КЕМЕРОВСКИЙ ИНСТИТУТ (филиал)
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

# Информатика и программирование

Лебедева Т.Ф.

## Алгоритмы обработки одномерных информационных массивов

Под ***массивом*** в программировании будем понимать упорядоченную конечную группу данных одного типа.

*Индекс* указывает порядковый номер элемента массива. Он может быть числом или выражением целого типа (в общем случае любого порядкового типа)

Количество элементов ,содержащихся в массиве, называется его *размерностью*.

В зависимости от числа индексов, массивы бывают одномерными, двумерными, и т. д.

**Формат объявления массива:**

```
Type
<имя типа> = array[<тип индекса>]    of      <тип компонент>;
Var
<идентификатор> [,<идентификатор>] : <имя типа>;
```

*Тип индекса – один из порядковых типов, чаще всего диапазон, например: 1..10.*

*Тип компонент* – может быть любым, кроме файла и множества.

Массив может быть объявлен также непосредственно при описании переменной в разделе описания переменных:

```
Var
<идентификатор [,<идентификатор>*] : array[<тип индекса>] of <тип компонент>;
```

***Примеры объявления массива:***

```
Type
 r = array [1.. 10] of real;

 Var
mas_int:  array [1.. 45] of integer;
mas_rel:  r;
```

Тип данных Массив позволяет одному идентификатору задать несколько значений, которые отличаются порядковым номером. Номер элемента массива указывается после идентификатора в квадратных скобках (M[5] – пятый элемент массива M).

При описании массива указывается диапазон номеров элементов массива и тип, к которому относится каждый его элемент. Массивы могут быть одно-, двух- и многомерными.

Пример описания и заполнения элементов массива.

```
Var {описание массивов}
M: array [1..5] of integer; {одномерный массив M с номерами элементов от 1 до 5, состоящий из целых чисел}
M1: array [2..3,11..15] of char; {двумерный массив M1 с номерами строк от 2 до 3, с номерами столбцов от 11 до 15, состоящий из символов}
Begin {заполнение массива}
  M[2]:=100; {второму элементу численного массива M присвоено значение 100}
  M1[2,3]:='d'; {элементу второй строки и третьего столбца символьного двухмерного массива M1 присвоено значение 'd'}
End.
```

## Способы задания массива в программе:

1) Задание в разделе типизированных констант

```
Const    gorod : array  [1..3] of string[15] = (‘Москва’, ‘Кемерово’ , ‘Омск’);
```

2) Ввод значений элементов массива с клавиатуры

```
Var    M: array [1.. 30] of real;
       i , n : integer;
Begin
  write (‘количество элементов = ‘);   readln ( n );
  for   i : = 1 to n do
     begin
          write (‘A[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ;
     end;
```

……

3) Формирование массива с помощью датчика случайных чисел

```
Var
M: array [1.. 100] of  byte;      i , n : integer;
Begin
 Randomize; { инициализация генератора случайных чисел}
 write (‘количество элементов = ‘); readln ( n );
 for   i : = 1 to n do
   begin
   A [I ] : = random (10);
   write ( ‘   A[ ‘, I , ’] = ‘, A [ I ]) ;
 end;
```

**Наиболее распространенные алгоритмы обработки одномерных массивов:**

1. нахождение суммы, произведения, среднего значения;
2. нахождение суммы или количества элементов массива, удовлетворяющих некоторым условиям;
3. нахождение минимального (максимального) элемента массива и его номера;
4. создание нового массива из элементов имеющегося;
5. поиск элемента в массиве по заданным критериям;
6. сортировка элементов массива.

Выполним построение математической модели для указанных выше алгоритмов (1 – 4) функциональной задачи денежного оборота торговой фирмы.

Пример 10. Имеются данные о ежедневном обороте торговой фирмы в течение месяца, т.е. одномерный массив, содержащий 30 элементов, A[1..30]:

| ***№ дня*** | ***1*** | ***2*** | ***3. . .*** | ***30*** |
|---|---|---|---|---|
| ***Оборот, тыс. руб*** | ***124*** | ***120*** | ***50. . .*** | ***63*** |

***Математическая постановка***

***а) Обозначение переменных:***

**A[1..30]**– *массив оборота, размерностью 30 элементов;*

**A[ I ]** *– элемент массива A (оборот в i-тый день), тыс. руб.;*

**i** *– счётчик цикла, номер дня;*

***S*** *– общая суммы оборота фирмы за месяц, тыс. руб.;*

***C*** *– средний оборот фирмы за месяц, тыс. руб.;*

***D*** *– план оборота за день, тыс. руб.;*

***K*** *– количество дней с оборотом D;*

***M*** *– минимальный (максимальный) оборот за месяц, тыс. руб.;*

**NM** *– номер дня с минимальным (максимальным) оборотом;*

**B [1..30***] – новый массив отчислений размерностью 30 элементов;*

**B[ I ]** *– элемент массива B (отчисления в i-тый день), тыс. руб.;*

***б) Тип переменных:***

**I , K, NM** *– простые переменные целого типа;*

**S, C, D, M** *– простые переменные вещественного типа;*

**A[ I ], B[ I ]** *- переменные с индексами вещественного типа.*

***в) Классификация по группам:***

*исходные данные:* ***A[1..30]****; промежуточный результат:* ***i****;*
*результаты:* ***S, C, D, M, NM, K, B[1..30]****.*

***г) Системы расчетных формул для различных типов задач:***

***1) определение общей и средней суммы оборота за месяц:***

***S = 0*** сумма обнуляется, чтобы к S не добавилось случайное значение, оставшееся в переменной S от предыдущего вычисления

***i = 1*** *начальный номер элемента*

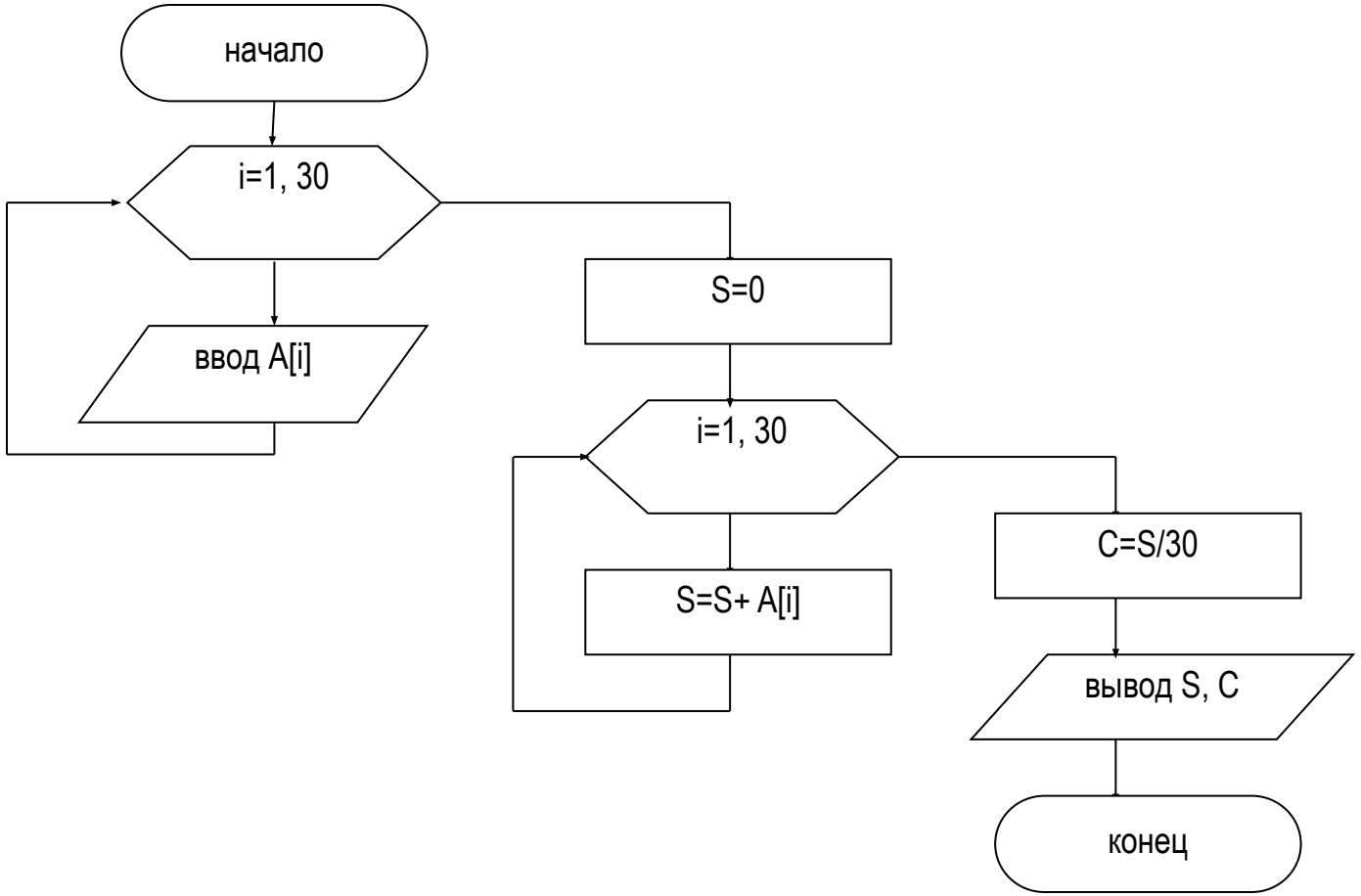
***S = S + A[i]*** накопление суммы в цикле

***i = i + 1*** формирование номера следующего элемента

***Если i≤30, то повторять действия, иначе выход из цикла***

***C = S/30*** нахождение среднего значения

## Представим алгоритм определение общей и средней суммы оборота в виде блок-схемы



```
Program primer_10_1;
Uses CRT;
Var s, c: real;
i : integer;
A : array [ 1..30]  of real;
begin
Clrscr; {очистка экрана}
for i := 1 to 30 do
  begin
     write (‘A[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
S : = 0;
for i : = 1 to 30 do
    s : = s + A [ I ];
 C : = S / 30;
Writeln ( ‘сумма оборота = ‘, S: 8:2, ‘тыс. руб’); {вывод суммы на экран}
Writeln ( ‘средний оборот = ‘, C : 8:2, ‘тыс. руб);
readln;
end.
```

**Примечание.** Блок-схема накопления произведения элементов массива имеет тот же вид, что на предыдущем рисунке. Но первоначальное значение произведения **P:=1**; формула накопления произведения имеет вид: **P:= P * A[i]**.

2) нахождение количества дней с оборотом, равным (≤ , ≥ , ≠ ,>,<) плану **D**:

*г) запись расчетных формул*

**K = 0** обнуление **K**

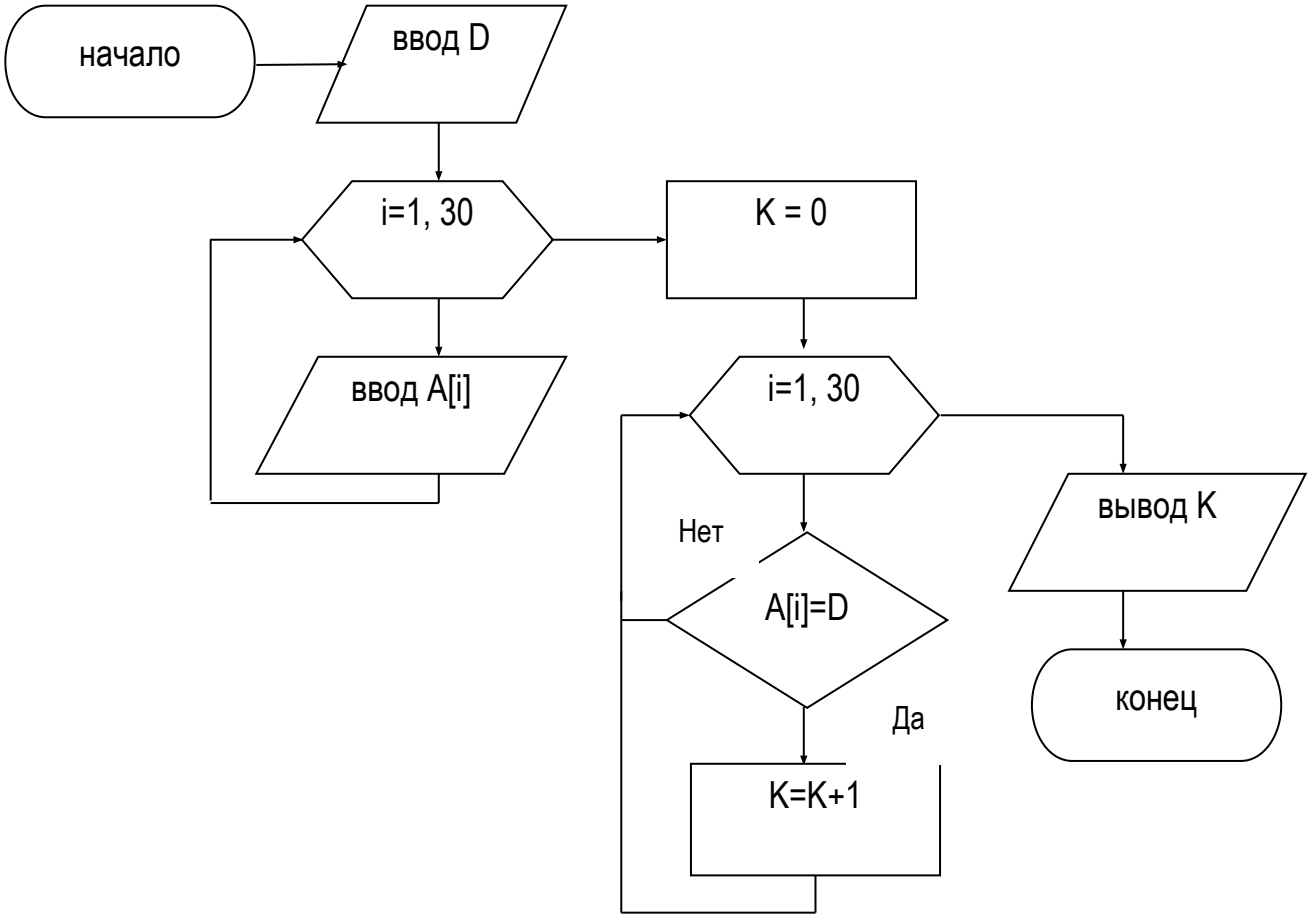
**i = 1** начальный номер элемента

***Если*** **A[ i ]=D**, ***то*** **K = K + 1** накопление количества

**i = i + 1** формирование номера следующего элемента

***Если*** **i ≤ 30**, ***то*** повторять действия, ***иначе*** выход из цикла

Представим алгоритм определение количества дней, в которые план оборота был выполнен, в виде блок-схемы



```
Program primer_10_2;
Uses CRT;
Var d : real;
i , k : integer;
A : array [ 1..30]  of real;
begin
Clrscr; {очистка экрана}
write (‘план оборота = ‘); readln ( d );
for  i := 1 to 30 do
  begin
     write (‘A[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
k : = 0;
for i : = 1 to 30 do
       if  A [ I ] = d     then      k : = k + 1;
Writeln ( ‘количество дней = ‘, k );
readln;
end.
```

3) определение максимального оборота предприятия за данный период и номера дня с максимальным оборотом:

*г) запись расчетных формул*

M = A[1] за начальное значение максимума выбираем первый элемент

NM = 1 начальный номер максимального элемента

i = 2 начальный номер следующего элемента

***Если*** A[ I ] > M, ***то*** M = A[ i ]; NM= I формирование нового максимума и его номера

i = i + 1 формирование номера следующего элемента

***Если*** i ≤ 30, ***то*** повторять действия, ***иначе*** выход из цикла

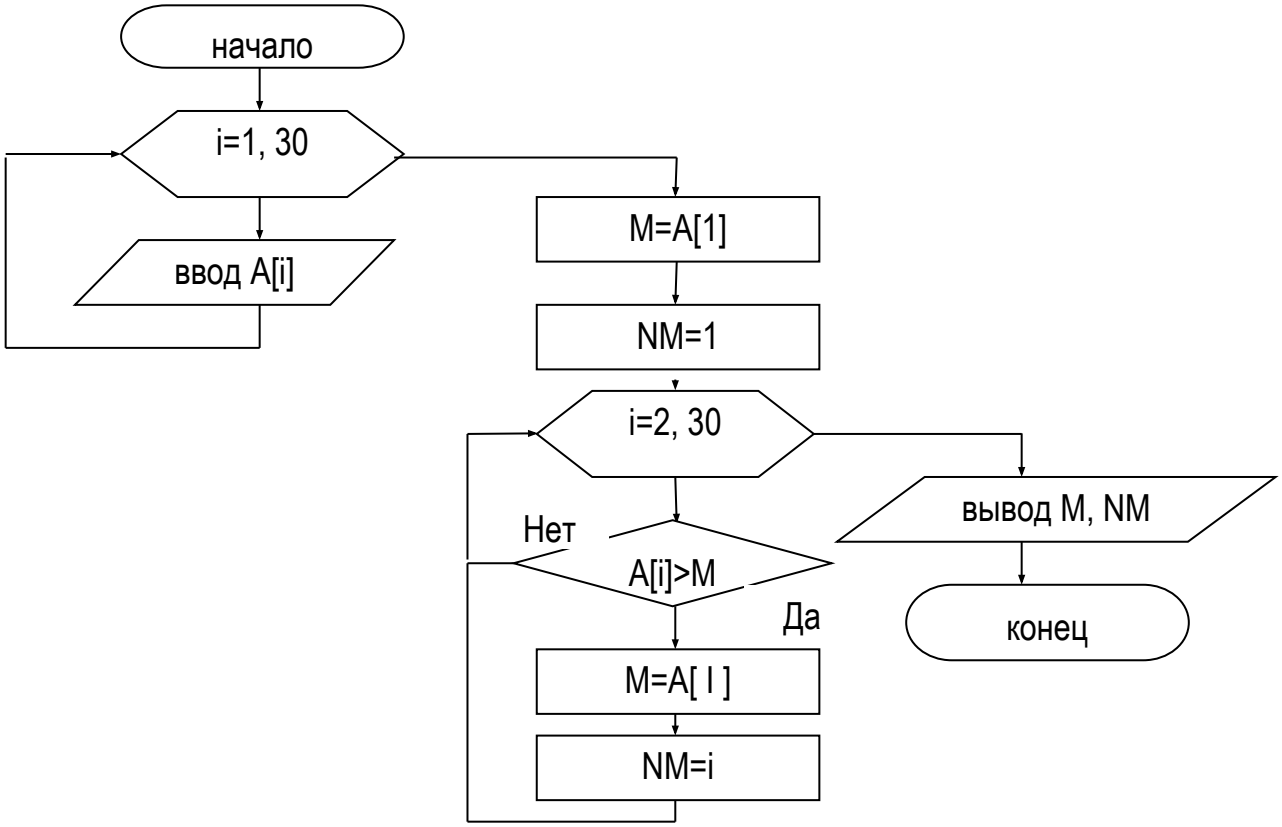
**Примечания:**

а) Если **M** – минимум, то условие имеет вид: **A[ i ]<M.**

б) За начальное значение максимума можно выбрать очень маленькое число ( например 0 или - 10000), тогда начальным номером следующего элемента будет 1.

За начальное значение минимума можно выбрать очень большое число ( например 10000).

## Представим алгоритм определение максимального оборота предприятия за данный период в виде блок-схемы



```
Program primer_10_3;
Uses CRT;
Var   m : real;
i , nm : integer;
A : array [ 1..30]  of real;
begin
Clrscr; {очистка экрана}
for i := 1 to 30 do
  begin
     write (‘A[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
m : = A[ 1 ]; nm : = 1;
for i : = 2 to 30 do
   if  A [ I ] > m    then     begin    m : = A [ i ];  nm : = i; end;
Writeln ( ‘максимальный оборот = ‘, m : 6 :2 , ‘тыс.руб’, ‘  номер дня =’,
          nm);
readln;
end.
```

4) 5% ежедневного оборота в отчисляется в дополнительный фонд заработной платы. Организовать новый массив, содержащий информацию об ежедневном отчислении.

*г) запись расчетных формул*

i = 1 начальный номер элемента

B[ i ] = 0,05*A[ i ] формирование элемента нового массива

i = i + 1

*если* i ≤ 30, *то* повторять действия, *иначе* выход из цикла

**Примечания:**

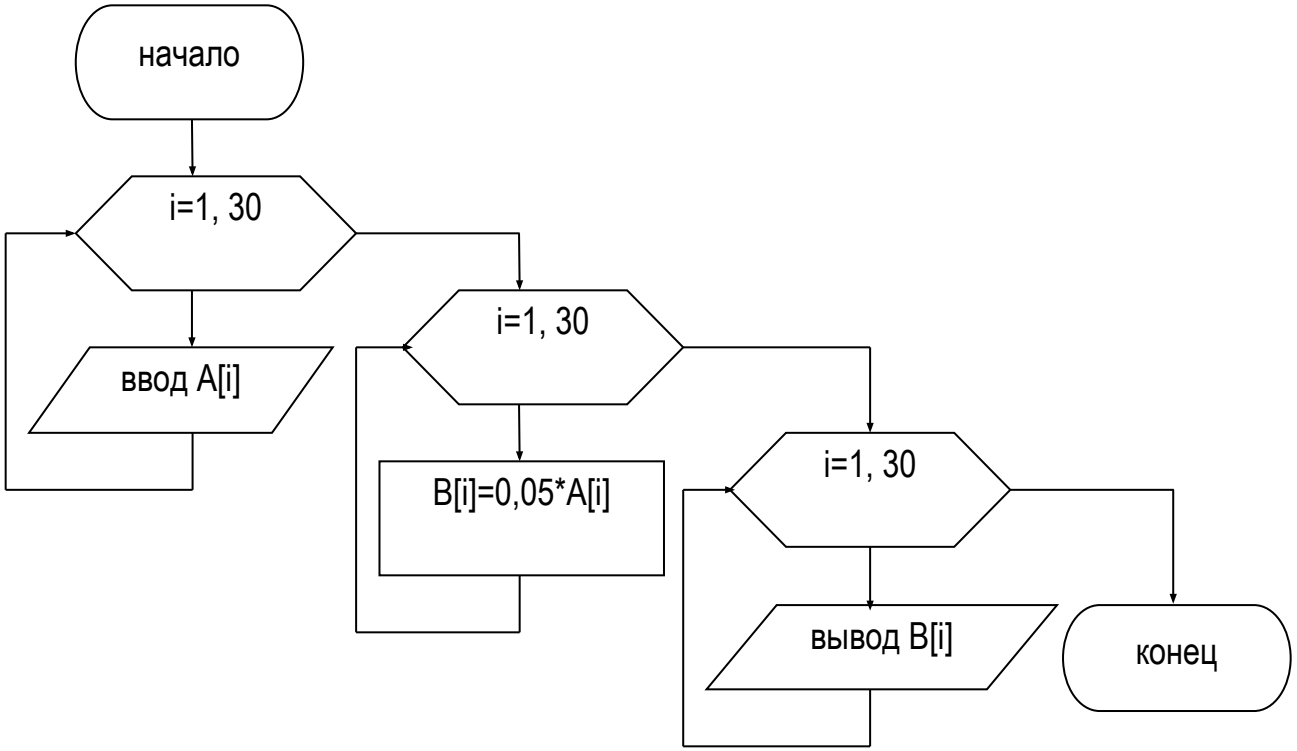
Алгоритмы формирования нового массива могут быть следующих видов:

a) формирование нового массива по некоторой формуле ( например с использованием датчика случайных чисел);

b) формирование нового массива из старого с тем же количеством элементов (пример 10_4);

c) формирование нового массива из старого с другим количеством элементов

## Представим алгоритм организации нового массива в виде блок-схемы



```
Program primer_10_4;
Uses CRT;
Var   i  : integer;
A , B: array [ 1..30]  of real;
begin
Clrscr; {очистка экрана}
for i := 1 to 30 do
  begin
      write (‘A[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
for i : = 1 to 30 do
   B [ I ]  : = 0.05 * A [ I ];
for i : = 1 to 30 do
    Writeln ( ‘отчисление = ‘, B [ I ] : 6 :2 , ‘тыс.руб’, ‘  за ’, i , ‘ день‘);
readln;
end.
```

***Пример 11*** Дан массив A из n целых чисел. Сформировать массив B, в который войдут только числа, кратные 3.

***Математическая постановка***

***а) Обозначение переменных:***

A[1..100 ], B[1..100] - *массивы целых чисел;*

*A[ I ] , B[ I ] – элементs массивов A, B;*

*N – количество элементов в массиве A;*

*i – счётчик цикла;*

*K – количество элементов в массиве B;*

***б) Тип переменных:***

*i, K – простые переменные целого типа;*

*A[ I ], B[ I ] - переменные с индексами вещественного типа.*

***в) Классификация по группам:***

*исходные данные: n, A[1..n];*

*промежуточный результат: i, k;*

*результаты: B[1..k].*

## г) запись расчетных формул

k = 0

i = 1 начальный номер элемента

*если* A[ i ] mod 3 = 0, *то* k = k + 1, B[ k ] = A[ I ]
формирование элемента нового массива

i = i + 1

*если* i ≤ n, *то* повторять действия, *иначе* выход из цикла

*если* k = 0, *то вывод «Массив B пуст» иначе* вывод массива B

```
Program primer_11;
Uses CRT;
Var   I , n , k : integer;
A , B: array [ 1..100]  of integer;
begin
Clrscr; {очистка экрана}
Write ('Введите количество элементов n <= 100');
readln ( n );
for i := 1 to n do
  begin
       write ('A[ ', I , ']  = ') ;  readln ( a[ i ]) ; {ввод массива A}
  end;
k : = 0;
for i : = 1 to n do
   if  A[ i ] mod 3 = 0 then     begin  k : = k + 1;
                                        B [ k ]  : = A [ I ];
                                 end;
If  k = 0  then Writeln ( 'Массив B пуст')  else
for i : = 1 to k do
     Writeln ( 'B [ ' , i,  ' ] =' ,B [ i ] );
readln;
end.
```

**5) Поиск элементов массива по заданным критериям**

Примерами подобного рода задач могут служить поиск первого отрицательного, первого положительного и любого первого элемента, отвечающего некоторому условию, а также поиск единственного или определенного количества элементов, равных некоторому конкретному значению. Особенность задач этого класса в том, что нет необходимости просматривать весь массив. Просмотр можно закончить сразу, как только требуемый элемент будет найден. Однако в худшем случае для поиска элемента требуется просмотреть весь массив, причем нужного элемента в нем может не оказаться.

**а) Поиск первого элемента, удовлетворяющего заданным критериям**

Существует несколько методов поиска. Самый простой заключается в последовательном просмотре каждого элемента массива. Если массив не очень большой, затраты времени линейного поиска не столь заметны.

Но при солидных объемах информации время поиска становится критичным. Поэтому существуют методы, позволяющие уменьшить время поиска, например двоичный поиск, который применяется только, если элементы массива сортированы по возрастанию или убыванию.

Чаще всего при программировании поисковых задач используют циклы while или repeat, в которых условие выхода формируется из двух условий:

1) пока искомый элемент не найден,

2) пока есть элементы массива.

После выхода из цикла осуществляют: проверку, по какому из условий произошел выход.

***Пример 12***. Разработать программу, определяющую первый отрицательный элемент массива. Реализуем структурный алгоритм, в котором для просмотра элементов используется цикл-пока со сложным условием: пока элементы не отрицательны и индекс элемента не вышел за границы массива. Элемент, на котором прервался цикл, если его индекс не превышает размера массива, и есть искомый.

```
Program poisk_1;
Var  a: array[1.. 100] of real;
i, n: integer;
Begin
Writeln('Введите количество элементов n <= 100'); Readln(n);
Writeln('введите ', n,  ' элементов массива ');
for i: =1 to n do Read( a[ i ]);
Readln;
i: = 1; {начальное значение индекса массива}
while ( a [ I ] >=0 )   and ( i <= n ) do i: = i + 1; {пока элемент не отрицателен
и индекс меньше n  - переходим к следующему элементу}
If  i <= n then WriteLn('первый отрицательный элемент ',a[ i ]:6:2, ' имеет
        индекс  ',  i:4)
else  Writeln(' Таких элементов в массиве нет', );
Readln;
End.
```

**б) Поиск первого вхождения элемента, равного заданному значению**

Самый простой способ поиска - последовательный. При последовательном способе сравнивается каждый элемент массива с заданным значением. Однако данный вид поиска является и самым продолжительным по времени. Оценим время поиска. Если искомое значение совпадает с первым элементом массива, то в процессе поиска будет выполнено одно сравнение. Если искомое значение совпадает с последним элементом, то - **n** сравнений. В среднем в процессе поиска понадобится выполнить **(n-1)/2** сравнений.

```
Program poisk_pocl;
Var a: array[1.. 100] of real;
i, n: integer; key : boolean;   s: real;
Begin
Writeln('Введите искомое значение '); Readln(s);
Writeln('Введите количество элементов n <= 100'); Readln(n);
Writeln('введите ', n, ' элементов массива ');
for i: =1 to n do  Read( a[ i ]);
i: = 1; {начальное значение индекса массива}
Key : = false;
while not  key and ( i <= n ) do begin
   key : = (s = a [ I ]); i: = i + 1;
End;
If  key then WriteLn('элемент  найден под номером',   ( i – 1):4)
else Writeln(' Таких элементов в массиве нет', );
Readln;
End.
```

Для ускорения обработки можно реализовать **двоичный (бинарный)** поиск. Этот метод применим, когда массив отсортирован по возрастанию значений. Метод двоичного поиска заключается в следующем:

1) Определяют примерную середину массива
2) проверяют, совпадает ли искомый элемент с элементом в середине массива. Если совпадает, поиск завершен.
3) Если не совпадает, то,
4) если искомый элемент меньше среднего, то поиск продолжают в левой половине массива, иначе - в правой половине.
5) Таким образом, диапазон элементов на каждом шаге уменьшается больше, чем вдвое. Если диапазон сократился до нуля, а элемент не найден, то такой элемент в массиве отсутствует.

Пусть, к примеру, нужно найти элемент **6** в таком массиве:

[2 4 6 8 10 12 14 16 18]

Найдем средний элемент этого массива (**10**) и сравним с ним **6.** После этого все, что больше 10 (да и саму десятку тоже), можно смело исключить из дальнейшего рассмотрения:

**[2 4 6 8]** 10 12 14 16 18

Снова возьмем середину в отмеченной части массива, чтобы сравнить ее с **6.** Однако здесь нас поджидает небольшая проблема: точной середины у нового массива нет, поэтому нужно решить, который из двух центральных элементов станет этой "серединой". От того, к какому краю будет смещаться выбор в таких "симметричных" случаях, зависит окончательная реализация нашего алгоритма. Давайте договоримся, что новой "серединой" массива всегда будет становиться левый центральный элемент. Это соответствует вычислению номера "середины" по формуле

**nomer_sred:= (nomer_lev + nomer_prav) div 2** {(1 + 4) div 2 =2}

Итак, отсечем левую половину массива:

2 4 **[6 8]** 10 12 14 16 18

Из приведенных примера уже видно, что поиск ведется до тех пор, пока не будет найден элемент или левая граница не окажется правее (!) правой границы.

*Реализуем двоичный (бинарный) поиск.*

```
Program poisk_bin;
Var a: array[1.. 100] of integer;
i, n, s, k, m : integer;  key : boolean;
Begin
Writeln('Введите количество элементов n <= 100'); Readln(n);
Writeln('ведите ', n,  ' элементов массива ');
for i: =1 to n do Read( a[ i ]);
Readln;
Writeln(' Исходный массив ');
for i: =1 to n do Write( a[i]: ); Writeln;
WriteLn( 'Введите значение для поиска' );   ReadLn(s);
K : =1; key:=false;
while ( n – k  >= 0)  and  not key    do {пока диапазон положителен и значение не найдено}
    begin
m := (n – k )  div  2;    {определяем среднее значение индекса}
   If   s = a [ m]    then   key := true   {элемент  найден}
    else {уменьшаем диапазон индексов}
   if    s  > a [ m ]  then  k : =  m+ 1 {смещаем левую границу}
   else n: = m - 1 {смещаем правую границу}
   end;
if  key  then   WriteLn(''элемент найден.  Номер равен ', m) else WriteLn ('элемент не найден');
Readln;
End.
```

### 6) Алгоритмы сортировки

Алгоритмы сортировки, предназначенные для упорядочивания расположения элементов (по алфавиту, по убыванию или возрастанию значений), являются важнейшими среди алгоритмов обработки массивов. Достоинство упорядоченного массива состоит в значительном облегчении поиска нужного элемента по сравнению с неупорядоченным массивом.

Критериями оценки различных методов сортировки могут быть:

- количество сравнений и пересылок записей;
- время сортировки заданного объема данных;
- требуемый объем оперативной памяти для сортировки;
- сложность алгоритмов.

Методы сортировки можно подразделить на ***внутренние*** (обрабатывающие массивы) и ***внешние*** (занимающиеся только обработкой файлов).

**Внутренние сортировки делятся на группы:**

1. **сортировки посредством выбора;**
2. **обменные сортировки;**
3. **сортировки вставками;**
4. **сортировки слиянием – объединение двух или более упорядоченных массивов в один.**

Эту лекцию мы посвятим только внутренним сортировкам. Их важная особенность состоит в том, что эти алгоритмы не требуют дополнительной памяти: вся работа по упорядочению производится внутри одного и того же массива.

Рассмотрим алгоритмы сортировки из первых трех групп. При этом будем использовать один массив вещественных чисел и выполнять сортировку по возрастанию значений элементов. Сортировка по убыванию производится аналогичным образом, отличия лишь в знаке операции отношения.

## 1 Сортировка простым выбором

**Алгоритм ПрВыб**

На каждом шаге (всего их будет ровно N-1) будем производить такие действия:

1) найдем минимум среди всех еще не упорядоченных элементов;

2) поменяем его местами с первым "по очереди" не отсортированным элементом.

последний (N-й) элемент массива автоматически окажется максимальным.

**Пример сортировки**

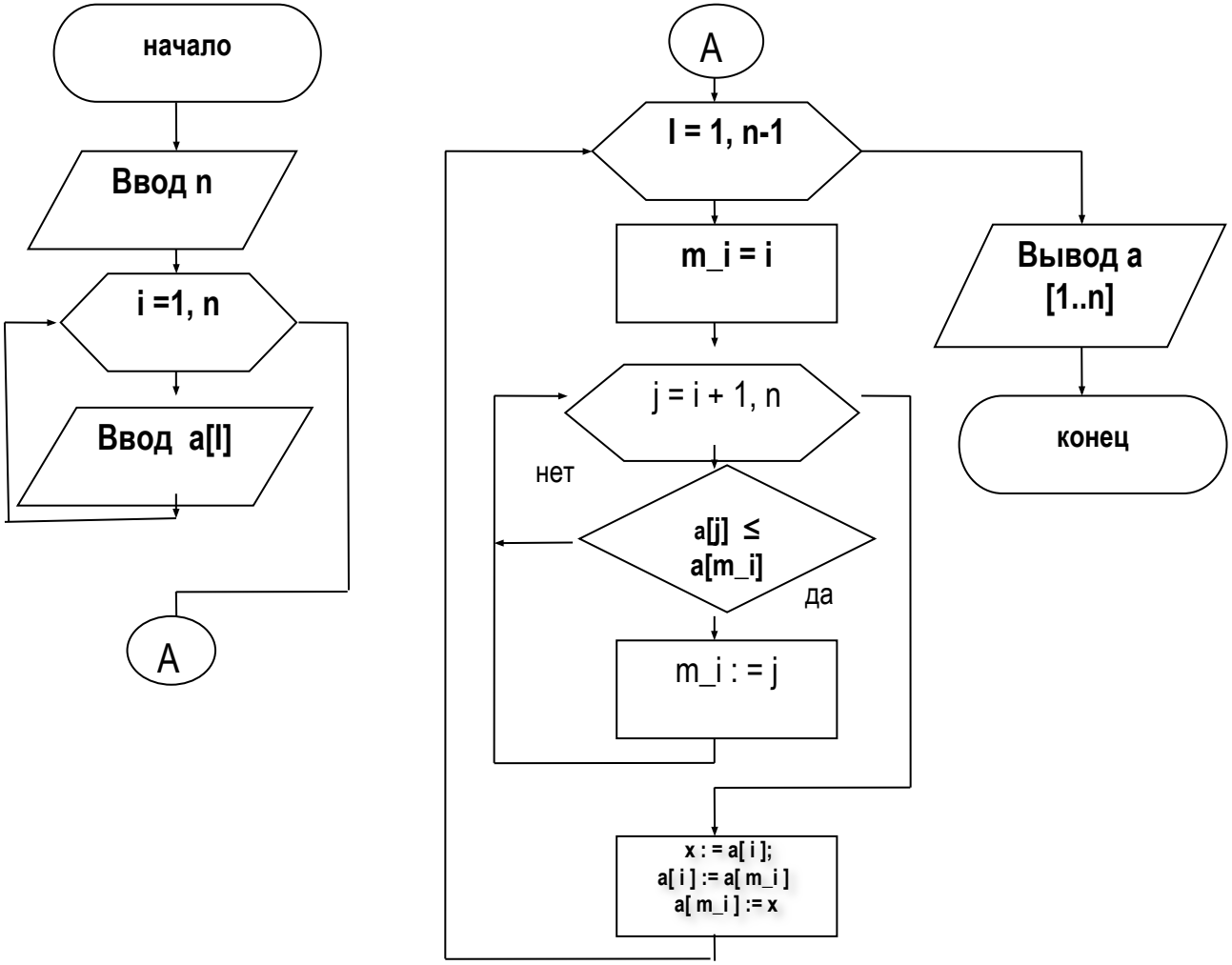
Предположим, что нужно отсортировать набор чисел:

**5 3 4 3 6 2 1**

Теперь мы будем придерживаться алгоритма ПрВыб (подчеркнута несортированная часть массива, а красным цветом выделен ее минимальный элемент):

**1 шаг: 5 3 4 3 6 2 1 {меняем 1 и 5 местами}**
**2 шаг: 1 3 4 3 6 2 5 {меняем 2 и 3 местами}**
**3 шаг: 1 2 4 3 6 3 5 {меняем 3 и 4 местами}**
**4 шаг: 1 2 3 3 6 4 5 {ничего не делаем}**
**5 шаг: 1 2 3 3 6 4 5 {меняем 4 и 6 местами}**
**6 шаг: 1 2 3 3 4 6 5 {меняем 5 и 6 местами}**
**результат: 1 2 3 3 4 5 6**

## *Блок-схема алгоритма сортировки посредством выбора*



**Реализация ПрВыб**

```
Program pr_vib;
Uses CRT;
Var   i , n , j , m_i: integer;
a: array [ 1..100]  of  real;    x : real;
begin
Clrscr; {очистка экрана}
write (‘количество элементов = ‘);   readln ( n );
for i := 1 to n do   begin
   write (‘a[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
for i := 1 to n-1 do begin
     m_i := i;
     for  j := i +1 to n do
        if a[ j ] <= a[ m_i ]   then m_i : = j;
      x : = a[ i ]; a[ i ] := a[ m_i ]  ; a[ m_i ] := x; { перестановка элементов}
end;
Writeln ( ‘отсортированный массив’);
for   i : = 1 to n do
   Write( ‘a [ ‘ , i,  ‘ ] =‘ , ); a [ i ] );  Writeln
readln;
end.
```

## 2 Сортировка прямыми обменами (метод пузырьков)

### Алгоритм ПрОбм

На каждом шаге (пока есть перестановки) будем производить такие действия:

- сравниваем каждый элемент, начиная с первого, с соседним; если он больше следующего, то меняем их местами.

Таким образом элементы с меньшим значением продвинутся к началу массива («всплывут»), а элементы с большим значением – к концу массива («тонут»).

### Пример сортировки

Предположим, что нужно отсортировать набор чисел:

**5 3 4 3 6 2 1**

Теперь мы будем придерживаться алгоритма ПрОбм (подчеркнуты переставляемые элементы):

**1 шаг:** 5 3 4 3 6 2 1 → 3 5 4 3 6 2 1 → 3 4 5 3 6 2 1 → 3 4 3 5 6 2 1 → 3 4 3 5 2 6 1 → 3 4 3 5 2 1 6

**2 шаг:** 3 4 3 5 2 1 6 → 3 3 4 5 2 1 6 → 3 3 4 2 5 1 6 → 3 3 4 2 1 5 6

**3 шаг:** 3 3 4 2 1 5 6 → 3 3 2 4 1 5 6 → 3 3 2 1 4 5 6

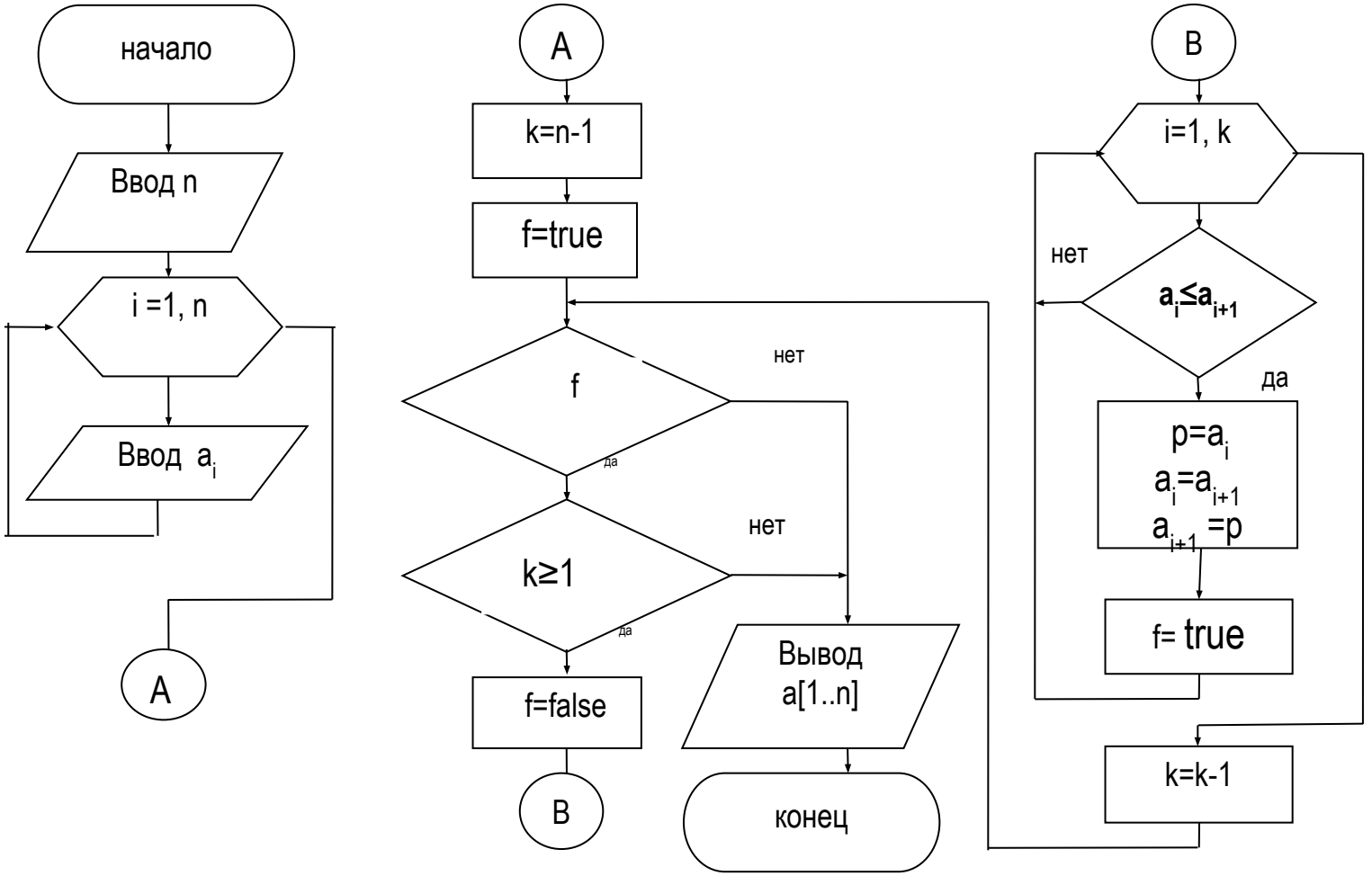
**4 шаг:** 3 3 2 1 4 5 6 → 3 2 3 1 4 5 6 → 3 2 1 3 4 5 6

**5 шаг:** 3 2 1 3 4 5 6 → 2 3 1 3 4 5 6 → 2 1 3 3 4 5 6

**6 шаг:** 2 1 3 3 4 5 6 → 1 2 3 3 4 5 6

**результат:** 1 2 3 3 4 5 6

## *Блок-схема алгоритма обменной сортировки* (методом пузырьков)



*Реализуем алгоритм простого обмене ( метод пузырьков)*

```
Program pr_obm;
Uses CRT;
Var   i , n , k: integer;
a: array [ 1..100]  of  real;    x : real; f: boolean;
begin
Clrscr; {очистка экрана}
write (‘количество элементов = ‘);   readln ( n );
for i := 1 to n do   begin
  write (‘a[ ‘, I , ’] = ‘) ;  readln ( a[ i ]) ; {ввод массива A}
  end;
k : = n – 1; f : = true;
While  f and  (k >=1)  do begin
   f : = false;
  for i: = 1 to k do
     if a[ i ] <= a[ i + 1]   then begin
                                  x : = a[ i ]; a[ i ] := a[i +1]  ; a[i + 1] := x; { перестановка элементов}
                                  f : = true;   end;
      k : = k – 1; end;
WriteIn ( ‘отсортированный массив’);
for i : = 1 to n do
   Write( ‘   a [ ‘ , i,  ‘ ] =‘ , ); a [ i ] );  WriteIn
readln;
end.
```

***3) Сортировка простыми вставками***

**Самый простой способ сортировки, который приходит в голову, - это упорядочение данных по мере их поступления. В этом случае при вводе каждого нового значения можно опираться на тот факт, что все предыдущие элементы уже образуют отсортированную последовательность.**

**Алгоритм ПрВст**

*1) Первый элемент записать "не раздумывая".*

*2) Пока не закончится последовательность вводимых данных,*

*для каждого нового ее элемента выполнять следующие действия:* - начав с конца уже существующей упорядоченной последовательности, все ее элементы, которые больше, чем вновь вводимый элемент, сдвинуть на 1 шаг назад;

3) записать новый элемент на освободившееся место.

При этом, разумеется, можно прочитать все вводимые элементы одновременно, записать их в массив, а потом "воображать", что каждый очередной элемент был введен только что. На суть и структуру алгоритма это не повлияет.

**Реализация алгоритма (фрагмент сортировки без ввода и вывода массива)**

```
……….
for i:= 2 to N do
    if a[ i-1 ] > a[ i ] then      begin
                                       x: = a[ i ];   j := i - 1;
                                       while ( j > 0) and (a [ j ] >x ) do
                                                                  begin a[j+1] := a[ j ]; j:= j - 1 ;   end;
                                      a[ j+1 ] := x;
                                     end;
……..
```

## Алгоритмы обработки двумерных информационных массивов

Многомерные массивы широко используются в статистике и математике, такие массивы имеют более одного измерения (индекса).

Двумерный массив представляет собой набор данных одинакового типа, в котором доступ к любому его элементу осуществляется по двум индексам: номеру строки и номеру столбца. Количество индексов определяет размерность массива.

***Размерность двумерного массива равна произведению числа строк на число столбцов.***

Двумерные массивы являются логической структурой данных удобной для решения задач связанных с обработкой величин зависящих от двух параметров.

Элемент многомерного массива обозначается именем массива с индексами, например,**X [ i, j ]** .

Индексы представляют собой выражения порядкового (чаще целого) типа. Обращаться к элементам массива можно в произвольном порядке, задавая значения индексов. Причем, первый индекс всегда нумерует строки, второй - столбцы.

***Объявление массива***

***1) Вариант***

```
Type
Matrix = array [1..100, 0..9] of real;
Var
  m : Matrix;
  i , j: integer;
```

***2) Вариант***

Пусть задан двумерный массив Matr, имеющий размер 10*20. Этот массив на языке Паскаль может быть описан следующим образом:

```
Var
   Matr : array [1..10,1..20] of integer;
```

**3) Вариант**

Пусть массив Tabl содержит четыре элемента и каждый элемент, в свою очередь, является массивом из трех вещественных чисел.

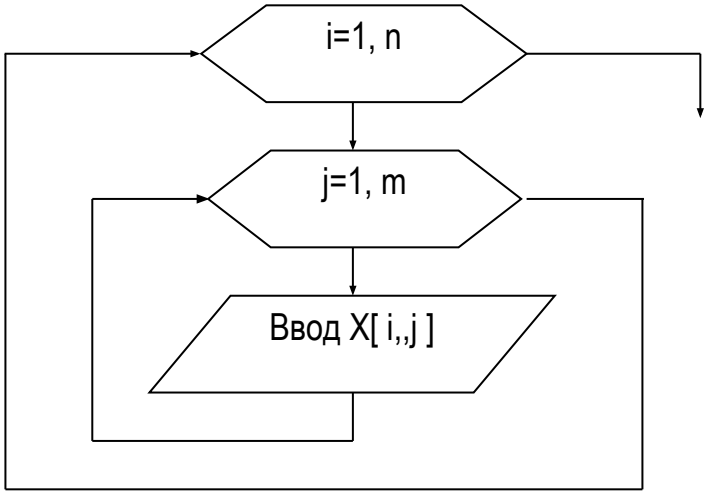
Объявить такой массив можно следующим образом::

```
Type M1 = array [1..3] of real;
M2 = array [1..4] of M1;
Var   Tabl : M2;
Mas1 : M1;
```

Для того чтобы обработать элементы массива (присвоить, ввести, вывести значения), необходимо организовать *вложенные циклы*, в которых перебираются все комбинации значений индексов

***Вложенным циклом или циклом в цикле называется такая структура, когда телом одного цикла является другой.***

***Рассмотрим пример алгоритма реализующего ввод двумерного массива по строкам в виде блок-схемы:***




***В этом примере элементы массива вводятся по строкам в следующем порядке:***

***X11, X 12, ..., X 1m***

***X 21, X 22, ..., X 2m***

***.....***

***Xn1, X n2, ..., X nm***

***Для каждого значения счетчика внешнего цикла i, вложенный цикл j выполняется заданное количество (m) раз.***

*Приведем фрагмент вывода двумерного массива в виде таблицы*

```
Const   n=5; m=6;
Type    MyArray = array [1..n, 1..m] of integer;
Var     A : MyArray;    i, j : integer;
Begin
……..
  for i := 1 to n do    {пробегая последовательно строки и столбцы массива}
    begin
      for j := 1 to m do
        write(X[i, j]:5);        {выведем элемент массива на экран, выделив ему 5 знакомест}
      writeln;           {переход на новую строку}
    end;
end;
```

***Задача 1**. Дана таблица действительных чисел. Сосчитайте сумму всех чисел в таблице.*

*Приведем фрагмент нахождения суммы*

```
……
  S:= 0;
  for i := 1 to n do
    for j := 1 to m do
      S := S+A[i,j];
```

**Обратите внимание, что внутри цикла со счетчиком i организован цикл со счетчиком j. В результате суммируются в начале числа 1-й строки (i=1, при j=1, 2, ..., m), затем суммируются числа 2-й строки (i=2, при j=1, 2, ..., m) и т.д.**

**Пример 12. Выпуск продукции в тыс. руб. представлен таблицей**

Таблица – Исходные данные для примера 12

| № месяца / Вид продукции | 1 | 2 | 3 | ….. |
|---|---|---|---|---|
| 1 | 50 | 56 | 64 | …….. |
| 2 | 64 | 53 | 58 | …… |
| ….. | ……. | ……. | …….. | ……. |

***Выполним построение математической модели и алгоритма решения функциональной задачи выпуска продукции.***

***а) Обозначения переменных:***

**n** – количество видов продукции;

**m** – количество месяцев;

**VP[1..n, 1..m ]**– массив выпуска продукции;

**i** – счётчик цикла, вид продукции; **j** – счётчик цикла, номер месяца;

**S[ i ]** – общий выпуск продукции **i**-того вида;

**Max[ I ]** – максимальный выпуск продукции **i**-того вида;

**K[ i ]** - месяц соответствующий максимальному выпуску продукции**.**

***б) Тип переменных:***

**n, m, i, j** – простые переменные целого типа;

**VP[1..n, 1..m ] , Max[ 1..n ], S[1..n]** – массивы вещественного типа;

**K[1..n ]** – массив целого типа;

**VP[i, j], K[ i ], Max[ i ], S[ i ]** – переменные с индексом;

***в) Классификация по группам:***

исходные данные: **n, m, VP[1..n, 1..m ]**

промежуточные результаты: **i, j;** результаты: **Max[ 1..n ], S[1..n] , K[1..n ]**

*г) Система расчетных формул:*

*1) нахождение общего выпуска продукции каждого вида:*

*(нахождение сумм по строкам)*

*i = 1 начальное значение вида продукции*

*S[ i ] = 0 обнуление суммы выпуска i-того вида продукции*

*j = 1 начальный номер месяца*

*S[ i ] = S[ i ] + VP[ i, j ] накопление суммы в цикле*

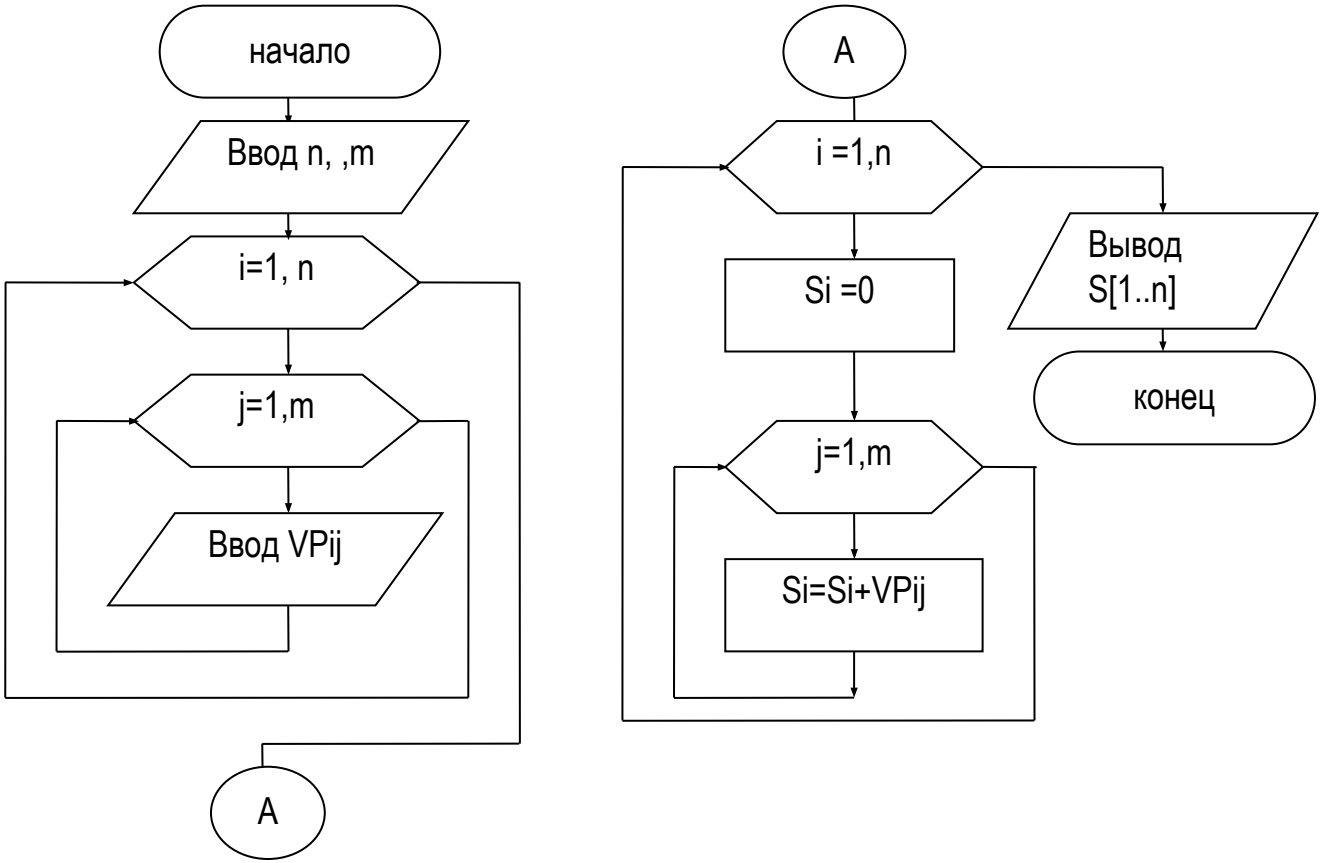
*j = j + 1 следующее значение номера месяца*

*Если j≤m, то повторять действия, иначе выход из цикла*

*i = i + 1 следующее значение вида продукции*

*Если i ≤ n, то повторять действия, иначе выход из цикла*

## *Представим алгоритм нахождения общего выпуска продукции каждого вида в виде блок-схемы*



```
Program sum_str;
Uses CRT;
Var   i , n , j , m : integer;
VP: array [ 1..10, 1..12]  of  real; S : array [1..10]  of  real;
begin
Clrscr; {очистка экрана}
write (‘количество видов продукции = ‘);   readln ( n );
write (‘количество месяцев = ‘);   readln ( m );
Writeln ( ‘введите выпуск продукции по ’, m,  ‘ в строку‘);
for i : = 1 to n do   begin
    for j : = 1 to m do
       read ( VP[ i, j ] ) ; {ввод массива VP}
    Writeln;
  end;
for i := 1 to n do begin
   S[ i ] : = 0;
   for j : = 1 to m do
     S[ i ]: = S[ i ] + VP[ i, j ] ;
  end;
Writeln ( ‘суммарный выпуск’);
for i : = 1 to n do
     WriteLn( ‘S [ ‘ , i,  ‘ ] =‘ , ); S [ i ]: 6 :2, ‘тыс.руб’ );
readln;
end.
```

*г) Система расчетных формул:*

*2) нахождение максимального выпуска продукции каждого вида и месяца с максимальным выпуском:*

*i = 1 начальное значение вида продукции*

*Max [ i ]= VP[ i, 1 ] за начальное значение максимума выбираем выпуск i-той продукции за 1-ый месяц*

*K [ i] = 1 начальный номер месяца*

*j = 2 начальный номер следующего элемента*

*Если VP[ i, j ] > Max [ i ] , то Max [ i ] = VP[ i, j ]; K [ i] = j формирование нового максимума и его номера*

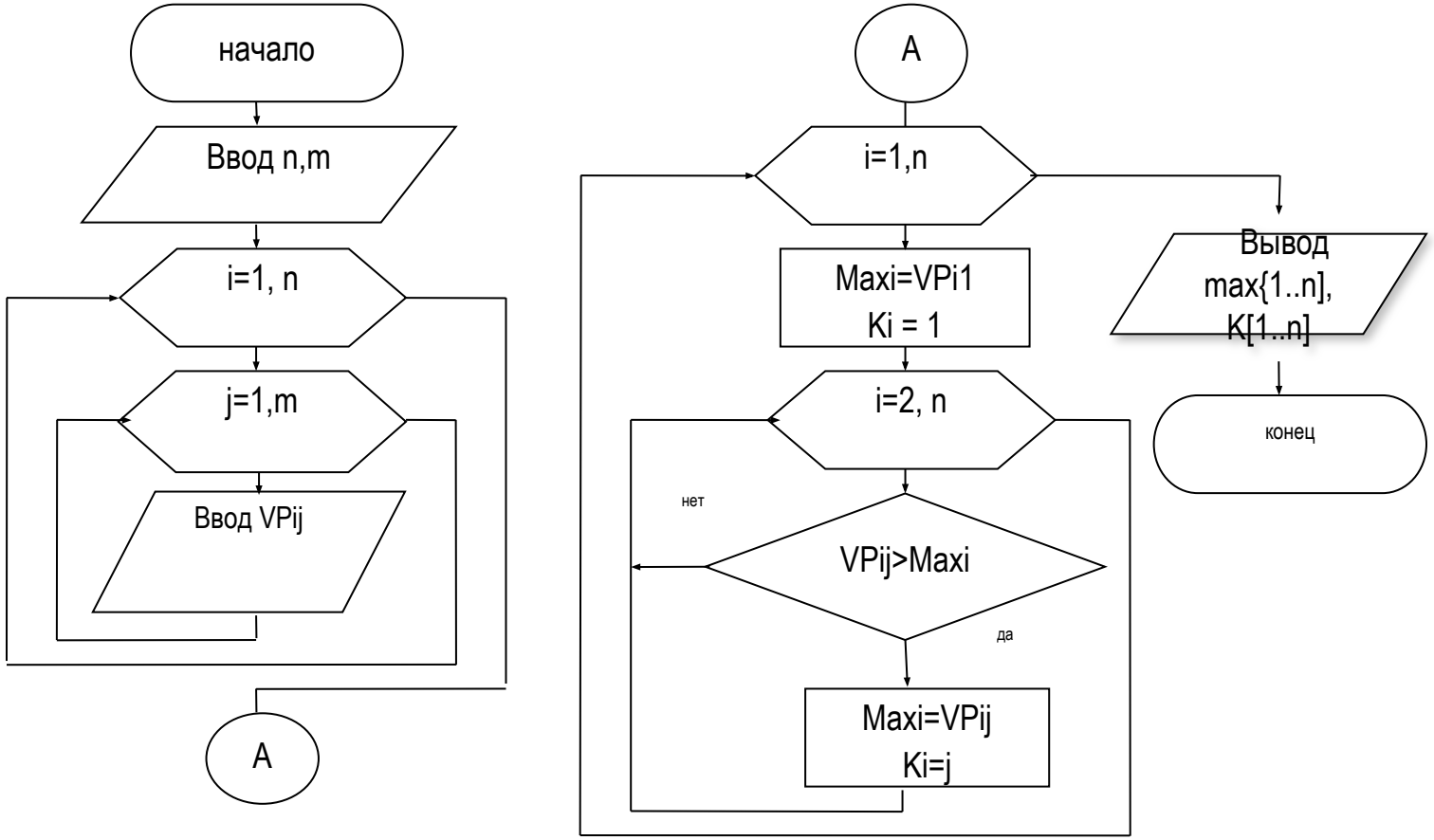
*j = j + 1 следующее значение номера месяца*

*Если j ≤ m, то повторять действия, иначе выход из цикла*

*i = i + 1следующее значение вида продукции*

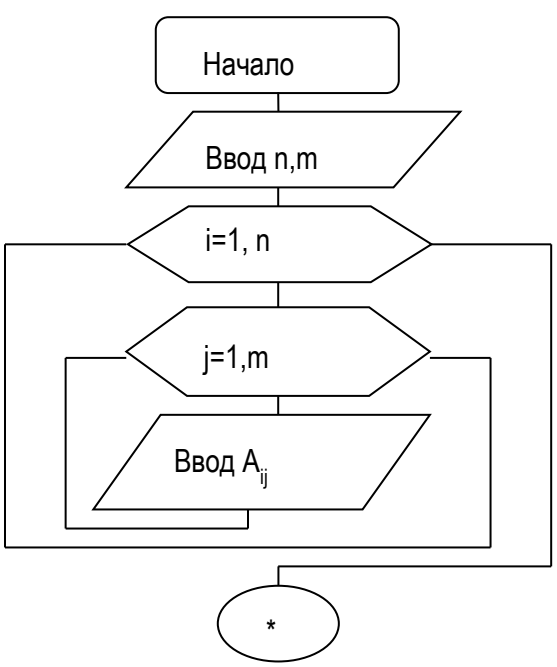
*Если i ≤n, то повторять действия, иначе выход из цикла*

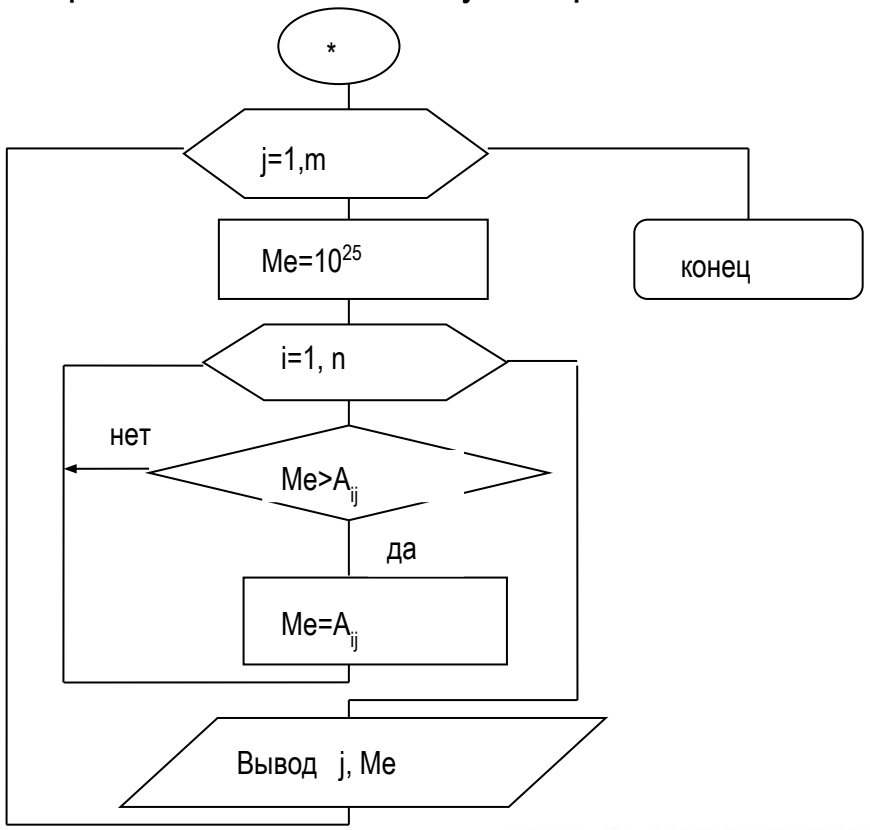
## Представим алгоритм нахождения максимального выпуска продукции каждого вида в виде блок-схемы



```
Program   max_str;
Uses CRT;
Var   i , n , j , m : integer;
VP: array [ 1..10, 1..12]  of  real; Max, K : array [1..10] of  real;
begin
Clrscr; {очистка экрана}
write (‘количество видов продукции = ‘);   readln ( n );
write (‘количество месяцев = ‘);   readln ( m );
Writeln ( ‘введите выпуск продукции по ’, m,  ‘ в строку‘);
for i := 1 to n do   begin
   for j := 1 to m do
     read ( VP[ i, j ]) ; {ввод массива VP}
  Writeln;
 end;
for i := 1 to n do begin
  Max[ i ] : = VP [ i, 1 ]; K [ i ] : = 1;
  for j:= 2 to m do
      if  VP[ i , j ] > Max[ i ]  then begin Max[ i ] : = VP [ i, j ];   K [ i ] : = j  end;
  end;
Writeln ( ‘максимальный выпуск продукции каждого вида  ’);
for i : = 1 to n do
  WriteLn( ‘max [ ‘ , i,  ‘ ] =‘ , ); ‘   номер месяца = ’, K [ i ] );
readln;
end.
```

**Задача 2**: Определить min элемент в каждом столбце массива A, содержащего n строк и m столбцов. Исходными данными для решения являются целочисленные переменные n и m, а также элементы массива A. Результатом решения будут значения минимальных элементов для каждого столбца массива. Представим блок-схему алгоритма.



***Задача 3***. Найти количество отрицательных элементов в каждом столбце.

количество элементов каждой строки хранить в одномерном массиве (Y) соответствующей размерности.

Приведем фрагмент нахождения количество отрицательных элементов в каждом столбце.

```
……..
for j := 1 to m do
    begin
      Y[ j ] := 0;          {записываем начальное значение количества элементов в
          соответствующую столбцу ячейку}
      for i := 1 to n do
        if X[i, j] < 0      {если отрицательный элемент найден}
          then
            Inc(Y[ j ];  {то увеличиваем текущее значение на единицу}
    end;
```

***Задача 4. В квадратной матрице найти суммы диагональных элементов***

Приведем фрагмент программы

```
……..
S1:= 0; S2:= 0;
  for i := 1 to n do begin
       S1 := S1+A[ i, i ]; S2 := S2+A[ i,  n- i+1]; end;
```

## Работа с переменными символьного и строкового типа данных

*Паскаль поддерживает стандартный символьный тип* ***Char*** *и динамические строки, описываемые типом* ***String*** *или* ***String[n]****.*

***Символьный тип данных (char).***
*Описание: var S: char;*

Значение типа **Char** – это непустой символ из алфавита компьютера, заключенный в одиночные кавычки. Кроме этой классической формы записи Turbo Pascal вводит еще два способа:

*1) представление символа его кодом* ***ASCII (American Standart Code for Interchange Information),*** *для этого используется префикс* ***#, например, #97 для символа 'a' или #65 – 'A'****;*

*для управляющих символов коды в диапазоне от 0 до 31:*

*#13 – клавиша Enter, #32 - символ «пробел»*

*2) представление символа его клавиатурным обозначением:* ***'G', '+', '>'***

Примечание: Символы с кодами от 0 до 127 представляют так называемую основную таблицу кодов ASCII. Эта часть идентична на всех IBM-совместимых компьютерах. Коды с символами от 128 до 255 представляют национальную часть – в России коды русских букв (кириллицу).

Так как символы языка упорядочены, то к символьным данным применимы операции сравнения. Операция сравнения осуществляется следующим образом: из двух символов меньше тот, который встречается в таблице ASCII раньше (т.е. код его меньше).

Например: ‘B’ > ‘A’

Иногда в программах возникает необходимость по коду определить символ и, наоборот, по символу определить его код. Для этого используют функции:

Функция **CHR ( X:Byte) : Char**

Эти функция возвращают символ, соответствующий ASCII -коду числа X.

Для определения кода по символу используют функцию ORD.

Функция **ORD ( C:Char) : Byte**

Функция **UPCASE ( C:Char) : Char** - преобразует символ из строчного в прописной. Эта функция рассчитана на обработку только одного символа.Она не преобразует символы кириллицы (русские буквы).

**Строка** – это последовательность символов, не превышающая 255-ти символов. Строковые константы обязательно заключается в апострофы.

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

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

***var s: string;***

n- максимально возможная длина строки- целое число в диапазоне 1..255.

Переменные типа **STRING** объявляется, как правило, путем указания имени переменной, зарезервированного слова **STRING** и указания (в квадратных скобках) максимального размера (длины) строки, которая может хранится в этой переменной. Если максимальный размер строки не указан, то он автоматически принимается равным 255- максимально возможной длине строки.

Пример.

```
Var s:string; {описание идентификатора s как строковой переменной}
Begin
s:='Привет'; {задание значения строковой переменной}
Writeln(s); {распечатка на экране слова "Привет"}
end.
```

## Операции со строками

В Турбо Паскале существуют два пути обработки переменных типа STRING.

Первый путь предполагает обработку всей строки как единого целого, т. е. единого объекта.

Второй путь рассматривает строку как составной объект, состоящий из отдельных символов, т.е. элементов типа CHAR , которые при обработке доступны каждый в отдельности.

Тип ***String*** -это массив символов ***Array [0..255] of char*** .

В нулевом элементе строки хранится ее длина (динамическая «переменная»).

Например, var st: string

i := ord(st [0] ) ; { i – текущая длина строки}

Так, первый путь предоставляет возможность присвоения значения строчной переменной за одну операцию.

Строковые константы записываются как последовательности символов, ограниченные апострофами. *Пример: ' Текстовая строка '*

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

В Турбо Паскале 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]-это 'р' и так далее.

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.

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.

Результат выполнения операций отношения над строками имеет логический тип, т.е. принимает значения True или False.

**‘папа’ > ‘мама’**

**‘Иванова’ > ‘Иванов’**

**‘мука’ <> ‘ мука’**

**При вводе переменных строкового типа рекомендуется каждую переменную вводить одним оператором *ReadLn (не Read); ;***

**Например, необходимо ввести имя, фамилию и отчество**

***Write (‘введи фамилию’); ReadLn(Fam);***

***Write (‘введи имя’); ReadLn(Imya);***

***Write (‘’введи отчество); ReadLn(Otch);***

**Стандартные функции обработки строк**

**Length(s:string): byte–** возвращает в качестве результата размер в символах строки S.

*Пример.* n := length('Pascal'); {n будет равно 6}

**Copy(st:string; Poz: integer; N: integer ):string**– выдает подстроку, выделенную из строки **S**t, длиной **N** символов начиная с позиции **Poz.**

*Пример.*

***S := 'I love you!' {длина = 11 }***

***SubS := Copy(S,3,4); { SubS = 'love' }***

***SubS := Copy(S,100,4); { SubS = '' }***

***SubS := Copy(S,3,100); { SubS = 'love you!' }***

**Pos(str1,str2:string):byte**– обнаруживает первое появление в строке **Str2** подстроки **Str1**. Результатом является номер символа в строке **Str2**, с которого начинается подстрока **Str**1. Если фрагмент в строке не найден, то функция возвращает нуль.

**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.

## Стандартные процедуры обработки строк

**Delete (st:string; Poz: integer; N: integer)** – удаление **N** символов из строки **St**, начиная с позиции **Poz**.

Так оператор **Delete(Words, 2, 3);** удаляет из строки **Words** фрагмент, состоящий из трех символов и начинающийся со второй позиции.

*Примеры:*

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

***……..***

***S := ‘Коля’;***

***Delete(S,1,1); { S = ‘оля’ }***

***Delete(S,100,3);***

***……..***

***Delete(‘Попробуй удали!’,3,8);***

***{ Ошибка! S не может быть константой }***

**Insert (Str1, Str2, Poz)** – вставка строки **Str1** в строку **Str2**, начиная с позиции **Poz**.

*Примеры:*

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

***……..***

***S := ‘Начало - - Конец’;***

***SubS := ‘ Середина’;***

***Insert(SubS, S, 9); { S = ‘Начало - Середина - Конец’ }***

**Str ( n [:<размер>:<точность>], St)** – преобразование числового значения n в строковое значение. Параметры “размер” и “точность” у числового значения имеют тот же смысл и задаются точно так же, как в операторе вывода данных. Результат преобразования данных помещается в строку St.

*Примеры:*

***Str(3.1415: 7 : 2, S ); { S = ‘_ _ _ 3.14’ }***

***Str(P:7,S); { P: Word = 4433; S = ‘_ _ _ 4433’ }***

**Val (St, n, Cod)** – преобразование числового значения из строкового представления St в значение целого или вещественного типа. Результат помещается в числовую переменную n. Cod - переменная целочисленного типа, в которую заносится признак завершения операции преобразования данных. В случае успешного завершения преобразования данных переменная Cod будет содержать значение 0, в противном случае в ней будет записан номер символа строки, не соответствующий формату строкового представления числовых данных.

*Пример:*

```
Write(‘Введите число: ‘);
ReadLn(S); { S: String }
val (S, x, ErrCode); { x: Word }
if ErrCode <> 0 then  WriteLn(‘Ошибка! Повторите ввод.’);
```

***Задача 1:*** Дана строка, состоящая из слов, разделенных между собой одним или несколькими пробелами. Найти длину самого длинного слова.

```
program zadacha1;
var stroka:string;
n,m,dlina:byte;
kv:integer;
begin
write('введи строку из слов, разделенных пробелами'); readln(stroka);
n: =0; m: = 0; dlina: =0;
repeat
n:=n+1;
if stroka[n]<>' ' then begin m:=m+1;
if m >=dlina then dlina:=m; end;
if stroka[n]=' ' then m:=0;
until n = length(stroka);
writeln(dlina);
readln;
end.
```

***Задача 2:*** Дана строка, преобразовать ее, заменив в ней каждую точку многоточием.

```
program zadacha2;
var s1,s2:string; n, m:byte;
begin
write('введи строку текста'); readln(s1);
S2 : = ''; {пустая строка}
for n : =1 to length(s1) do     begin
   if s1[n]='.' then begin s2:=s2+'…' else s2 : =s2+s1[n];
  end;
writeln(s2);
readln;
end.
```

***Задача 3:*** Подсчитать в строке количество знаков препинания и количество пробелов

```
program zadacha3;
var s :string; c: char;
 n,m,l:integer;
Begin
 write('введи строку'); readln(s);
n: =0; m: = 0; l : = 0;
for n:=1 to length(s) do begin
  c : = s[n];
  if (c='!') or (c=':') or (c=';') or (c=',') or  (c='.') or (c='?') or (c='-') or (s[n]='...') then
          m:=m+1;
 if c =' ' then l:=l+1; end;
writeln('количество знаков препинания = ',m);
writeln(' количество пробелов = ',l);
readln;
end.
```

***Задача 4:*** Определить, является ли строка "Палиндромом". Если нет, то выяснить, станет ли она "палиндромом" после удаления из нее всех пробелов

```
program zadacha4;
var s,s1,s2:string;
n,m:byte;
 begin
write('введи строку'); readln(s);
s1: =''; {пустая строка}
for n := length(s) downto 1 do s1:=s1+s[n];
if s1=s then writeln('Palindrom') else
begin
s1:=''; m : = 0;
for n:=1 to length(s) do begin s1:=s1+s[n];
  if s[n]=' ' then begin delete(s1,n-m,1); m:=m+1; end;
end;
s2:='';
for n := length(s1) downto 1 do s2:=s2 + s1[n];
 if s2=s1 then writeln('Stala polindromom posle udaleniya probelov');
end;
readln;
end.
```

# Вопросы?