов Массивы *d,e* также совместимы между собой по операции присваивания, но не совместимы с другими массивами.

Допустимы, например, команды:

```
a:=b;
d:=e;
```

Но не допустимы команды:

```
e:=b;
c:=e;
d:=c;
c:=a;
```

Все остальные действия, в том числе ввод и вывод, осуществляются поэлементно, т.е. с каждым отдельным элементом массива. Следовательно, для работы с массивами целесообразно использовать конструкцию цикла.

2. Алгоритмы обработки одномерных массивов

При помощи одномерных массивов можно организовать очередь, стек или список данных. Кроме того, одномерным массивом можно представить множество, если по каким-либо причинам отсутствует возможность использования стандартного типа данных «`set of`».

2.1. Способы формирования и вывода массива

1. Ввод значений элементов массива с клавиатуры.

```
Type TArr = Array[1..100] Of Integer;
Var A : TArr;
    N : Integer;
Procedure Init1(Var n : Integer; Var A : TArr);
Var I : Integer;
Begin
     Write('Введи количество элементов в массиве: ');
     Readln(N);
     For i:=1 To N Do Begin
        Write('Введи значение ', i, '  элемента массива: ');
        Readln(A[i])
      End;
End;
Begin
     Init1(n,A);
End.
```

2. Формирование массива генератором случайных чисел.

Для подключения генератора случайных чисел используется процедура `Randomize`. В PascalABC.Net есть несколько вариантов функции `Random`, возвращающей случайное число из заданного промежутка:

`function Random(maxValue: integer): integer;`
– возвращает случайное целое в диапазоне от 0 до *maxValue*–1;

`function Random(a,b: integer): integer;`
– возвращает случайное целое в диапазоне от *a* до *b*;

`function Random: real;`
– возвращает случайное вещественное в диапазоне [0..1).

Пример.

```
Var A : Array[1..100] Of Real;
    B : Array[1..100] Of Integer;
    I, n : Integer;
    x,y : Integer;
Begin
 Randomize;
 Write('Введи количество элементов в массиве: ');
 Readln(N);
 Write('Диапазон значений элементов массива [x, y]: ');
 Readln(x,y);
{Формирование и вывод массива случайных вещественных чисел}
 For i:=1 To N Do A[i]:=x+(y-x+1)*random;
 For i:=1 To N Do Write(A[i],' ');
 WriteLn;
{Формирование и вывод массива случайных целых чисел}
 For i:=1 To N Do B[i]:=random(x,y);
 For i:=1 To N Do Write(B[i],' ');
End.
```

3. Формирование массива по правилу.

Пример. Цикл формирования массива *B* из 10 чисел, элементы которого равны квадратам своих индексов.

```
For i:=1 To 10 Do B[i]:=Sqr(i);
```

Задача. Сформировать массив *F* из первых 10 чисел Фибоначчи:

$$F_1 = F_2 = 1,$$

$$F_n = F_{n-2} + F_{n-1}, \text{ при} n > 2.$$

```
Var F : Array[1..10] Of Integer;
    I : Integer;
Begin
     F[1]:=1; F[2]:=1;
     For i:=3 To 10 Do F[i]:=F[i-2]+F[i-1];
     For i:=1 To 10 Do Write(F[i],' ');
End.
```

4. Вывод элементов массива на экран

Процедура вывода элементов в строку:

```
Procedure Print1(n : Integer; A : TArr);
Var I : Integer;
Begin
 For i:=1 To N Do
     Write(A[i], ' ');
 WriteLn
End;
```

Процедура вывода в столбец:

```
Procedure Print2(n : Integer; A : TArr);
Var I : Integer;
Begin
 For i:=1 To N Do
     WriteLn('A[', i, ']=',A[i]);
End;
```

2.2. Алгоритмы поиска минимума и максимума в массиве

Задача. Задан массив *X*, состоящий из *N* элементов. Необходимо найти индексы элементов и элементы, являющиеся наименьшими и наибольшими в массиве.

Замечание. Следует учитывать, что в массиве возможны совпадающие элементы. Таким образом, минимум и максимум будут нестрогими. То есть в массиве возможны другие элементы, совпадающие с максимумом или минимумом.

Для реализации алгоритма вводятся переменные, содержащие промежуточный минимум и промежуточный максимум. На каждом шаге эти величины соответствуют минимуму и максимуму среди элементов, просмотренных до текущего шага. Первоначально промежуточному минимуму и промежуточному максимуму присваивается значение первого элемента последовательности. Если очередной элемент оказывается меньше промежуточного минимума или больше промежуточного максимума, то он и его индекс запоминаются в соответствующих переменных в качестве очередного промежуточного рекорда. После анализа по этой схеме последнего элемента последовательности промежуточные значения становятся окончательными (см. рис.).

№	1	2	3	4	5	6	7
X[i]	**2**	**7**	**1**	**3**	**9**	**5**	**1**
min	***2***	2	***1***	1	1	1	1
Nmin	***1***	1	***3***	3	3	3	3
max	***2***	***7***	7	7	***9***	9	9

Процедура, выполняющая поиск минимума и максимума среди элементов массива, приведена ниже:

```
Procedure SearchMinMax(var X:TArr; n:Integer;
                       var Min,NMin,Max,NMax:Integer);
Var i:Integer;
Begin
  NMin:=1; Min:=X[NMin];
  NMax:=1; Max:=X[NMax];
     For i:=1 to n Do
          If X[i] < Min Then Begin
               NMin:=i; Min:=X[NMin];
          End;
     If X[i] > Max Then Begin
          NMax:=i; Max:=X[NMax];
     End;
End;
```

Эта процедура всегда позволит найти минимум и максимум среди элементов последовательности. Даже если условные операторы в теле цикла ни разу не выполнятся, то минимум и максимум будут соответствовать первому элементу массива.

Задача. Задан массив *X*, состоящий из *N* элементов. Необходимо найти наименьший положительный элемент последовательности.

Замечание. Здесь положительность искомого минимума – дополнительное условие. Минимум в этом случае называют *условным*. Возможна ситуация, при которой все элементы последовательности окажутся неположительными. Тогда искомое значение не будет найдено.

№	1	2	3	4	5	6	7
X[i]	-2	-7	6	3	-9	5	1
min	?	?	*6*	*3*	3	3	*1*
Nmin	?	?	*3*	*4*	4	4	*7*
найден	ЛОЖЬ	ЛОЖЬ	**ИСТ**	ИСТ	ИСТ	ИСТ	ИСТ

Основная особенность алгоритма поиска условного минимума (условного максимума) состоит в использовании логического флага *«найден»*. Первоначально этот флаг обращён в логическое значение *«ложь»*. Если в процессе анализа элементов массива обнаружено хотя бы одно положительное число, флаг обращается в значение *«истина»*. В этот момент происходит инициализация промежуточного значения минимума. В дальнейшем, новые значения сравниваются с промежуточным минимумом, если выполнено условие положительности элемента. После анализа всего массива флаг *«найден»* показывает, найден ли условный минимум.

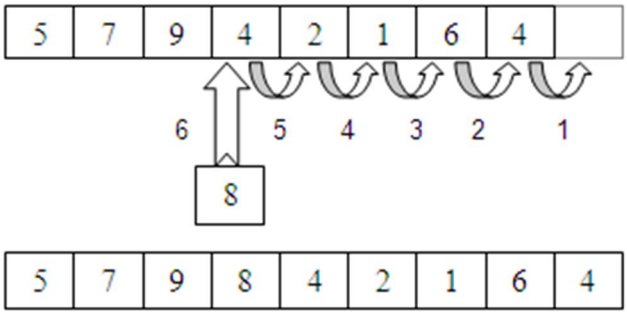
Текст процедуры поиска минимума представлен ниже:

```
Procedure SearchPosMin(var X:TArr; n:Integer;
var Min:Integer; var nMin:Integer; var Found:Boolean);
Var
i:Integer;
Begin
  Min:=0;
  Found:=False;
  For i:=1 to n Do
    If X[i]>0 Then
      If Not Found Or (X[i]<Min) Then Begin
         Found:=True; nMin:=i; Min:=X[nMin];
       End;
End;
```

Инициализация в начале подпрограммы промежуточного минимума нулевым значением (заведомо не удовлетворяющем критерию поиска) необходима для возможности сравнения *X[i] < Min*.

2.3. Алгоритмы вставки и удаления элементов

Задача. Задан массив *X*, состоящий из *N* элементов. Необходимо вставить число *y* в позицию *k*. При этом элементы с номерами от *k* до *n* должны быть перемещены на один индекс ближе к концу последовательности.

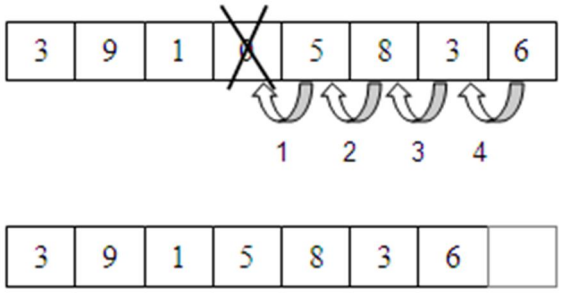


На рисунке стрелки с цифрами показывают порядок перемещения элементов. Как видно из рисунка, элементы массива должны перемещаться на одну ячейку вперёд, начиная с хвоста последовательности. В этом случае потери данных не произойдёт. В результирующем массиве станет на один элемент больше.

Процедура вставки на языке Pascal выглядит следующим образом:

```
Procedure ElInsert(var X:TArr; var n:Integer;
                   Y:Real; k:Integer);
Var
i:Integer;
Begin
  For i:=n downto k Do
         X[i+1]:=X[i];
  X[k]:=Y;
  n:=n+1;
End;
```

Задача. Задан массив *X*, состоящий из *N* элементов. Необходимо удалить из массива элемент с номером *k*. При этом элементы с номерами от *k+1* до *n* должны быть перемещены на один индекс ближе к началу последовательности.



На рисунке стрелки с цифрами показывают порядок перемещения элементов. Как видно из рисунка, элементы массива должны перемещаться на одну ячейку назад, начиная с элемента, следующего за удаляемым. В этом случае потери данных не произойдёт. В результирующем массиве станет на один элемент меньше.

Процедура вставки на языке Pascal выглядит следующим образом:

```
Procedure ElDelete(var X:TArr; var n:Integer;
                   k:Integer);
Var
i:Integer;
Begin
  For i:=k+1 to n Do
     X[i-1]:=X[i];
  n:=n-1;
End;
```

2.4. Поиск элемента на основе линейного просмотра массива

Поиск элемента массива является одной из наиболее распространенных задач обработки массивов.

Задача. Дан массив, состоящий из *n* элементов, и некоторое значение, того же типа, что и элементы массива. Определить содержится ли это значение в массиве.

Пусть переменная *A* – массив, состоящий из n элементов, переменная *X* – искомое значение; переменная *p* – индекс ячейки массива, в которой содержится искомое значение (если искомого значения в массиве нет, то *p*=0).

1-й способ решения:

```
p:=0;
For i:=1 To n Do
 If A[i]=X Then p:=i;
If p>0 Then WriteLn('Искомый элемент содержится в массиве,
его номер', i)
Else WriteLn('Искомого элемента в массиве нет');
```

Однако очевидно, что если искомый элемент стоит на первом месте, то просмотр оставшихся n–1 элемента массива уже лишний. Для того чтобы решить эту проблему следует заменить цикл с параметром на цикл с условием:

```
I:=<минимальное значение индекса элементов массива>;
While <массив не просмотрен> And <элемент не найден> do
<перейти к следующему элементу массива>;
If <значение I не вышло за максимальное значение индексов
элемента> Then <элемент X есть в массиве> Else <элемента X
в массиве нет>;
```

Опишем функцию, возвращающую номер элемента, имеющего искомое значение. Если такого элемента в массиве нет, то функция возвращает значение 0.

```
Function Search(A : TArr; n : Integer) : integer;
Var I : Integer;
Begin
 I:=1;
 While (i<=n) And (A[i]<>X) Do i:=i+1;
 If i<=n Then Search:=I Else Search:=0
End;
```

Замечание. Данная функция возвращает номер первого вхождения искомого элемента в массив.

Для нахождения последнего вхождения элемента в массив, необходимо организовать линейный поиск с конца массива. Опишем процедуру нахождения номера последнего вхождения элемента X в массиве A, состоящем из n элементов:

```
Procedure Search(A: TArr; n: Integer; Var p : Integer);
Var I : Integer;
Begin
 I:=n;
 While (i>=0) And (A[i]<>X) Do i:=i-1;
 If i>=0 Then p:=I Else p:=0
End;
```

2.5. Перестановка элементов массива

Задача. В ряд установлены в произвольном порядке *N* стаканов и *N* кружек. За минимальное число обменов надо переставить их так, чтобы с одной стороны стояли стаканы, а с другой кружки. Составить соответствующий алгоритм (дополнительные массивы и алгоритмы сортировки использовать нельзя).

Обозначим стаканы нулями, а кружки – единицами, и будем хранить информацию о расстановке предметов в одномерном массиве *A*. Предположим, что необходимо расположить предметы так, чтобы слева стояли стаканы, а справа – кружки.

Данная задача похожа на предыдущую задачу. Она усложняется лишь тем, что предметы расставлены в произвольном порядке. Чтобы переставить элементы за минимальное число обменов, будем действовать следующим образом:

1) найдем первую слева кружку:

```
While A[i]=0 Do i:=i+1;
```

2) найдем первый справа стакан:

```
While A[j]=1 Do j:=j-1;
```

3) если найденные предметы находятся не в своих половинах (`i<j`), следовательно, их нужно поменять местами (`Swap(A[i], A[j])`);

4) будем повторять эти действия, пока указатели `i` и `j`, двигавшиеся с концов массива навстречу друг другу, не пересекутся (`i≥j`).

```
Type TArr=Array[1..100] Of Byte;
Var A : TArr;
    n : Byte;

Procedure Init(Var A : TArr; Var n : Byte);
{Формирование массива вводом значений с клавиатуры.}
```

```
Var i : Byte;
Begin
 Write('Введите n<=100'); ReadLn(n);
 Write('Введите информацию о расстановке предметов: 0 -
стакан, а 1 - кружка');
 For i:=1 To n Do
  ReadLn(A[i])
End;

Procedure Print(A : TArr; n : Byte);
{Вывод массива на экран.}
Var i : Byte;
Begin
 For i:=1 To n Do
  Write(A[i]:2);
 WriteLn
End;

Procedure Swap(Var x,y : Byte);
{Меняет местами значения двух переменных x и y.}
Var z : Byte;
Begin
 z:=x; x:=y; y:=z
End;

Procedure Solve(Var A : TArr; n : Byte);
Var i,j : Byte;
Begin
 {Начальные значения указателей.}
 i:=1;
 j:=n;
 {До тех пор, пока указатель i не окажется правее указателя
j повторять}
 Repeat
  While A[i]=0 Do i:=i+1; {поиск первой слева кружки;}
  While A[j]=1 Do j:=j-1; {поиск первого справа стакана;}
  {если предметы стоят не в своих половинах, то следует
поменять их местами.}
  If i<j Then Swap(A[i],A[j])
 Until i>j;
End;
```

```
Begin
 {Формирование массива.}
 Init(A,n);
 {Преобразование массива.}
 Solve(A,n);
 {Вывод полученного массива на экран.}
 Print(A,n);
End.
```

2.6. Сортировки

2.6.1. Постановка задачи

Для решения многих задач необходимо упорядочить данные по определенному признаку. Процесс упорядочения заданного множества объектов по заданному признаку называется *сортировкой*. Для простоты изложения рассматривается одномерный массив из целых чисел (**Type** TArr=**Array**[1..NMax] **Of** Integer; **Var** A:MyArray), значение Nmax определяется в разделе констант). Суть большинства алгоритмов сортировки от такого упрощения не изменяется. Алгоритмы сортировки отличаются друг от друга степенью эффективности, под которой понимается количество сравнений и количество обменов, произведенных в процессе сортировки. В основном мы будем оценивать эффективность количеством операций сравнения (порядком этого значения). Заметим, что элементы массива можно сортировать:

- по возрастанию – каждый следующий элемент больше предыдущего – $A[1] < A[2] < \ldots < A[N]$;
- по неубыванию – каждый следующий элемент не меньше предыдущего, то есть больше или равен предыдущему $A[1] \le A[2] \le \ldots \le A[N]$;
- по убыванию – каждый следующий элемент меньше предыдущего $A[1] > A[2] > \ldots > A[N]$;
- по невозрастанию – каждый следующий элемент не больше предыдущего, то есть меньше или равен $A[1] \ge A[2] \ge \ldots \ge A[N]$.

Научившись выполнять сортировку по одному признаку, изменить ее, чтобы получить сортировку по другому признаку, не составляет особого труда.

2.6.2. Сортировка простым выбором

Рассмотрим идею на примере. Пусть исходный массив *A* состоит из 10 элементов:

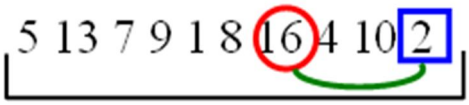
5 13 7 9 1 8 16 4 10 2.

После сортировки массив должен выглядеть так:

1 2 4 5 7 8 9 11 13 16.

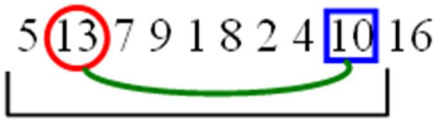
Процесс сортировки представлен ниже. Максимальный элемент текущей части массива заключен в кружок, а элемент, с которым происходит обмен, в квадратик. Скобкой помечена рассматриваемая часть массива.

1-й шаг. Рассмотрим весь массив и найдем в нем максимальный элемент – 16 (стоит на седьмом месте), поменяем его местами с последним элементом – с числом 2.



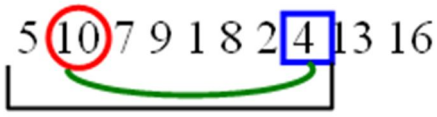
Максимальный элемент записан на свое место.

2-й шаг. Рассмотрим часть массива – с первого до девятого элемента. Максимальный элемент этой части – 13, стоящий на втором месте. Поменяем его местами с последним элементом этой части – с числом 10.



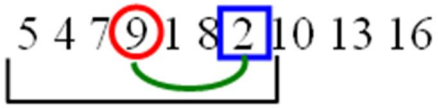
Отсортированная часть массива состоит теперь уже из двух элементов.

3-й шаг. Уменьшим рассматриваемую часть массива на один элемент. Здесь нужно поменять местами второй элемент (его значение – 10) и последний элемент этой части – число 4.

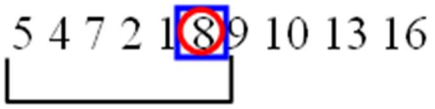


В отсортированной части массива – 3 элемента.

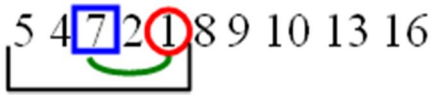
4-й шаг.



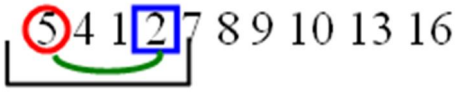
5-й шаг. Максимальный элемент этой части массива является последним в ней, поэтому его нужно оставить на старом месте.



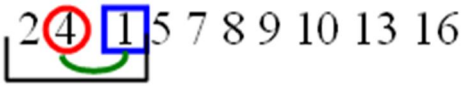
6-й шаг.



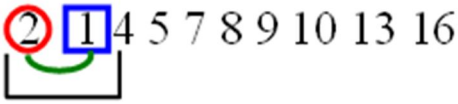
7-й шаг.



8-й шаг.



9-й шаг.



Приведем фрагмент программной реализации.

```
Procedure Sort(N:Integer; Var A:TArr);
Var i, j, k, m: Integer;
{Значение максимального элемента рассматриваемой части массива.}
Begin
 For i:=N DownTo 2 Do Begin
  {Цикл по длине рассматриваемой части массива.}
  {Поиск максимального элемента и его номера в текущей части массива.}
  k:=i;
  m:=A[i];
```

```
  {Начальные значения максимального элемента и его индекса
в рассматриваемой части массива.}
  For j:=l To i-1 Do
   If A[j]>m Then Begin
    k:=j; m:=A[j]
   End;
  If k<>i Then Begin
   {Перестановка элементов.}
   A[k]:=A[i]; A[i]:=m;
  End;
 End;
End;
```

2.6.3. Сортировка простым обменом

Рассмотрим идею метода на примере. Отсортируем по возрастанию массив из пяти элементов: 5 4 8 2 9.

Первый просмотр: рассматривается весь массив

```
i=1   5 4 8 2 9
        > меняем
i=2   4 5 8 2 9
          < не меняем
i=3   4 5 8 2 9
              > меняем
i=4   4 5 2 8 9
                < не меняем
```

Элемент 9 находится на своем месте.

Второй просмотр: рассматриваем часть массива с первого до предпоследнего элемента.

```
i=1    4 5 2 8

         < не меняем

i=2    4 5 2 8

           > меняем

i=3    4 2 5 8

                 < не меняем
```

8 – на своем месте.

Третий просмотр: рассматриваемая часть массива содержит три первых элемента.

```
i=1    4 2 5

         >меняем

i=2    2 4 5

              <не меняем
```

5 – на своем месте.

Четвертый просмотр: рассматриваем последнюю пару элементов.

```
i=1  2 4 5 8 9

         <не меняем
```

4 – на своем месте. Наименьший элемент – 2 оказывается на первом месте. Количество просмотров элементов массива равно `N-1`.

Итак, наш массив отсортирован. Этот метод также называют *методом «пузырька»*. Название это происходит от образной интерпретации, при которой в процессе выполнения сортировки более «легкие» элементы (элементы с заданным свойством) мало-помалу всплывают на «поверхность».

```
Procedure Sort(N:Integer; Var A:TArr);
Var k,i,w: Integer;
```

```
{k – номер просмотра, изменяется от 1 до N-1;
i – номер первого элемента рассматриваемой пары;
w – рабочая переменная для перестановки местами элементов
массива.}
Begin
 {Цикл по номеру просмотра.}
 For k:=1 To N-1 Do
  {Цикл по количеству обрабатываемых пар элементов.}
  For i:=1 To N-k Do
  If A[i]>A[i+1] Then Begin
   {Перестановка элементов.}
   w:=A[i]; A[i]:=A[i+1];A[i+1]:=w;
  End;
End;
```

2.6.4. Сортировка простыми вставками

Сортировка этим методом производится последовательно шаг за шагом. На `i`-м шаге считается, что часть массива, содержащая первые `i-1` элементов, уже упорядочена, то есть `A[1]<A[2]<...<A[i-1]`. Далее берется `i`-й элемент, и для него подбирается место в отсортированной части массива, такое, что после его вставки упорядоченность не нарушается, то есть требуется найти такое `j (0<j<i-1)`, что `A[j]<A[i]<A[j+1]` (при `j=0` происходит только одно сравнение, если не использовать «барьерную» технику). Затем выполняется вставка элемента `A[i]` на место `j+1`. На каждом шаге отсортированная часть массива увеличивается. Для выполнения полной сортировки потребуется выполнить `N-1` шаг.

Рассмотрим этот процесс на примере. Пусть требуется отсортировать массив из 10 элементов по возрастанию.

1-й шаг. 13 6 8 11 3 15 9 15 7 — Рассматриваем часть массива из одного элемента 13. Вставляем второй элемент массива 6 так, чтобы упорядоченность сохранилась. Так как 6<13, записываем 6 на первое место. Отсортированная часть массива содержит два элемента (6 13).

2-й шаг. 6 13 8 11 3 1 5 915 7 — Возьмем третий элемент массива 8 и подберем для него место в упорядоченной части массива. 8>6 и 8<13, следовательно, его место второе.

3-й шаг.	6 8 13 11 3 1 5 9 15 7	Следующий элемент – 11. Он записывается в упорядоченную часть массива на третье место, так как 11>8, но 11<13.
4-й шаг.	6 8 11 13 3 1 5 9 15 7	Далее, действуя аналогичным образом, определяем, что 3 необходимо записать на первое место.
5-й шаг.	3 6 8 11 13 1 5 9 15 7	По той же причине 1 записываем на первое место.
6-й шаг.	1 3 6 8 11 13 5 9 15 7	Так как 5>3, но 5<6, то место 5 в упорядоченной части – третье.
7-й шаг.	1 3 5 6 8 11 13 9 15 7	Место числа 9 – шестое.
8-й шаг.	1 3 5 6 8 9 11 13 15 7	Определяем место для предпоследнего элемента 15. Оказывается, что этот элемент массива уже находится на своем месте.
9-й шаг.	1 3 5 6 8 9 11 13 15 7	Осталось подобрать подходящее место для последнего элемента 7.
	1 3 5 6 7 8 9 11 13 15	Массив отсортирован полностью.

Процедура сортировки:

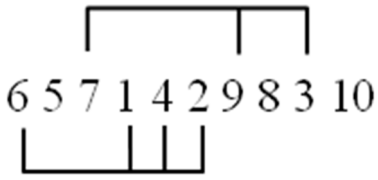
```
Procedure Sort(N:Integer; Var A:TArr);
Var i,j,w:Integer;
Begin
 For i:=2 To N Do Begin
  w:=A[i];
  j:=i-1;
  While (j>0) And (w<A[j]) Do Begin
   A[j+1]:=A[j];
   Dec(j);
  End;
  A[j+1]:=w;
 End;
End;
```

2.6.5. Сортировка подсчетом

Пусть дан массив A из 10 элементов: 10, 5, 11, –5, 1, –4, 13, 12, –4, 13. Возьмем первый элемент, он больше пяти элементов этого массива. Запишем 5 в дополнительный массив счетчиков (Count). Выполним эту операцию для всех элементов массива A. В массиве Count имеем: 5, 4, 6, 0, 3, 1, 8, 7, 2, 9. Если разрешается использовать дополнительный массив для хранения отсортированных данных, то остается переписать каждый элемент исходного массива на соответствующее место в результирующем массиве (B).

```
Procedure Sort(N:Integer; A:TArr; Var B:TArr);
Var i,j:Integer;
    Count:TArr;
Begin
 FillChar(Count,SizeOf(Count),0);
 For i:=N DownTo 2 Do
  For j:=i-1 DownTo 1 Do
   If A[i]<A[j] Then Inc(Count[j]) Else Inc(Count[i]);
 For i:=1 To N Do B[Count[i]+1]:=A[i];
End;
```

Естественное желание – убрать массив B. Ниже по тексту приведена соответствующая процедура. Разберите её. Покажите, что она действительно работоспособна. В качестве подсказки приведем значения элементов массива Count, увеличенные на единицу.




```
Procedure Swap(Var x,y:Integer);
Var w:Integer;
Begin
 w:=x;x:=y;y:=w;
End;
Procedure Sort(N:Integer; Var A:TArr);
Var i,j:Integer;
    Count:TArr;
Begin
 FillChar(Count,SizeOf(Count),0);
```

```
 For i:=N DownTo 2 Do
  For j:=i-l DownTo 1 Do
   If A[i]<A[j] Then Inc(Count[j]) Else Inc(Count[i]);
 For i:=l To N Do Inc(Count[i]);
  For i:=1 To N Do
   While Count[i]<>i Do Begin
    Swap(A[i],A[Count[i]]);
    Swap(Count[i],Count[Count[i]]);
   End;
End;
```

Предположим, что значения элементов массива A принадлежат интервалу [u,v]. Например, A состоит из следующих элементов:

2, 4, 3, 2, 4, 2, 3, 4, 3, 2.

Подсчитаем, сколько раз каждое значение встречается в массиве: двойка – 4 раза, тройка – 3 раза и четверка – 3 раза (массив Count: 4, 3, 3). Изменим значения Count на 4, 7, 10. При этих значениях (Count) элементы массива A можно сразу записывать на свое место в результирующий массив B.

```
Procedure Sort(N:Integer; A:TArr; Var B:TArr);
{Считаем, что значения элементов массива A принадлежат
интервалу [1..4].}
Const u=l;
      v=4;
Var i:Integer;
    Count:Array[u..v] Of Integer;
Begin
 FillChar(Count,SizeOf(Count),0);
 For i:=l To N Do Inc(Count[A[i]]);
 For i:=u+l To v Do Inc(Count[i],Count[i-l]);
 For i:=l To N Do Begin
  B[Count[A[i]]]:=A[i];
  Dec(Count[A[i]]);
 End;
End;
```

А можно ли не использовать массив B? Оказывается, да. В следующей версии процедуры Sort показано, как это сделать.

```
Procedure Sort(t:Integer; Var A:TArr);
Const u=1;
      v=4;
Var i,j,q:Integer;
    Count:Array[u..v] Of Integer;
Begin
 FillChar(Count,SizeOf(Count) ,0);
 For i:=1 To t Do Inc(Count[A[i]]);
 q:=1;
 For i:=u To v Do
  For j:=1 To Count[i] Do Begin
   A[q]:=i;
   Inc(q);
  End;
End;
```

3. Двумерные массивы

При решении практических задач часто приходится иметь дело с различными таблицами данных, математическим эквивалентом которых служат матрицы. Такой способ организации данных, при котором каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен, называется двумерным массивом или таблицей.

3.1. Описание двумерных массивов

Массивы, положение элементов в которых описывается двумя индексами, называются *двумерными*. Их можно представить в виде прямоугольной таблицы или матрицы.

Рассмотрим матрицу A размерностью 2×3, то есть в ней будет две строки, а в каждой строке по три элемента:

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}.$$

Каждый элемент имеет свой номер, как у одномерных массивов, но сейчас номер уже состоит из двух чисел – номера строки, в которой находится элемент, и номер столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, a_{12}– это элемент, стоящий в первой строке и во втором столбце.

Существует несколько способов объявления двумерного массива.

Способ 1. В Pascal двумерный массив можно описать как одномерный, элементами которого являются одномерные массивы. Например, для матрицы *A*, приведенной выше:

```
Const n=2; m=3;
Type TLineArr = Array[1..m] Of <тип элементов>;
     TMatrix = Array[1..n] Of TLineArr;
Var  v: TLineArr;
     A: TMatrix;
```

В данном случае переменная v объявлена как одномерный массив из трех элементов вещественного типа. Переменная A описана как двумерный массив из двух строк, в каждую из которых включено по три элемента.

Способ 2. Описание массива A можно сократить, исключив определение типа TLineArr в определении типа TMatrix:

```
Const n=2; m=3;
Type TMatrix=Array[1..n] Of Array[1..m] Of <тип элементов>;
Var A: TMatrix;
```

Способ 3. Еще более краткое описание массива A можно получить, указывая имя массива и диапазоны изменения индексов для каждой размерности массива:

```
Const n=2; m=3;
Type TMatrix = Array[1..n,1..m] Of <тип элементов>;
Var A: TMatrix;
```

Если указанный тип используется для определения одного массива в программе, то удобно объявление массива в разделе описания переменных:

```
Var A: Array[1..n,1..m] Of <тип элементов>;
```

3.2. Обработка двумерных массивов

Рассмотренные выше методы решения задач обработки одномерных массивов могут применяться для обработки двумерных массивов. Поскольку положение элемента в двумерном массиве описывается двумя индексами (первый – номер строки, второй – номер столбца), программы большинства матричных задач строятся на основе вложенных циклов. Обычно внешний цикл работает по строкам матрицы, то есть с его помощью выбирается требуемая строка матрицы, а внутренний цикл – по столбцам матрицы, то есть здесь выбирается нужный элемент из выбранной уже строки. Для задания значений элементам массива могут быть использованы операторы присваивания и операторы ввода данных.

Пример. В приведенном ниже примере осуществляется ввод и вывод двумерного массива A размерностью 10×15. Формирование и вывод массива описаны в виде двух процедур, которые вызываются последовательно из основной программы. Надо заметить, что формирование двумерного массива можно осуществлять всеми тремя способами, описанными для одномерных массивов, то есть: ввод с клавиатуры, через генератор случайных чисел или с

помощью файла. Пусть в нашем примере элементы задаются генератором случайных чисел.

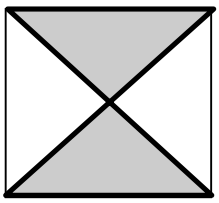
```
Const n=10; m=15;
Type TMatrix = Array[1..n,1..m] Of Integer;
Var  A: TMatrix;
Procedure Init(Var x: TMatrix);
{процедура формирования массива}
Var  i, j:  Integer;
Begin
     For i:=1 To n Do
       For j:= 1 To m Do
          x[i,j]:=-25+Random (51);
End;

Procedure Print(x: TMatrix);
{ процедура вывода массива на экран }
Var  i, j:  Integer;
Begin
     For i:=1 To n Do
       Begin
          {вывод i-ой строки массива}
          For j:=1 To m Do Write(x[i,j]:5);
          Writeln; {переход  на начало следующей строки}
     End;
End;

Begin          {основная программа}
     Init(A); {вызов процедуры формирования массива}
     Writeln('Массив A:');
     Print(A);{вызов процедуры вывода}
     Readln;
End.
```

Формирование двумерного массива по правилу

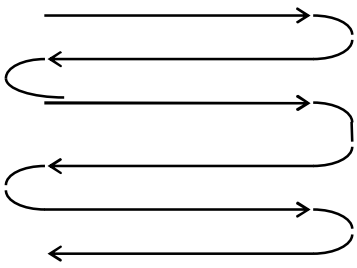
Задача. Сформировать массив A размера $n \times n$ по правилу: элемент массива равен 1, если он принадлежит заштрихованной области, в противном случае элемент равен 0.



```
Procedure Init(Var x: TMatrix);
Var i, j: Integer;
Begin
  For i:=1 To n Do
     For j:=1 To n Do
{если элемент выше главной диагонали, то i<j, если выше
побочной диагонали, то i+j<n}
          If (i<=j) And (i+j<=n+1) Or (i>=j) And (i+j>=n+1)
            Then x[i,j]:=1 Else x[i,j]:=0;
End;
```

Задача. Заполнить массив A размером $n \times m$ следующим образом, например, n=6 и m=8:

```
1  2  3  4  5  6  7  8
16 15 14 13 12 11 10 9
17 18 19 20 21 22 23 24
32 31 30 29 28 27 26 25
33 34 35 36 37 38 39 40
48 47 46 45 44 43 42 41
```



То есть заполняется в виде «змейки». Для того чтобы заполнить, надо вывести правило заполнения, а оно в данном случае будет таким: если ряд нечетный (то есть номер строки – нечетное число), то `A[i,j]=(i-1)*n+j`, иначе (то есть когда строка четная) `A[i,j]=i*n-j+1`. По этому правилу и составляем процедуру заполнения:

```
Procedure Fill(Var x: TMatrix);
Var i, j: Integer;
Begin
     For i:=1 To n Do
       For j:=1 To m Do
          If i Mod 2=1 Then x[i,j]:=(i-1)*n+j
          Else x[i,j]:=i*n-j+1;
End;
```

При решении задач с двумерными массивами можно выделить несколько видов задач.

Нахождение суммы элементов

Можно найти сумму всех элементов, можно только некоторых, которые удовлетворяют данному условию. Но мы рассмотрим более сложный пример.

Задача. Сформировать одномерный массив, каждый элемент которого равен сумме отрицательных элементов соответствующей строки заданной целочисленной матрицы.

Для решения задачи требуется описать одномерный массив, размерность которого равна количеству строк в двумерном массиве.

```
Const n=10; m=15;
Type TLineArr=Array[1..n] Of Integer;
     TMatrix=Array[1..n,1..m] Of Integer;
Var   B: TLineArr;
      A: TMatrix;
```

Формирование одномерного массива по заданному правилу опишем в виде процедуры. Ей будем передавать два параметра – двумерный массив и одномерный, который является результатом. В теле процедуры будут использоваться вложенные циклы. Внешний цикл в ней определяет номер строки, который совпадает с номером элемента одномерного массива. Здесь же задаются начальные значения элементов одномерного массива, равные 0. Во внутреннем цикле анализируется каждый элемент выбранной строки, и если он отрицательный, то добавляется к сумме всех предыдущих отрицательных элементов выбранной строки матрицы.

```
Procedure Sum(x: TMatrix; Var y: TLineArr );
Var  i, j: Integer;
Begin
     For i:=1 To n Do Begin
      y[i]:=0; {задание начальных значений элементов массива суммы}
      For j:=1 To m Do {накопление суммы отрицательных элементов}
       If x[i,j]<0 Then y[i]:=y[i]+x[i,j];
     End;
End;
```

В основной программе обращаемся к процедуре `Sum(A,B)` и остается только вывести на экран одномерный массив `B`, в котором записаны суммы отрицательных элементов каждой строки.

Нахождение количества элементов с данным свойством

Задачи на нахождение номеров элементов с заданными свойствами и на нахождение количества таких элементов во всем массиве останутся практически такими же. В них только добавится второй цикл или вывод двух индексов вместо одного.

Задача. Найти максимальный элемент массива и его индексы.

Так как элементы могут повторяться, то договоримся, что будем запоминать только индексы первого максимального элемента. Опишем процедуру, которой передается массив, и ее результатом является значение максимального элемента и индексы первой встречи такого значения.

```
Procedure Maximum(x: TMatrix; Var max, maxi, maxj:
Integer);
Var i, j: Integer;
Begin
     {начальные значения}
     max:=x[1,1]; maxi:=1; maxj:=1;
     For i:=1 To n Do
       For j:=1 To m Do
          If x[i,j]>max Then
          {присвоение новых}
          Begin
               max:=x[i,j];
               maxi:=i; maxj:=j;
          End;
End;
```

Задача. Найти количество отрицательных элементов в каждой строке. Решать эту задачу можно несколькими способами.

1-й способ – количество элементов каждой строки хранить в одномерном массиве соответствующей размерности. Тогда можно описать такую процедуру:

```
Procedure Q_1(x: TMatrix; Vary: TLineArr);
Var i, j: Integer;
Begin
     For i:=1 To n Do
       Begin
          y[i]:=0;
          For j:=1 To m Do
            If x[i,j]<0 Then Inc(y[i]);
     End;
End;
```

2-й способ – использовать счетчик, находить количество элементов строки и выводить значение на экран.

```
Procedure Q_2(x: TMatrix);
Var i, j, k: Integer;
Begin
     For i:=1 To n Do
       Begin
          k:=0;
          For j:=1 To m Do
            If x[i,j]<0 Then Inc(k);
          Writeln(i,'  -   ', k); {вывод номера строки и
количества заданных элементов}
     End;
End;
```

Работа с несколькими массивами

Задача. Составить программу вычисления произведения двух квадратных целочисленных матриц A и B размером 5×5 соответственно. Элементы результирующей, также целочисленной, матрицы C (размером 5×5) определяются по формуле:

$$c[i,j] = \sum_{k=1}^{n} a[i,k] \cdot b[k,j],$$

где n – размерность матриц A и B.

Формирование матриц будем производить с помощью генератора случайных чисел, вычисление элементов результирующей матрицы C – с

помощью вложенных циклов, где во внутреннем цикле (по параметру k) будет накапливаться сумма, определяющая элемент $c[i, j]$.

```
Const n=5;
Type  TMatrix  =  Array[1..n,1..n] Of Integer;
Var A, B, C:  TMatrix;
Procedure Init(Var x: TMatrix);
...
Procedure Print(x: TMatrix);
...
Procedure Mult(x,y: TMatrix; Var z: TMatrix);
Var  k, i, j:  Integer;
Begin
     For i:=1 To n Do
       For j:=1 To n Do
          Begin
          z[i,j]:=0;
          For k:=1 To n Do z[i,j]:=z[i,j]+x[i,k]*y[k,j];
          End;
End;
Begin {основная программа}
     Writeln('массив A:'); Init(A); Print(A);
     Writeln('массив B:'); Init(B);  Print(B);
     Mult(A,B,C);
     Writeln('массив C:');Print(c);
     Readln;
End.
```

Определение, отвечает ли заданный массив некоторым требованиям

Задача. Определить, является ли данный квадратный массив симметричным относительно своей главной диагонали.

Если он является симметричным, то для него выполняется равенство $a[i,j]=a[j,i]$ для всех $i = 1, ..., n$ и $j = 1, ..., n$ при условии, что $i > j$. Поэтому можно составить следующую функцию:

```
Function Check2(x: TMatrix): Boolean;
Var i, j: Integer;
     t: Boolean;
Begin
     t:= True;
```

```
     For i:=2 To n Do
       For j:=1 To i-1 Do
          If x[i,j]<>x[j,i] Then t:=False;
     Check2:=t;
End;
```

Таким образом, если встретится хотя бы одна такая пара, что соответствующие элементы не будут равны, то значение функции будет *ложь* (*false*).

Изменение значений некоторых элементов, удовлетворяющих заданному свойству

Решение таких задач похоже на решение для одномерных массивов.

Задача. В массиве размерностью $n \times m$ к элементам четных столбцов прибавить элемент первого столбца соответствующей строки.

```
Procedure Substitation1(Var x: TMatrix);
Var i, j: Integer;
Begin
     For i:=1 To n Do
       For j:=2 To m Do
          If  j Mod 2=0 Then x[i,j]:=x[i,j]+x[i,1];
End;
```

Задача. Заменить все отрицательные элементы на противоположные по знаку.

```
Procedure Substitation2( Var x: TMatrix);
Var i, j: Integer;
Begin
     For i:=1 To n Do
       For j:=1 To m Do
          If  x[i,j]<0 Then x[i,j]:=-x[i,j];
End;
```

3.3. Поиск элементов в двумерном массиве

Задача. Определить, есть ли в данном массиве элемент, равный 0.

Для решения опишем логическую функцию, значение которой равно истине, если такой элемент есть, и ложь – в противном случае. Самый простой способ – это просмотреть все элементы и если такой найден, то присвоить значение *True*, иначе оставить *False*.

```
Function Check1(x: TMatrix): Boolean;
Var i, j: Integer;
          t: Boolean;
Begin
     t:= False;
     For i:=1 To n Do
       For j:=1 To m Do
          If x[i,j]=0 Then t:=True;
     Check1:=t;
End;
```

Задача. Определить, есть ли в массиве размерностью $n \times m$ кратный 5, но нечетный элемент.

Для решения используем идею линейного поиска.

```
Function Search(x: TMatrix):Boolean;
Var i, j: Integer;
Begin
     i:=1;  j:=1;
     While (i<=n) And
     ((x[i,j] Mod 5<>0) Or (x[i,j] Mod 2 = 0))  Do
          If j=m Then Begin inc(i); j:=1 End
               Else inc(j);
     Search:=(i>n)
End;
```

Задача. Определить, есть ли в массиве строка, состоящая только из нулевых элементов.

```
Function Search(x: TMatrix):Boolean;
Var i, j: Integer;
Begin
     i:=1;
     Repeat
```

```
            j:=1;
            While (j<=m) And (x[i,j]=0) Do inc(j);
            If j>m Then inc(i);
        Until (j>m) Or (i>n);
        Search:=(i<=n)
End;
```

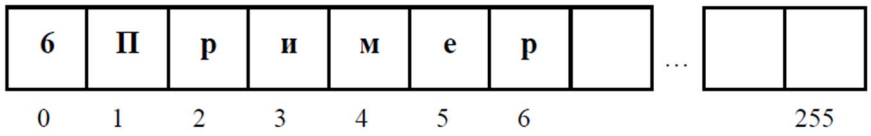
4. Обработка строковой информации

4.1. Понятие строки

Строковый тип данных в языке Pascal представляет собой массив символов длиной не более 255. При этом нулевой байт содержит фактическую длину строки, остальные 255 –символы. Элементами массива принадлежат типу `Char`. Например:

```
Var S: String;
begin
 S := 'Пример';
end.
```

В представленном примере переменная *S* представляет массив символов (см рис.).



Первый байт указывает длину строки равной 6.

При написании программ можно использовать функции и процедуры, работающие над строковыми данными.

4.2. Стандартные операции над строками

1. *Операция склеивания* применяется для объединения нескольких текстов в один и обозначается знаком «+».

Пример	*Результат*
‘мама’+’ ‘ +’мыла‘+’ ’+’раму’	‘мама мыла раму’
‘123’+’321’	‘123321’
‘321’+’123’	‘321123’

2. *Операции сравнения.* Сравнение двух строковых величин осуществляется посимвольно слева направо. Символы сравниваются по их кодам (чем меньше код, тем меньше символ).

Пример
‘String’ < ’string’
‘STRING’=’STRING’
‘string’>’str’
‘string’>’state’

4.3. Стандартные процедуры и функции обработки строк

Процедура / функция	Описание	Пример	Результат
`Delete(s,n,m);`	Удаление из строки *m* символов, начиная с *n*-го	`S:= ‘ABCDE’;` `Delete(s,3,2);`	`S=’ABE’`

`Insert(s1,s2,n);`	Вставка строки *s1* в строку *s2*, начиная с *n*-го символа	`S1:=’ABC’;`
		`S2:=’123’;`
		`Insert(s1,s2,3);`
			`S1=’ABC’`
			
			`S2=’12ABC3’`

`Copy(s,n,m)`	Копирование из строки *sm* символов, начиная с *n*-го	`Text:=‘abcdef’;` `S:=Copy(text,3,2);`	`Text=‘abcdef’` `S=’cd’`
`Length(s)`	Определяет длину текста, т.е. количество символов. Т.к. нумерация символов начинается с 1, то номер последнего символа равен результату функции `length`	`K:=length(‘МЭ 11 (2)’);` `S:=` ` `; {пробел} `R:=length(s);` `S1:=``; {пустая строка}` `L:=length(s1);`	`K=9` `R=1` `L=0`

`Pos(s1,s2)`	Определяет номер первого символа первого вхождения подстроки *s1* в строку *s2*. Если такого вхождения нет, то *k*=0	`S1:=’ши’;`
		`S2:=’У машины шины’;`
		`K1:=Pos(s1,s2);`
		`K2:=Pos(s2,s1);`
			
			`K1=5`
			`K2=0`

`Str(n,s);`	Переводит число *n* в строку *s*	`Str(234,s1);` `Str(3.14,s2);`	`S1=’234’` `S2=’3.14’`

Процедура / функция	Описание	Пример	Результат

`Val(s,n,k);`	Переводит строку *s* в число *n*. Если в строке присутствуют символы, которые нельзя использовать в записи числа, то значение *k* равно номеру первого такого символа, в противном случае *k*=0.	`Val('234',n1,k1);`
		`Val('3.14',n2,k2);`
		`Val('3,14',n3,k3);`
			`N1=234; k1=0`
			`N2=3.14; k2=0`
			`N3=0; k3=2`

Приведенные в таблице процедуры и функции являются основными при работе со строками. С их помощью можно реализовать все основные действия со строковыми переменными. Однако в PascalABC.NET имеется ряд дополнительных подпрограмм для работы со строками. Изучите их самостоятельно, используя справочную систему.

4.4. Примеры задач по обработке строк

Задача. Подсчитать количество вхождений буквосочетания в предложении.

Пусть *S* – исходная строка, *St* – искомое буквосочетание.

```
Var s, st : string;
    k,I,n : integer;
Begin
Write('Введи текст: ');
 Readln(s);
 Write('Введи буквосочетание: ');
 Readln(st); n:=length(st);
 k:=0;
 For i:=1 To length(s)-length(st)+1 Do
   If Copy(s, I, n)=st Then inc(k);
 Writeln('Количество вхождений данного буквосочетания=',k);
End.
```

Задача. Заменить все вхождения подстроки ‘ЭВМ’ на подстроку ’компьютер’.

```
Var s : string;
    k : integer;
```

```
Begin
  Write('Введи текст: ');
  Readln(s);
  k:=Pos('ЭВМ',s);
  While k<>0 DoBegin
    Delete(s,k,3);
    Insert('компьютер',s,k);
    k:=Pos('ЭВМ',s)
  End;
  Writeln(s);
  Readln
End.
```

Задача. Подсчитать сумму всех цифр предложения.

```
Var s : string;
    k,i,sum,n : integer;
Begin
 Write('');
 Readln(s);
 sum:=0;
 For i:=1 To length(s) Do Begin
  Val(s,n,k);
  If k=0 Then sum:=sum+n
 End;
 Writeln(sum);
 Readln
End.
```

Задача. Дано натуральное число. Найти сумму четных цифр числа и составить новое число, поменяв в данном числе порядок цифр на обратный.

```
var  s1,s2:string;
     k:longint;
     i,n,s,c:integer;
Begin
  writeln('Введите число');
  Readln(k);
  s:=0;
  str(k,s1);
  for i:=1 to length(s1) do begin
   val(s1[i],c,n);
```

```
    if c mod 2 =0 then s:=s+c;
    s2:=s1[i]+s2;
  end;
  val(s2,k,n);
  writeln('новое число: ',k);
  writeln('сумма четных цифр равна ',s);
  readln;
End.
```

Задача. Даны две строки. Можно ли из символов первой строки составить вторую, при условии, что каждый символ первой строки можно использовать только один раз и необходимо использовать все символы первой строки.

```
var  s1,s2:string;
     i,j:integer;
     p:boolean;
Begin
  writeln('Введите 1-ю строку');
  Readln(s1);
  writeln('Введите 2-ю строку');
  Readln(s2);
  p:=true;
  i:=1;
  while p and (i<=length(s2)) do begin
   j:=pos(s2[i],s1);
   if j=0 then p:=false
    else Delete(s1,j,1);
   i:=i+1;
  end;
  if (s1='') and p then writeln ('да')
   else writeln('нет');
  readln;
End.
```

Задача. Дана строка. Сформировать новую строку, исправив ошибки в расстановке пробелов в исходной строке (слова должны разделяться ровно одним пробелом, перед знаками препинания пробелы не ставятся, после знаков препинания должен быть один пробел).

Решение. Рассмотрим способ решения задачи, основанный на формировании новой строки. Вставка и удаление символов в строке приводит

к изменению ее длины, а, значит, и к изменению порядковых номеров символов в ней. Следовательно, целесообразнее обработку строки вести с конца, тогда будут изменяться порядковые номера уже обработанных символов, а у необработанных будут оставаться прежними.

```
Var S : String;
    i : Byte;
Begin
 Write('Введите текст: '); ReadLn(S);
 For i:=Length(S)-1 Downto 1 Do
 {Если обозреваемый символ – пробел и за ним стоит пробел
или знак препинания, то обозреваемый пробел лишний, его
следует удалить.}
 If (S[i]=' ') And ((S[i+1]=' ') Or (S[i+1] In
[',','.',':',';','?','!']))
  Then Delete(S,i,1)
 {Если обозреваемый символ – знак препинания пробел и за
ним нет пробела, то после обозреваемого знака препинания не
хватает пробела, его следует вставить.}
  Else If (S[i] In [',','.',':',';','?','!']) And
(S[i+1]<>' ') Then Insert(' ',S,i+1);
 WriteLn(S) {Вывод преобразованного текста.}
End.
```

Задача. Разработать алгоритм подсчета количества в предложении слов, содержащих три слога, и вывода их на экран (в каждом слоге содержится ровно одна гласная буква).

Решение. Для решения задачи введем следующие переменные:

`count_char` – количество гласных букв в очередном слове;

`count_word` – количество слов, состоящих из трех слогов;

`S` – исходное предложение;

`S1` – обрабатываемое слово.

Будем просматривать исходную строку (`S`) посимвольно слева направо и так же посимвольно будем формировать текущее слово (`S1`) предложения и вести подсчет гласных букв в нем. Очередное слово закончится тогда, когда встретиться пробел, и если в этом слове содержится три гласные буквы, то увеличим значение переменной `count_word` на единицу, выведем сформированное слово (`S1`) и начнем процесс формирования следующего слова.

```
Var S, S1 : String;
    count_char, count_word, i : Byte;
Begin
 Write('Введите исходное предложение'); ReadLn(S);
 {Добавление справа к исходному предложению пробела для
удобства обработки последнего слова.}
 S:=S+' ';
 {Начальное значение формируемого слова и счетчиков.}
 S1:='';
 count_word:=0;
 count_char:=0;
 {Посимвольный просмотр исходной строки.}
 For i:=1 To Length(S) Do
  {Если обрабатываемый символ пробел}
  If S[i]=' ' Then Begin
   {и в сформированном слове три гласные буквы}
   If count_char=3 Then Begin
    {увеличение счетчика слов и вывод слова на экран.}
    count_word:=count_word;
    WriteLn(S1)
   End;
  {«Обнуление» S1 для формирования следующего слова.}
  S1:='';
  count_char:=0
  End Else {Если символ не пробел.} Begin
   {Если обрабатываемый символ S[i] - гласная буква, то
увеличение счетчика гласных букв текущего слова.}
   If S[i] in ['а','е','и','о','у','ы','э','ю','я']
    Then count_char:=count_char+1;
   {Добавление символа к формируемому слову.}
   S1:=S1+S[i];
 End;
 Writeln('Количество слов, состоящих из трех слогов ',
count_word)
End.
```

Задача. Даны два предложения (между словами один или несколько пробелов). Напечатать слова, которые есть только в одном из них.

Решение. Преобразуем данные предложения (переменные `Sa` и `Sb`) в массивы слов `A` (из `n` элементов) и `B` (из `m` элементов) соответственно (процедура `Init_Word`). После этого преобразования задача сводится к

поиску в массивах элементов-слов, которые есть только в одном из них, а это можно сделать следующим образом:

1) для каждого значащего (не равного пустому тексту) i-ого элемента массива A
   а) «удаляем» (заменяем на пустой текст) из A элементы, равные A[i];
   б) ищем равные элементы в массиве B, каждый из которых «удаляем» из B (заменяем на пустой текст);
   в) если нашелся хотя бы один парный элемент в B, то «удаляем» и элемент A[i];
2) для оставшихся значащих элементов массива B «удаляем» повторения.

После этих действий в массивах останутся искомые элементы, причем содержаться они будут в них ровно по одному разу. Выводим значащие элементы массивов A и B.

```
Type Ar_Word = Array[1..100] Of String;
Var Sa, Sb : String;
    A, B : Ar_Word;
    n, m : Byte;

Procedure Init_Word(S : String; Var A : Ar_Word; Var n :
Byte);
Var i : Byte;
Begin
 {Каждое слово строки заканчивается пробелом кроме, может
быть, последнего слова, поэтому для удобства работы надо
добавить в конец строки пробел.}
 S:=S+' ';
 n:=0;
 While S<>'' Do {Пока строка не пустая} Begin
  {удаление пробелов в начале строки.}
  While (S[1]=' ') And (S<>'') Do Delete(S,1,1);
  {если строка не пустая}
  If S<>'' Then Begin
   n:=n+1; {увеличение числа слов (еще одно слово
найдено).}
   i:= Pos(' ',S)-1; {Определение позиции первого пробела
(перед ним заканчивается очередное слово).}
   A[n]:=Copy(S,1,i); {Копирование этого слова в массив.}
   Delete(S,1,i); {Удаление его из строки.}
```

```
  End
 End
End;

Procedure Solve(Var A,B : Ar_Word);
Var k, i, j : Byte;
Begin
 For i:=1 To n Do {Для каждого значащего элемента массива
A}
  If A[i]<>'' Then Begin
   {удаление его повторений;}
   For j:=i+1 To n Do If A[i]=A[j] Then A[j]:='';
   {поиск парных в массиве B.}
   k:=0;
   For j:=1 To m Do
   If A[i]=B[j] Then Begin
    k:=k+1; {Подсчет количества парных A[i]-ому в B.}
    B[j]:='' {Удаление их из B.}
   End;
   {Если был найден хотя бы один парный A[i]-ому в B, то
необходимо удалить A[i].}
   If k>0 Then A[i]:='';
  End;
  For i:=1 To m Do {Для каждого значащего элемента массива
B}
   If B[i]<>'' Then {удаление его повторения.}
    For j:=i+1 To m Do If B[i]=B[j] Then B[j]:='';
End;

Procedure Print(A : Ar_Word; n : Byte);
Var i : Byte;
Begin
 For i:=1 To n Do
  If A[i]<>'' Then Write(A[i],' ')
End;


Begin
 {Ввод исходных предложений и преобразование их в массивы
слов.}
 Write('Введите первое предложение '); ReadLn(Sa);
 Init_Word(Sa,A,n);
 Write('Введите второе предложение '); ReadLn(Sb);
```

```
 Init_Word(Sb,B,m);
 Solve(A,B); {Процедура удаления из массивов совпадающих
элементов (шаги 1, 2).}
 {Вывод оставшихся значащих слов.}
 Print(A,n);
 Print(B,m);
End.
```

Задача. Разведчик послал сообщение, состоящее из русских букв, в Россию, зашифровав его следующим образом: число, являющееся порядковым номером первой встреченной в шифровке русской буквы в алфавите, означает, что из шифровки нужно выбирать лишь те буквы, позиция которых в шифровке, кратна этому числу (при этом найденную первую русскую букву в расшифрованное сообщение включать не надо). Каждая встреченная в шифровке точка, означает разделение между словами в сообщении. Все остальные символы принимать во внимание не следует. Расшифруйте полученную в России шифровку.

Входные данные: строка символов шифровки.
Выходные данные: строка символов расшифрованного сообщения.

Пример:
Входные данные:
ВБУП4PZ3A2C!.ОПТУОZ3Б!:ЕУКДРЕАРО!ЛД

Выходные данные:
УРА! ПОБЕДА!

Решение. Пусть `S` – исходная строка (строка символов шифровки), а `SNew` – искомая строка (строка символов расшифрованного сообщения).

Алгоритм решения данной задачи состоит из двух частей:

1) поиск первой русской буквы в исходной строке `((S[j]>='А') And (S[j]<='Я'))`, и определение `k` – ее порядкового номера в алфавите (`k:=Ord(S[j])-Ord('А')+1`);

2) просмотр исходной строки, и формирование из ее символов новой строки по следующему правилу: если `I Mod k=0`, то к строке `SNew` необходимо добавить символ `S[i]`; если `S[i]='.'`, то к строке `SNew` добавить пробел.

```
Var S, SNew : String;
    n, k, i, j : Byte;
Begin
 Write('Введите строку символов шифровки '); ReadLn(S);
 n:=Length(S);  {Значение переменной n – длина исходного
сообщения.}
 {Поиск первой русской буквы в тексте.}
 j:=1;
 While ((S[j]<'A') Or (S[j]>'Я')) And (j<=n) Do j:=j+1;
 {Формирование строки расшифрованного сообщения.}
 k:= Ord(S[j])-Ord('A')+1
 SNew:='';
 For i:=k To n Do
  If (i<>j) And (i Mod k=0) Then SNew:=SNew+S[i]
   Else If S[i]='.' Then SNew:=SNew+' ';
 {Вывод полученной строки.}
 WriteLn('Расшифрованное сообщение - ', SNew)
End.
```

5. Задания для самостоятельного решения

1. Переставить в обратном порядке элементы, расположенные между первым минимальным и последним максимальным элементами массива.

2. На плоскости отмечено несколько точек, и даны их координаты. Составить алгоритм нахождения координат точек, наиболее удаленных от начала координат.

3. Дан одномерный массив неотрицательных целых чисел. Составить алгоритм, определяющий, является ли его минимальный элемент некоторой степенью числа 2.

4. На столе через одну лежат вилки и ложки по *n* штук. За минимальное число обменов переложить их так, чтобы слева лежали ложки, а справа вилки.

5. Даны два одномерных целочисленных массива *A* и *B*. Составить алгоритм, определяющий, можно ли путем перестановки элементов массива *A* получить массив *B* (алгоритмы сортировки использовать нельзя).

6. Даны два двумерных массива одинаковой размерности. Создать третий массив той же размерности, каждый элемент которого равен сумме соответствующих элементов первых двух.

7. Даны два двумерных массива *A* и *B* одинаковой размерности. Создать массив *C*, где каждый элемент равен 1 (*True*), если соответствующие элементы *A* и *B* имеют одинаковый знак, иначе элемент равен 0 (*False*).

8. Составить программу вывода на экран арифметического квадрата, в нем первый столбец и первая строка заполнены 1, а каждый из остальных элементов равен сумме своих соседей сверху и слева.

9. Заполнить массив *A* размером *n*×*m* следующим образом, например, для *n*=5 и *m*=7:

а)
```
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

б)
```
 1  0  2  0  3  0  4
 0  5  0  6  0  7  0
 8  0  9  0 10  0 11
 0 12  0 13  0 14  0
```

10. Заполнить квадратный массив *B* размерностью *n*×*n*, например, для *n*=6:

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

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

11. Найти сумму и количество элементов каждого столбца с заданным условием (хранить эти значения в массивах):

- элементы, кратные *k1* или *k2*;
- элементы, попадающие в промежуток от *A* до *B*;
- элементы, которые являются простыми числами;
- данные элементы положительны и лежат выше главной диагонали.

12. Найти сумму элементов в строках с *k1*-й по *k2*-ую.

13. Найти номера:

- всех максимальных элементов;
- первых отрицательных элементов каждой строки (столбца);
- последних отрицательных элементов каждой строки (столбца).

14. Найти количество элементов в каждой строке, больших (меньших) среднего арифметического элементов данной строки.

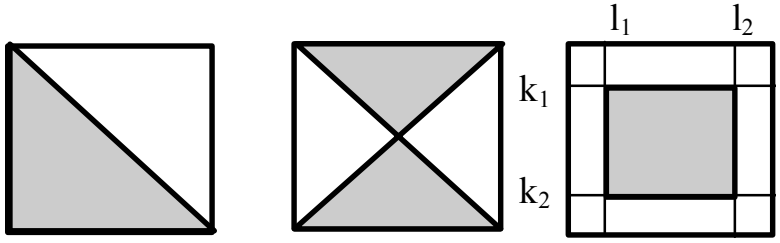
15. Найти произведение двух двумерных массивов *A* и *B*, если массив *A* имеет размерность *n*×*m*, а *B*–*m*×*n*. Определить размерность результирующего массива и правило нахождения элемента с номерами *i* и *j*.

16. Даны два квадратных массива *A* и *B*. Вывести на экран тот из них, у которого след меньше (сумма элементов главной диагонали).

17. Определить:

- есть ли в данном массиве отрицательный элемент;
- есть ли два одинаковых элемента;
- есть ли данное число *A* среди элементов массива;

- есть ли в заштрихованной области массива элемент, равный *A* (массив имеет размерность *n*×*n*):



18. Определить:
- является ли массив логическим квадратом, то есть суммы по всем горизонталям, вертикалям и двум диагоналям должны быть равны;
- добавить к предыдущему условию, что сумма должна быть равна данному числу *A*.

19. Определить, есть ли в данном массиве строка (столбец):
- состоящая только из положительных элементов;
- состоящая только из положительных или нулевых элементов;
- состоящая только из элементов, больших числа *A*;
- состоящая только из элементов, принадлежащих промежутку от *A* до *B*.

20. В каждой строке сменить знак максимального по модулю элемента на противоположный.

21. Последний отрицательный элемент каждого столбца заменить нулем.

22. Положительные элементы умножить на первый элемент соответствующей строки, а отрицательные – на последний, то есть положительные элементы первой строки умножаем на первый элемент первой строки, а отрицательные – на последний элемент тоже первой строки, то же самое и с остальными строками.

23. Заменить все элементы строки с номером *k* и столбца с номером l на противоположные по знаку (элемент, стоящий на пересечении, не изменять).

24. К элементам столбца *k1* прибавить элементы столбца *k2*.

25. Дана строка. Сформировать новую строку, исправив ошибки в расстановке пробелов в исходной строке (слова должны разделяться ровно

одним пробелом, перед знаками препинания пробелы не ставятся, после знаков препинания должен быть один пробел).

26. Дана строка, содержащая минимум две буквы “z”. Изменить ее следующим образом: символы строки расположенные между первой и последней буквами “z”, переставить в обратном порядке.

Пример:

Исходная строка: «wdzkeupmzab»

Результирующее значение строки: «wdzmpuekzab»

27. Даны две строки. Определить, можно ли из символов первой строки и пробелов составить вторую строку. Разрешается использовать не все символы первой строки, и каждый символ можно использовать несколько раз. Например, из строки «ВЕРТИКАЛЬ» можно составить строку «ВЕТКА ЕЛИ».

28. Дан текст. Проверить правильность расстановки круглых скобок в тексте. Скобки в тексте расставлены правильно, если количество открывающих скобок $k_($ равно количеству закрывающих скобок $k_)$ и для любой подстроки с первого по j-й символ $k_(<= k_)$.

Пример:

()(()())(()) – правильная расстановка скобок

()())()(() – неправильная расстановка скобок

29. Дан текст, в котором слова разделены одним или несколькими пробелами. Найти количество слов-палиндромов (слово – палиндром, если оно одинаково читается справа налево и слева направо).

Пример:

исходная строка – «На картине шалаш, а рядом казак»

количество слов-палиндромов – 2 («шалаш», «казак»)

30. Дан текст, в котором слова отделяются друг от друга одним или несколькими пробелами. Найти в нем слова-рифмы для слова максимальной длины (рифма – совпадение трех последних символов).

Литература

Основная литература

1. Окулов С. М. Основы программирования. –5-е изд., испр. – М.: БИНОМ. Лаборатория знаний, 2010.
2. Задачи по программированию / Под ред. С. М. Окулова.– М: БИНОМ. Лаборатория знаний, 2007.
3. Павловская Т. А. Паскаль. Программирование на языке высокого уровня. – СПб.: Питер, 2007.

Дополнительная литература

4. Грогоно П. Программирование на языке Паскаль. – М.: Наука, 1982.
5. Епанешников А.М., Епанешников В.А. Turbo Pascal 7.0.– М: Стрикс, 1997.
6. КультинН. Б. Turbo Pascal в задачах и примерах. – СПб.: БХВ-Петербург, 2000.
7. Немнюгин С. А. Turbo Pascal. – СПб.: Питер, 2000.
8. Фаронов В. В. Turbo Pascal 7.0. Начальный курс. Учебное пособие.– М.: Нолидж, 1997.