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

Информатика и программирование

Лебедева Т.Ф.

Рассмотрим примеры использования оператора цикла **for.**

Пр.1 Пусть требуется вывести на экран таблицу квадратов N целых чисел:

```
        Var   i, n : integer;
Begin
readln( n );
for i : = 1 to n do  writeln (I : 4,  i *  i : 8) ;
```

Пр.2 Подсчитать сумму 20 вводимых чисел:

```
S: = 0;
for i : = 1 to 20 do
        begin  readln(x);  S: = S + x; end; write (S:6:2);
```

В некоторых случаях бывает удобно, чтобы <параметр цикла> принимал последовательные, но не возрастающие, а убывающие значения. В этом случае надо использовать оператор цикла с параметром следующего вида:

for <параметр цикла> := <N> downto <K> do
<оператор>;

Пр.3 Пусть требуется вывести на экран последовательность букв от M до A и их номера:

```
        Var Ch : char;
begin
for Ch := 'M' downto 'A' do
        writeln (Ch:2,  ' номер-',    ord( ch) :4);
```

Рекомендации по программированию цикла for :

1) Не следует внутри цикла изменять параметр цикла;
2) Не следует внутри цикла изменять начальное и конечное значения параметра цикла с помощью операторов присваивания или ввода;
3) Значение параметра цикла не определено после выхода из цикла;
4) Нельзя перейти к оператору тела цикла по оператору goto <метка>, минуя заголовок цикла;
5) Допускается досрочный выход из цикла с использованием процедур:

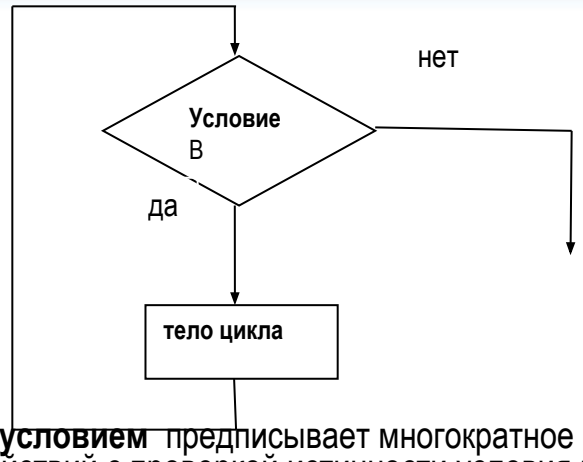
Break - реализует выход из цикла любого типа;

Continue - осуществляет переход на следующую итерацию цикла, игнорируя оставшиеся до конца тела цикла операторы;

Halt (<код, завершения>) - осуществляет выход из программы, возвращая операционной системе заданный код завершения. Считается, что программа завершилась нормально, если код завершения равен нулю. Возвращение кода завершения, отличного от нуля, обычно означает, что программа завершена по обнаружении каких-либо ошибок. Коды завершения назначаются программистом, а информация о них помещается в программную документацию;

Exit- осуществляет выход из подпрограммы. Если процедура использована в основной программе, то выполняется выход из программы

Цикл с предусловием




```
while  B Do
Begin
тело цикла
End;
```

Цикл с предусловием предписывает многократное выполнение одной и той же последовательности действий с проверкой истинности условия перед телом цикла.

Выполнение оператора заключается в многократном повторении двух действий:

- вычислении логического выражения B (условия), записанного в заголовке цикла;
- выполнении оператора, являющегося телом цикла.

Эти действия повторяются, пока выражение в заголовке не станет ложным. Проверка истинности выражения производится до первого выполнения оператора, то есть, если выражение сразу ложно, то тело цикла не выполнится ни разу.

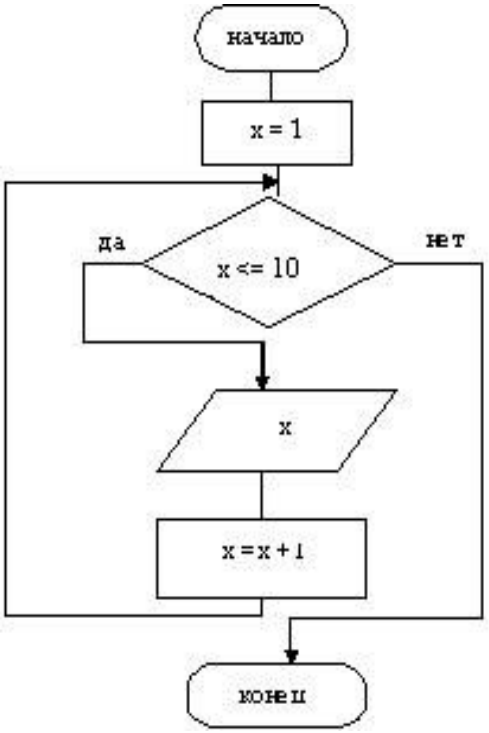
В качестве примера рассмотрим фрагмент программы, который выводит на экран натуральные числа от 1 до некоторого N.

```
Write ('Введите число > ');
        readln (n);
        i := 0;
        while (i <= n) do
          begin
              i := i + 1;
              writeln (i)
         end;
```

Пример5: Вывести на экран в одну строку через один пробел десять значений целого типа, начиная с единицы, в возрастающем порядке.
Алгоритм решения представлен на рисунке.

Текст программы на Паскале:

```
Const k = 10; Var x: integer;
begin x : = 1;
while x <= k do
  begin
    write(x,' '); x: = x + 1;
  end;
readln;
end.
```



Цикл с постусловием

Оператор цикла **repeat** в Паскале также используется в циклах с неизвестным числом повторений и с известным числом повторений. Вид оператора:

repeat s until b;

Здесь s – тело цикла,

b - логическое выражение.

При выполнении оператора

сначала выполняется тело цикла,

а затем проверяется логическое условие.

Таким образом, обеспечивается,

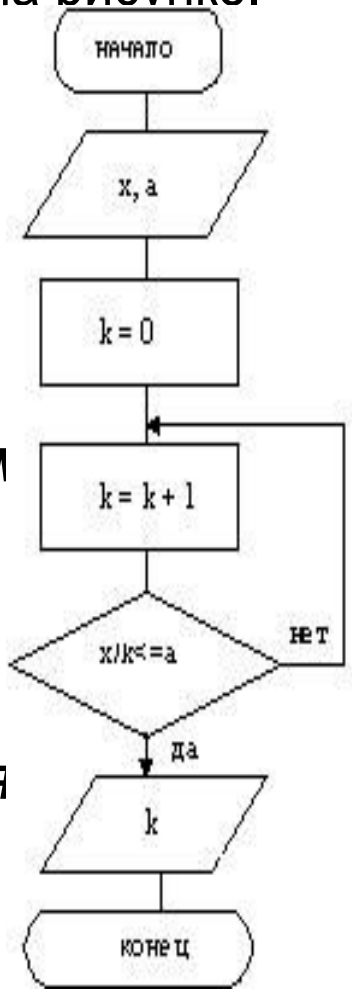
по меньшей мере, одно выполнение тела цикла.

Цикл повторяется, пока логическое выражение **ложно**. Когда оно становится истинным, происходит выход из цикла.

Пример 6: Составить программу определения k, при котором x/k становится меньше или равен a, где k = 1, 2, 3… . Алгоритм решения представлен на рисунке.

Текст программы на Паскале:

```
 Var x, a : real; k : integer;
begin
  write ('x='); readln (x); {ввод значения x}
  write ('a='); readln (a); {ввод значения a}
  k : = 0; {начальное значение k перед циклом
   repeat {начало цикла}
     k : =k+1; {увеличение k в цикле}
  until x/k <= a; {проверка логического условия}
  writeln('k=',k); {вывод найденного значения
        экран}
  readln;
end.
```



Выполним построение математической модели и алгоритма решения задачи табулирования функции.

Пример 7. Получить таблицу значений функции **y = f (x)** для аргумента **x**, изменяющегося от **a** до **b** с шагом **h** (если **a>b,** то **h** должен быть меньше нуля, т.е. отрицательным).

а) Обозначение переменных:

x – аргумент функции; y – значение функции;

a – начальное значение интервала изменения аргумента;

b – конечное значение интервала изменения аргумента;

h – шаг изменения аргумента на интервале;

i – счётчик цикла; **n** – число повторений цикла.

б) Тип переменных:

i, n – простые переменные целого типа;

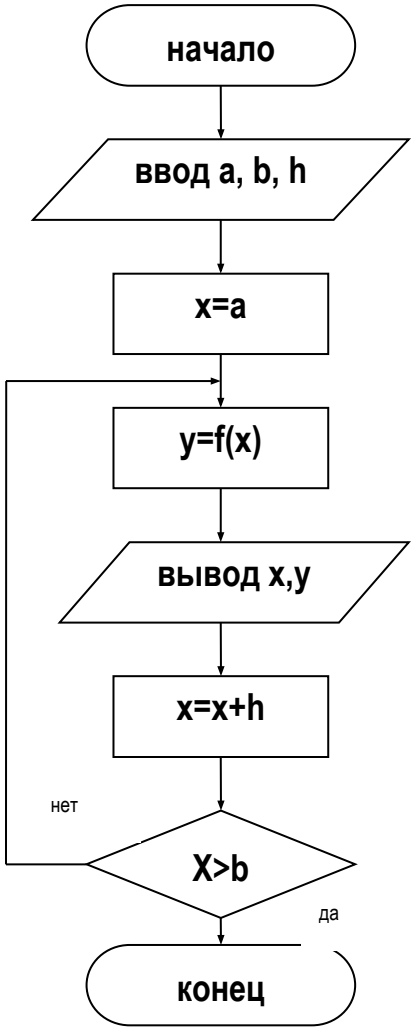
x, y, a, b, h – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **a, b, h;**

промежуточные результаты**: i, n**; результаты: **x, y**.

Блок-схема цикла с постусловием к примеру 7



Присваивание начального значения параметру цикла

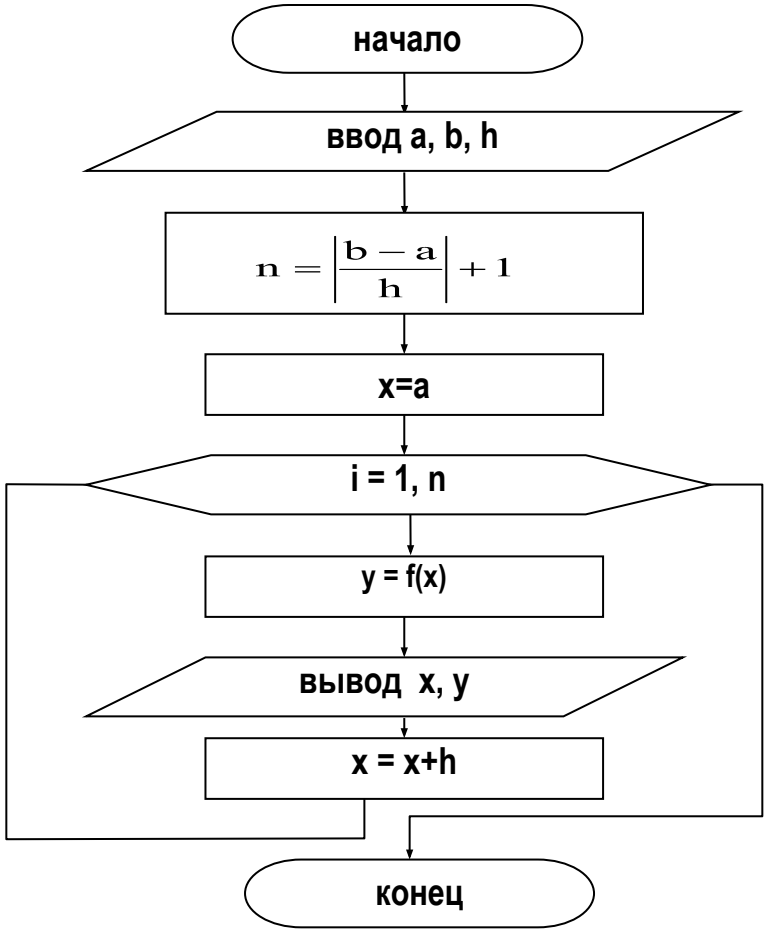
Рабочий участок или тело цикла

Изменение параметра цикла

Проверка условия на выход из цикла

```
Program primer_7_1;
Uses CRT;
Var x, y, a, b, h: real;
begin
write (‘a=’); readln (a); {ввод значения a}
write (‘b=’); readln (b); {ввод значения b}
write (‘h=’); readln (h); {ввод значения h}
Clrscr; {очистка экрана}
writeln( ‘                Таблица’);
writeln( ‘     x                                  y’); {вывод заголовка таблицы}
writeln(‘-------------------------------------------------’); {вывод горизонтальной черты}
x : = a; {начальное значение x перед циклом}
 repeat {начало цикла}
    y : =x * sin( x ); {вычисление функции в цикле}
    Writeln( x :8:3,   y:14:3); {вывод строки таблицы на экран}
    x:= x +h; {изменение параметра x}
until x > b; {проверка логического условия}
writeln(‘-------------------------------------------------’); {вывод горизонтальной черты}
readln;
end.
```

Блок-схема цикла со счетчиком i к примеру 7



```
Program primer_7_2;
Uses CRT;
Var x, y, a, b, h: real;
i, n : integer;
begin
write (‘a =’); readln (a); {ввод значения a}
write (‘b =’); readln (b); {ввод значения b}
write (‘h =’); readln (h); {ввод значения h}
Clrscr; {очистка экрана}
writeln( ‘                Таблица’);
writeln( ‘     x                                  y’); {вывод заголовка таблицы}
writeln(‘-----------------------------------------------------’); {вывод горизонтальной черты}
x : = a; {начальное значение x перед циклом}
N : = trunc( (b –a) / h ) + 1; {вычисление количества строк таблицы (повторений цикла)}
 for i : = 1 to n do         {начало цикла}
   begin
     y : =x * sin( x ); {вычисление функции в цикле}
     Writeln( x :8:3,   y:14:3); {вывод строки таблицы на экран}
     x:= x +h; {изменение параметра x}
   end;
writeln(‘-----------------------------------------------------’); {вывод горизонтальной черты}
readln;
end.
```

Пример 8. Первоначальная стоимость оборудования производственного цеха составляет R0 руб. Ежегодно на сумму D руб. закупали новое оборудование. Ежегодная амортизация (уменьшение стоимости) имеющегося оборудования составляет P% от его стоимости. Составить алгоритм для вычисления стоимости оборудования цеха RN через N лет после ввода его в эксплуатацию согласно формуле ,

$$R_N = R_{N-1}(1 - \frac{P}{100}) + D$$

где **RN** - стоимость оборудования в **N**-й год; где **RN-1** - стоимость оборудования в предыдущем году. Получить таблицу результатов для N=1, 2,…10.

а) Обозначение переменных:

R0 – первоначальная стоимость оборудования;

D – стоимость ежегодно закупаемого нового оборудования;

P – процент ежегодной амортизации оборудования;

N – счётчик цикла, количество лет эксплуатации оборудования.

б) Тип переменных:

P, N – простые переменные целого типа;

R0, D – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **R0, D, P**; результаты: **N, R0**.

г) расчетные формулы в последовательности их выполнения:

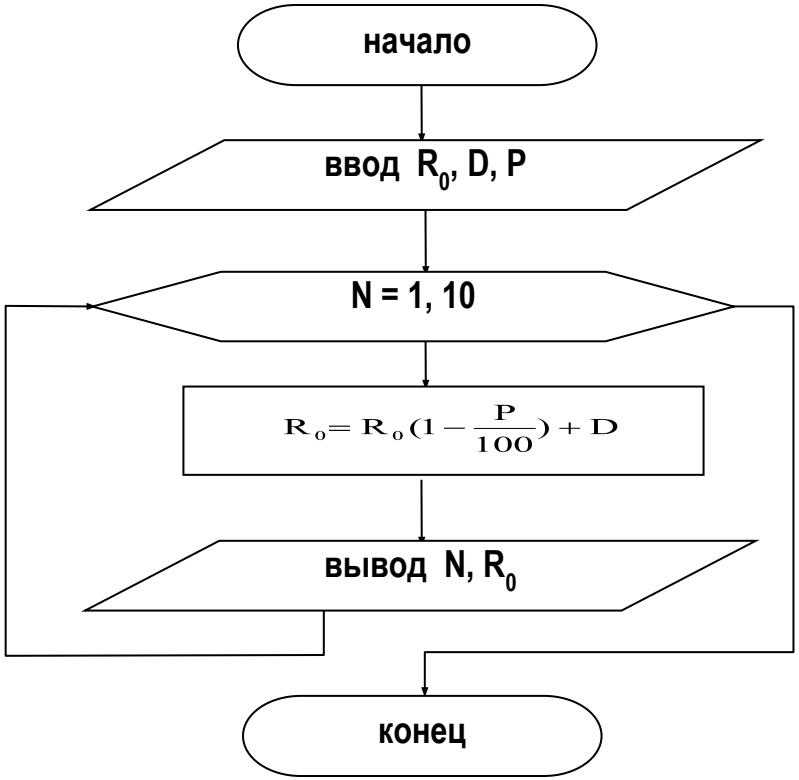
N=1

$R_N = R_0(1 - \frac{P}{100}) + D$

N=N+1

Если N≤10, то повторять действия, иначе выход из цикла

Блок-схема цикла со счетчиком N к примеру 8



Пример 9

Разработать программу, которая определяет первый отрицательный элемент последовательности значений функции sin x при заданных шаге h и диапазоне изменения x [a, Ь].

Вариант с использованием п р о ц е д у р ы break:

```
Program primer_9;
var i, n : integer;   x, y, a, b, h : real;
Begin
Write (‘Введите a, b, h:');
Readln(a, b, h);
n:=round((b-a)/ h + 1.5); {определяем количество элементов}
x := a;
for i : = 1 to n do
   begin
     y: =sin(x);
     If y < 0 then
       begin
         Writeln( ‘y= ', y:8:6, ' при x= ‘,  x:6:3 );
          break; {осуществляем досрочный выход из цикла}
       end;
   end;
{место, куда будет передано управление при выполнении break}
If y> 0 then WriteLn ('Элемент не найден,');
Readln;
End.
```

Для операторов ***repeat*** и ***while*** значения параметра цикла (переменной, входящей в условие) должны изменяться в теле цикла, иначе цикл никогда не завершится (ситуация зацикливания).

Пр.4 ***s : = 0; x: = 0;***

While x < 10 do begin s : = s + x; x : = x – 1; end;

Пр.5 Найти произведение четных чисел, меньших n:

Readln(n);

p: = 1; k : = 2;

Repeat

p: = p * k;

k : = k + 2;

Until k > n;

Циклы с операторами ***repeat*** и ***while*** часто заменяют в Паскале конструкцию

If <условие> then goto <metka> , особенно, если переход по метке должен осуществляться к операторам, расположенным выше. Такие конструкции в Паскале крайне нежелательны.

Пр.6 Реализовать фрагмент контроля правильности ввода исходных данных.

```
Repeat
  Write( 'введите a, b');
   Readln (a, b); Writeln( 'a=', a :6:2, 'b=', b :6:2);
   Readln ('данные введены верно? (1/0' );
   Readln (c);
Until c = 1;
……..
```

К циклам с неизвестным числом повторений относятся, в частности, итерационные циклы, в которых происходит последовательное приближение к результату с заданной точностью.

Пример 9.

Вычислить сумму бесконечного ряда

$$x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - \ldots\ldots\ldots\ldots + x^i/i - \ldots\ldots\ldots$$

с заданной точностью ε $|x| < 1$. Вычисление суммы заканчивается, когда очередной член ряда становится по модулю меньше точности.

Математическая постановка

а) Обозначение переменных:

x – аргумент; **s** – значение суммы;

e – значение точности (погрешности);

m – член ряда;

p – промежуточная переменная;

i – номер члена ряда;

б) Тип переменных:

i – простая переменная целого типа;

x, s, e, m, p – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **x, e**;

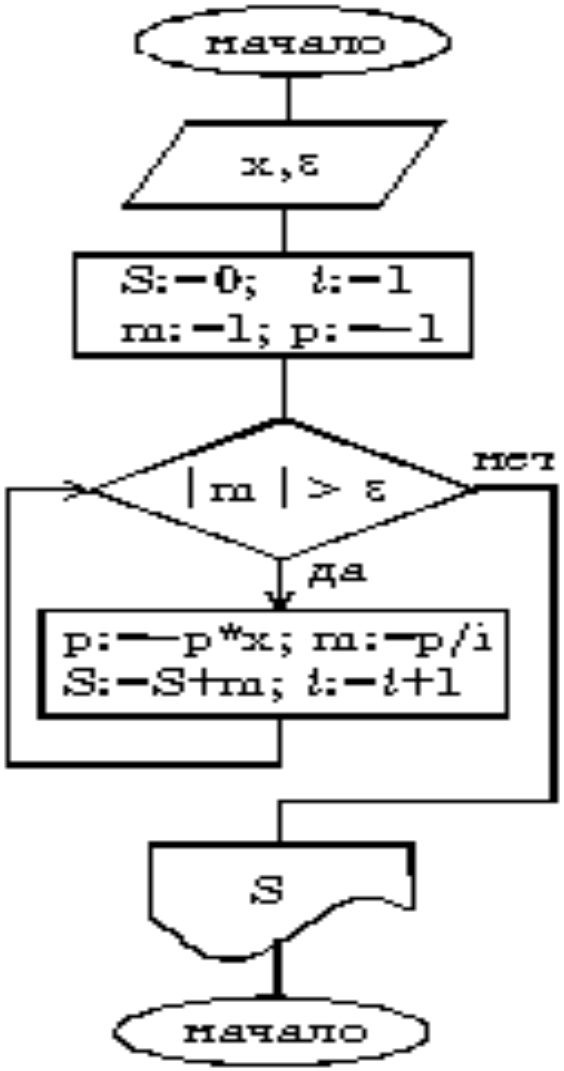
промежуточные результаты: **i, p, m**;

результат: **s**.

г) запись расчетных формул:

```
S = 0
m  = 1
p = -1
i  = 1
если abs (m) >  e выполнить действия
    p = - p * x
    m  = p / i
    s  = s + m
     i = i + 1
```

блок-схема алгоритма примера 9



```
Program primer_9;
Uses CRT;
Var x, s, e, m, p: real;
i : integer;
begin
Clrscr; {очистка экрана}
write (‘x=’); readln (x); {ввод значения x}
write (‘e=’); readln (e); {ввод значения точности}
S : = 0;
m : = 1;
p := -1;
i : = 1; {начальные  значения перед циклом}
 while  abs (m) >  e do    {начало цикла}
    begin
      p := - p * x;  m : = p / i  ; {  вычисление члена ряда}
     s : = s + m;
     i : = i + 1;
    end;
Writeln ( ‘s = ‘, s : 14 :8); {вывод суммы на экран}
Writeln ( ‘количество членов ряда = ’,  i );
readln;
end.
```

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 и 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*

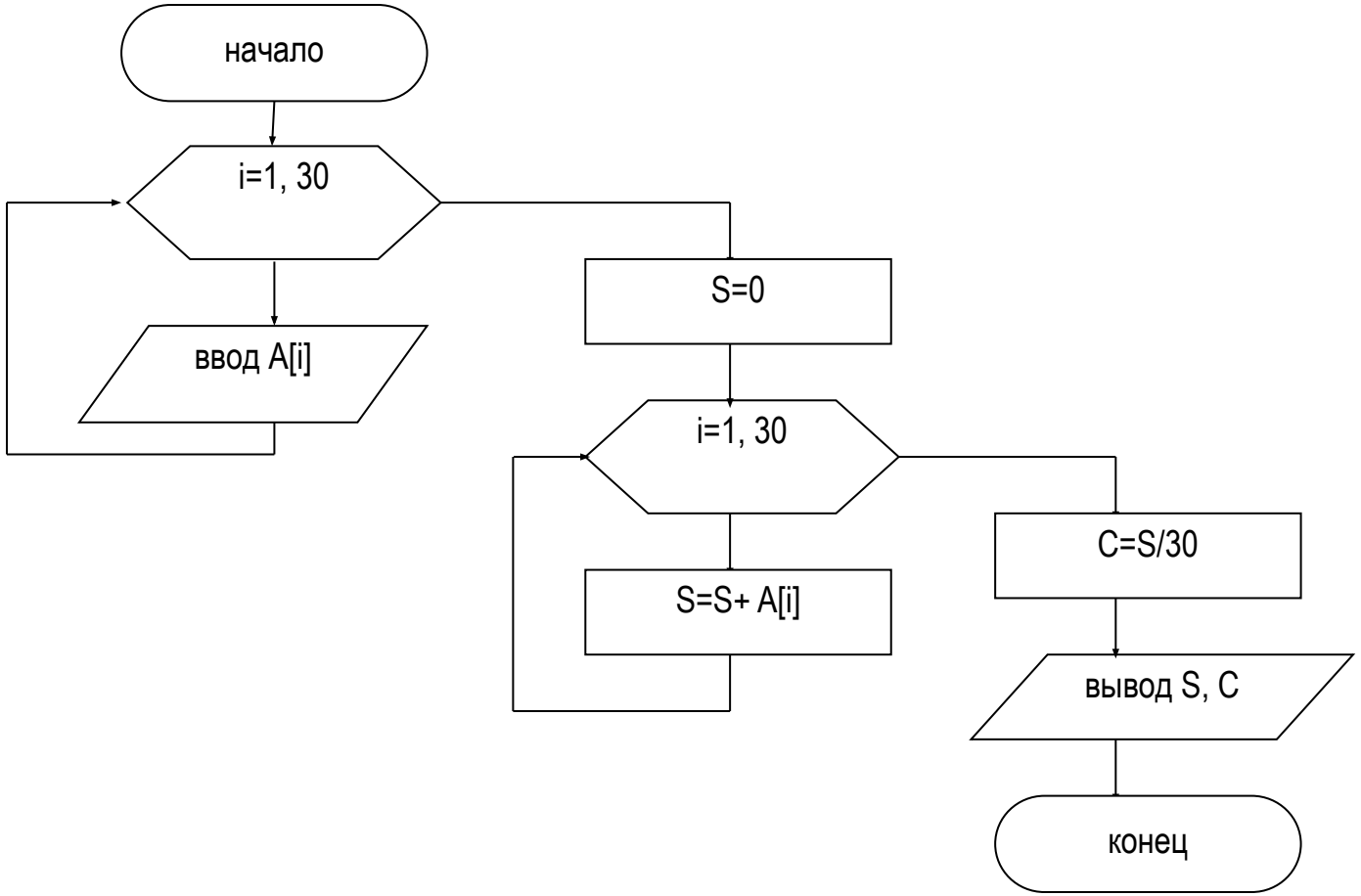
S = S + A[i] накопление суммы в цикле

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**

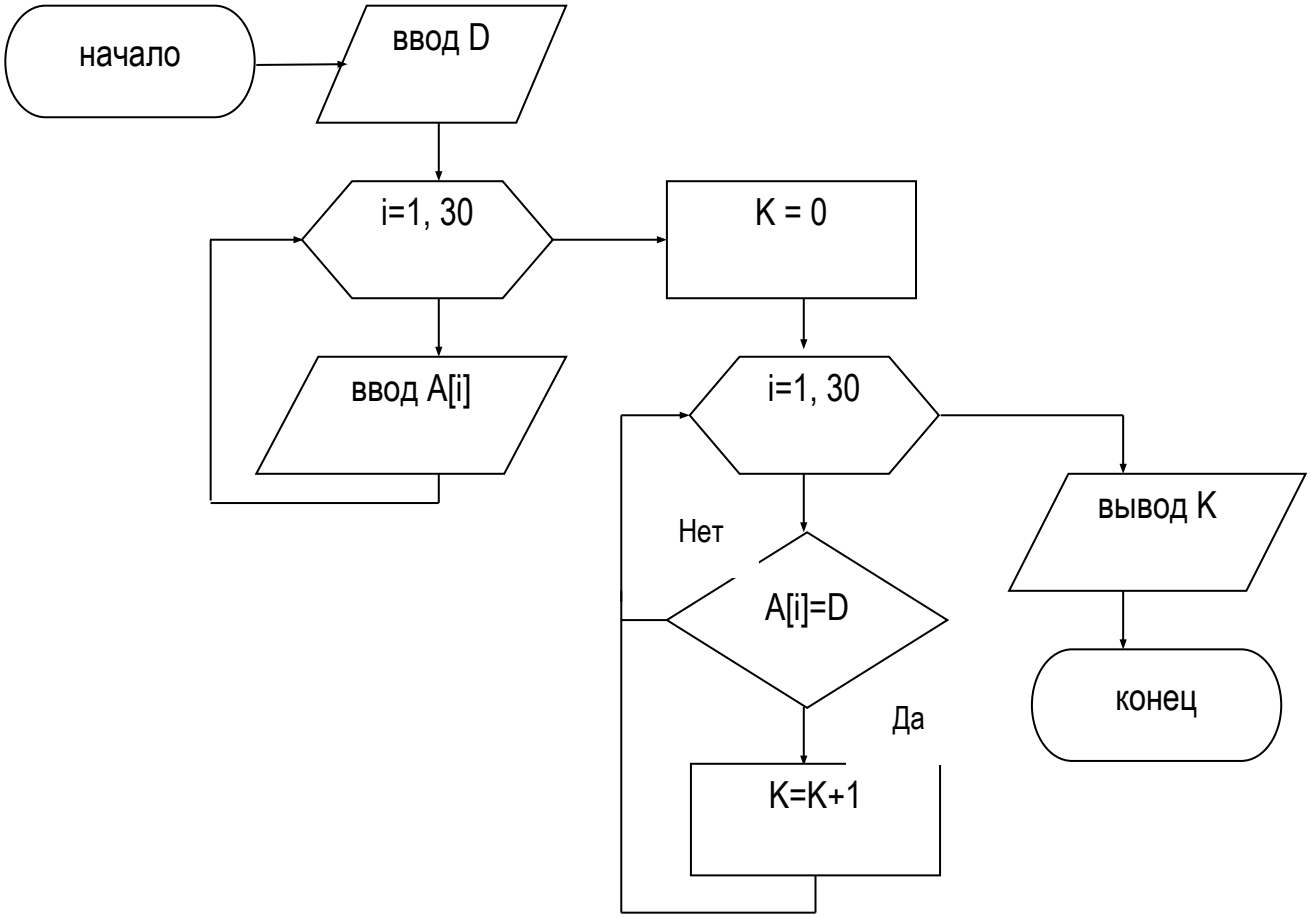
i = 1 начальный номер элемента

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

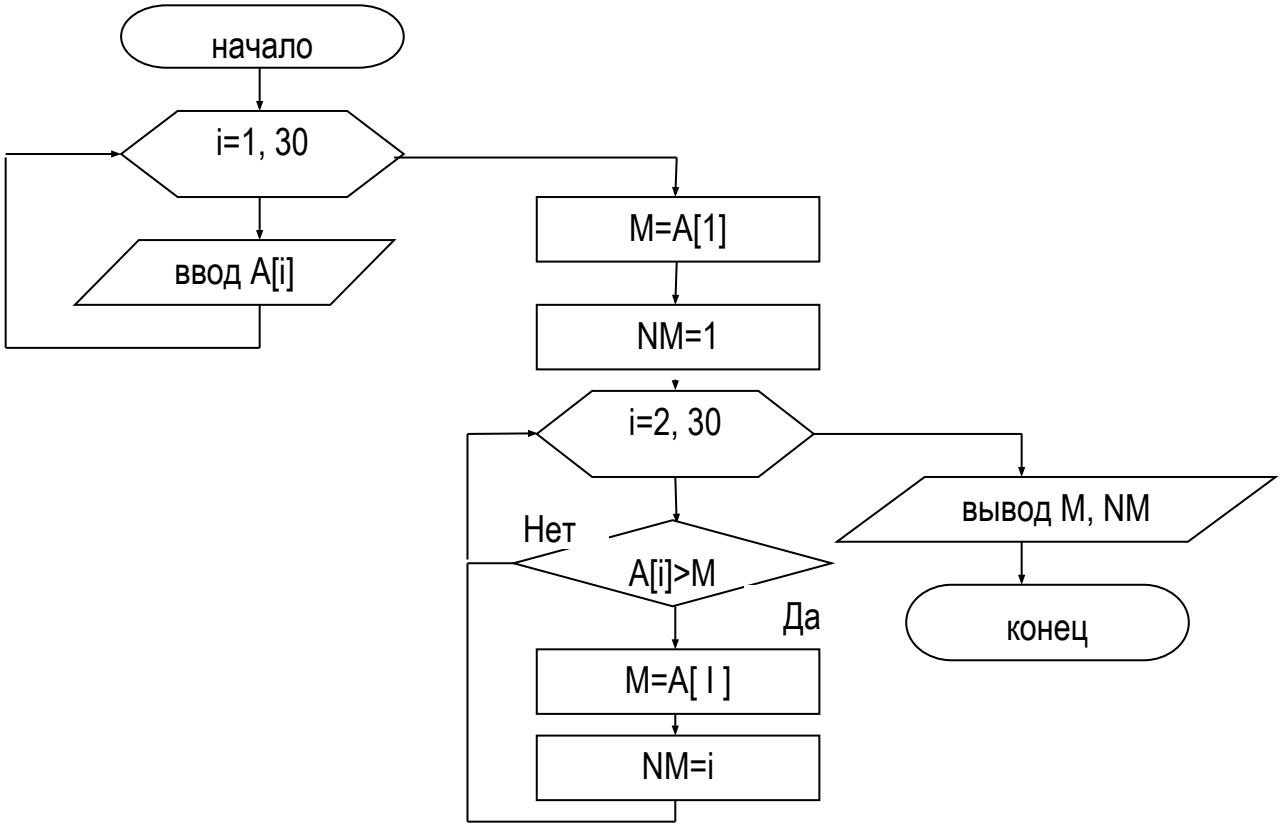
Если i ≤ 30, ***то*** повторять действия, ***иначе*** выход из цикла

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

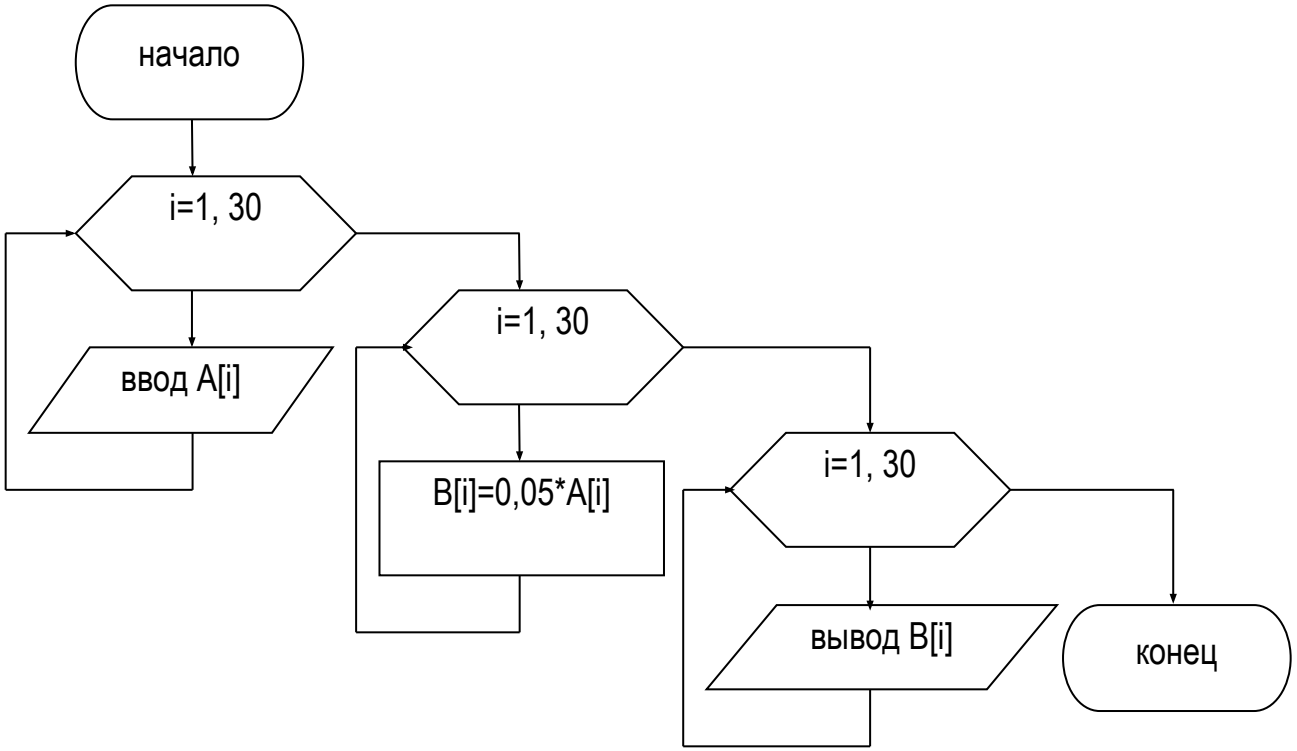
Примечания: Алгоритмы формирования нового массива могут быть следующих видов:

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 из n целых чисел A . 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, NM *– 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 элементов 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-до или циклы-пока, в которых условие выхода формируется из двух условий

первое условие - пока искомый элемент не найден,

а второе 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 п - 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 количество элементов n <= 100'); Readln(n);
Writeln('Введите искомое значение '); Readln(s);
Writeln('введите ', n,  ' элементов массива ');
for i: =1 to n do Read( a[ i ]);
Readln;
i: = 1; {начальное значение индекса массива}
Key : = false;
while not  key   and  ( i <= n ) do begin
key : = (s = a [ l ]);     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 этой последовательности (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 отмеченном куске последовательности, чтобы сравнить ее с 6. Однако здесь нас поджидает небольшая проблема: точной середины у новой последовательности 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" последовательности 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 последовательности:

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, l : 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
L := (n – k )  div  2 + к;    {определяем среднее значение индекса}
If   s = a [ L]    then   key := true   {запись найдена}
else {уменьшаем диапазон индексов}
 if    s  > a [ l ]  then  k : =  L + 1 {смещаем левую границу}
else n: = L - 1 {смещаем правую границу}
end;
if  key  then   WriteLn(''элемент найден.  Номер равен ', l ) 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 файлами).

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 каждой группы. При этом будем использовать 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

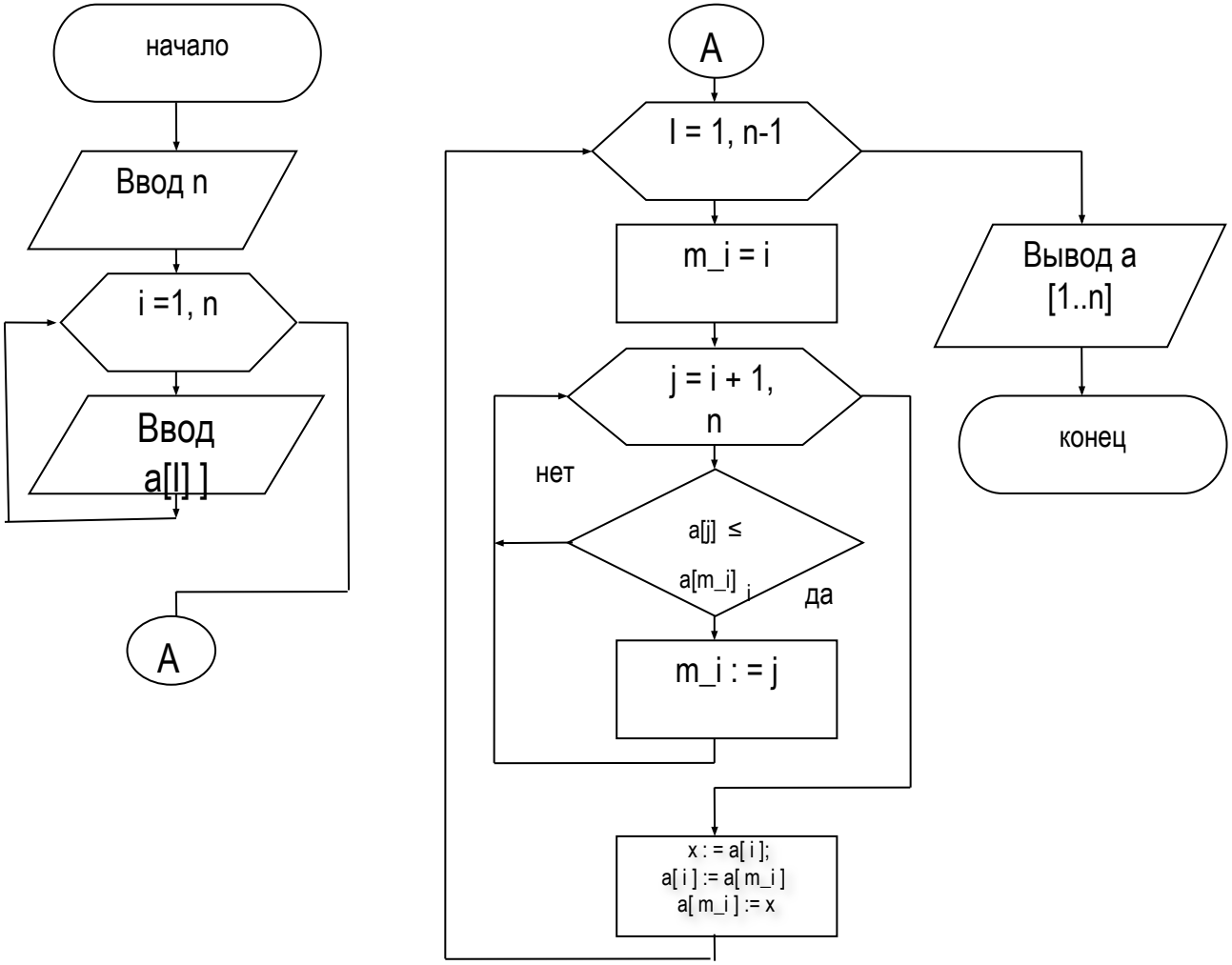
Предположим, что нужно отсортировать набор чисел:

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

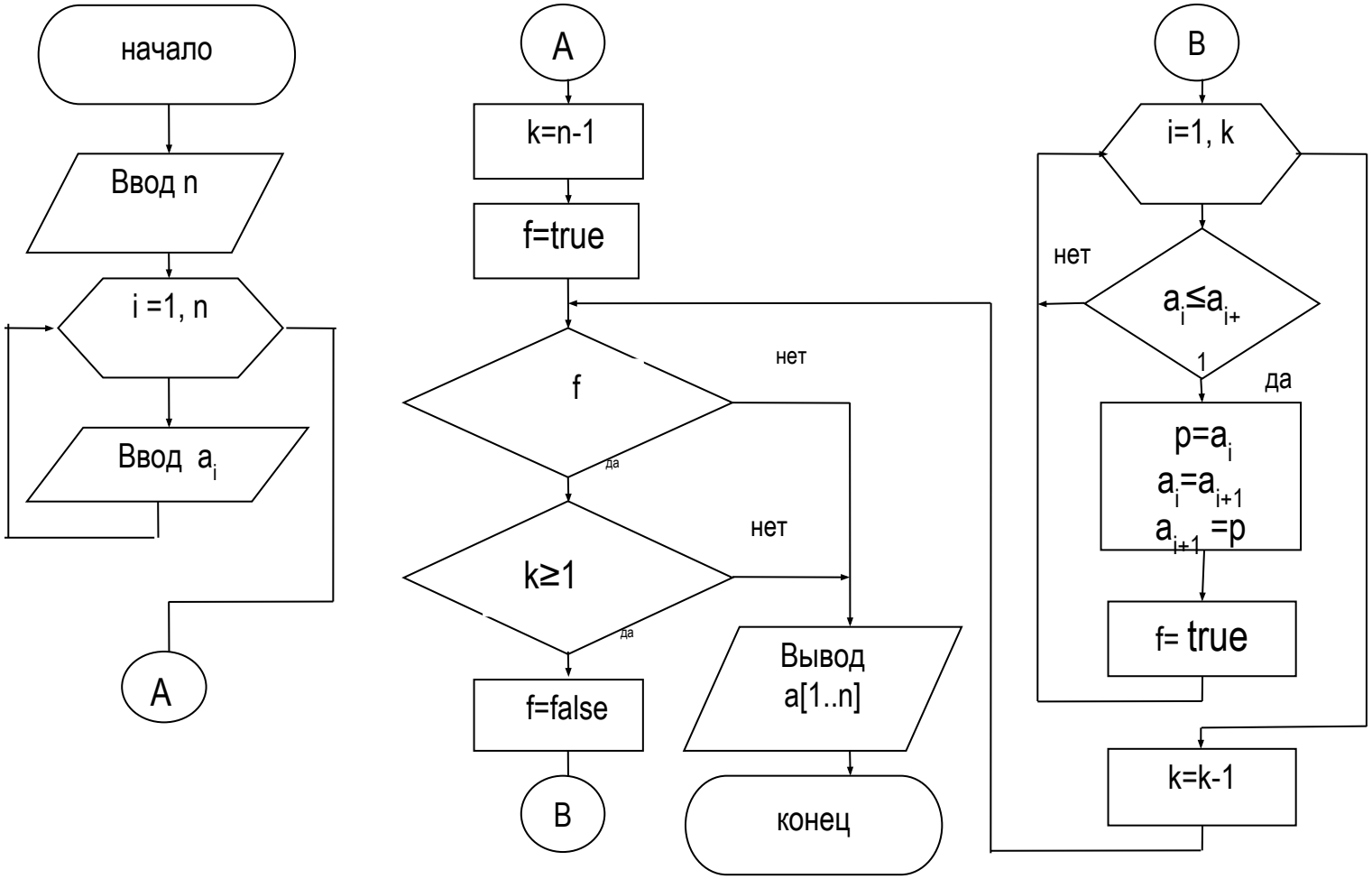
4 шаг: 3 3 2 1 4 5 6 → 3 2 3 1 4 5 6 → 3 2 1 3 4 5 6

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;
Writeln ( ‘отсортированный массив’);
for i : = 1 to n do
Write( ‘   a [ ‘ , i,  ‘ ] =‘ , ); a [ i ] );  Writeln
readln;
end.
```

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

.....

```
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;
```

……

Вопросы?