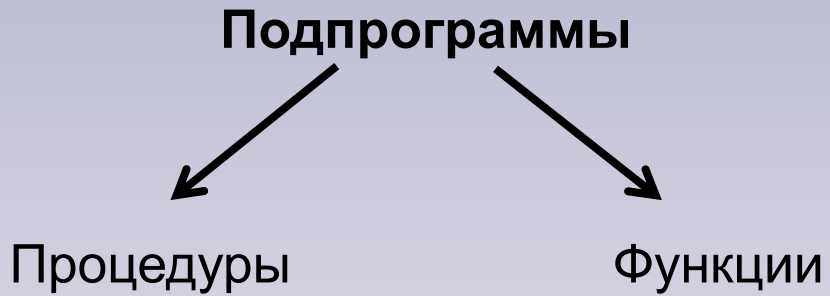
Вспомогательные алгоритмы. Подпрограммы

Подпрограммы – это обособленные от основной программы операторы, которые можно многократно вызывать из различных участков кода.



Процедура – независимая именованная часть программы. После однократного описания может использоваться бесконечное число раз. Вызов производится по имени из последующих блоков кода. Нужна для выполнения тех или иных действий.

Функция - готовый блок кода специального вида. Всегда возвращает тот или иной результат. Ее вызов в программировании является выражением. Он применяется в других операциях. Примеры – при присваивании в правой части.

```
    procedure имя(список формальных параметров);
раздел описаний
begin
      операторы
end;
```

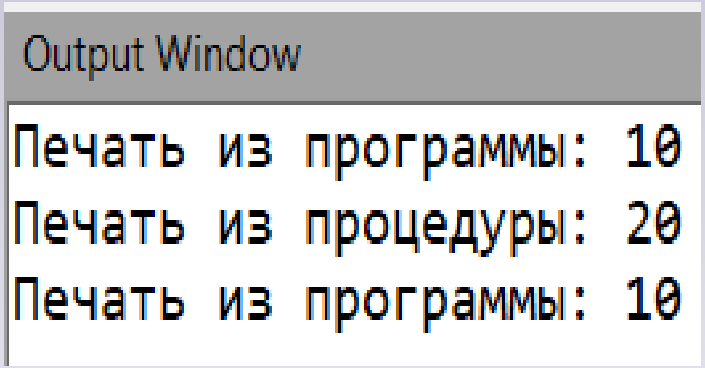
```
    function имя(список формальных параметров): тип
возвращаемого значения;
раздел описаний
begin
  операторы
end;
```

имя(список фактических параметров);

В стандарте языка Паскаль передача параметров может производиться двумя способами - **по значению** и **по ссылке**. Параметры, передаваемые по значению, называют **параметрами-значениями**, передаваемые по ссылке - **параметрами-переменными**.

При передаче параметров по **значению** подпрограмма получает **копии** параметров, и операторы подпрограммы работают с этими копиями. **Доступа к исходным значениям параметров у подпрограмм нет**, а, следовательно, **нет и возможности их изменить**.

```
var a: integer;
procedure Proc(x:integer);
begin
  x:=x+10;
  writeln('Печать из процедуры: ', x);
end;
begin
a:=10;
writeln('Печать из программы: ', a);
Proc(a);
writeln('Печать из программы: ', a);
end.
```



В данном примере значение формального параметра x было изменено в Proc, но эти изменения **не отразились** на фактическом параметре a из основной программы.

При передаче параметров **по ссылке** подпрограмма получает копии адресов параметров, что позволяет осуществлять доступ к ячейкам памяти по этим адресам и изменять исходные значения параметров. Для того, чтобы параметр передавался по ссылке необходимо при описании метода перед формальным параметром написать ключевое слово **var**.

```
var a,b: integer;
procedure Proc(x:integer; var y: integer);
begin
  x:=x+10;
  y:=y+10;
  writeln('Печать из процедуры: x = ', x,' y = ', y);
end;
begin
a:=10; b:=10;
writeln('Печать из программы: a= ', a, ' b= ',b);
Proc(a,b);
writeln('Печать из программы: a= ', a, ' b= ',b);
end.
```

```
Печать из программы: a= 10 b= 10
Печать из процедуры: x = 20 y = 20
Печать из программы: a= 10 b= 20
```

В данном примере в процедуре были изменены значения формальных параметров x и y. Эти изменения не отразились на фактическом параметре a, так как он передавался по значению, но значение b было изменено, так как он передавался по ссылке.

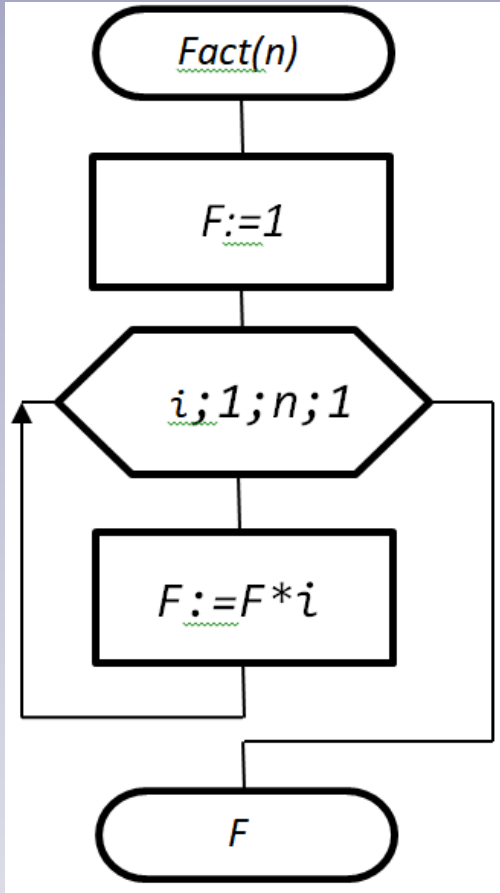
При обращении фактическими параметрами, подставляемыми на место формальных параметров-значений, могут быть выражения, а фактическими параметрами, подставляемыми на место формальных параметров-адресов, могут быть только переменные.

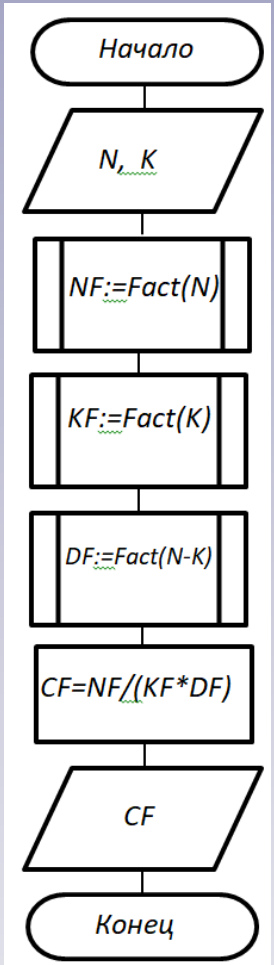
Функция отличается от процедуры тем, что результат её работы возвращается в виде значения этой функции, и, следовательно, вызов функции может использоваться наряду с другими операндами в выражениях.

Пример Функция находит площадь и периметр треугольника, заданного длинами сторон.

```
function Triangle(a,b,c:real; var p,s:real):integer;
var pp: real;
    begin
 if((a+b<=c) or (a+c<=b)
                        or (b+c<=a)) then result:=0
 else   begin
           p:=a+b+c;
           pp:=0.5*p;
           s:=sqrt(pp*(pp-a)*(pp-b)*(pp-c));
           result:=1;
end;end;
```

```
var a,b,c: real; //длины сторон - фактические параметры
Perim, Square:real; // периметр и площадь - фактические //параметры
begin
readln(a,b,c);
if(Triangle(a,b,c,Perim,Square)=1) then writeln('Периметр = ',Perim,'
Площадь = ', Square)
     else writeln('Треугольник не существует');
end.
```


Fact(n)
F:=1
i;1;n;1
F:=F*i
F


Начало
N, K
NF:=Fact(N)
KF:=Fact(K)
DF:=Fact(N-K)
CF=NF/(KF*DF)
CF
Конец

Структура данных – множество элементов данных и множество связей между ними

Структура данных – программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных с помощью вычислительной техники.

Проще говоря, структура данных – это контейнер, в котором хранятся различные данные (зарплата сотрудника, цены на акции, список покупок и т. д.) в определенной компоновке (формате, или способе организации их в памяти). Эта «компоновка» позволяет структуре данных быть эффективной в одних операциях и неэффективной в других.

Исходя из разных сценариев, данные должны храниться в определенном формате. **Цель** – выбрать структуру данных наиболее оптимальную для рассматриваемой проблемы.


Структуры данных
Простые базовые структуры
Статические структуры
Полустатические структуры
Динамические структуры
Файловые структуры (файлы)
Числовые
Символьные
Логические
Перечисление
Интервал
Указатели
Вектор
Массивы
Множества
Записи
Таблицы
Стеки
Очереди
Деки
Строки
Последовательные
Прямого доступа
Комбинированного доступа
Организованные разделами
Линейные связные списки
Разветвлённые связные списки
Графы
Деревья

Массив – это упорядоченный набор однотипных значений (элементов массива), хранящихся в смежных ячейках памяти. Каждый массив имеет имя, что дает возможность различать массивы между собой и обращаться к ним по имени.

Массивы могут быть одномерными и многомерными

Одномерные массивы

Каждый элемент массива имеет три характеристики:

1) **имя**, совпадающее с именем массива;
2) **индекс** – целое число, равное порядковому номеру элемента в массиве;
3) **значение** – фактическое значение элемента, определенное его типом.

Шкаф – это массив

Ящики – это индексы

Содержимое ящиков – элементы массива

```
Var имя_массива: array [начальный_индекс..конечный_индекс] of
тип_элементов;

var a:array[1..10] of integer;

type тип_массива = array [начальный_индекс ..
конечный_индекс] of тип_элементов;

var имя_массива: тип_массива;
```

индекс	1	2	3	4	5	6	7	8	9	10
значение	3	0	15	4	6	-2	11	0	-9	7

```
a[1] = 3;  a[5]=6;  a[7]=11;  a[9]=-9;  a[10]=7
```

```
var m: array[-5..5] of real;
var z: array[‘a’..’z’] of byte;
```

```
const n:integer=5;
var a:array[1..n] of integer;
    i: integer;
begin
//Введем массив
for i:=1 to n do
begin
 write('Введите a[',i,']','  ');
 read(a[i]);
end;
writeln('Вывод');
for i:=1 to n do
write(a[i],' ');
end.
```

Output Window

```
Введите a[1]   4
Введите a[2]   6
Введите a[3]   7
Введите a[4]   8
Введите a[5]   9
Вывод
4 6 7 8 9
```

Передача статического массива в подпрограмму

```
const n=5;
type arr=array[1..n] of integer;
```

```
procedure p(a: Arr);
...
p(a1);
```

При передаче статического массива в подпрограмму по значению также производится копирование содержимого массива - фактического параметра в массив - формальный параметр. Передавать так - ПЛОХО!

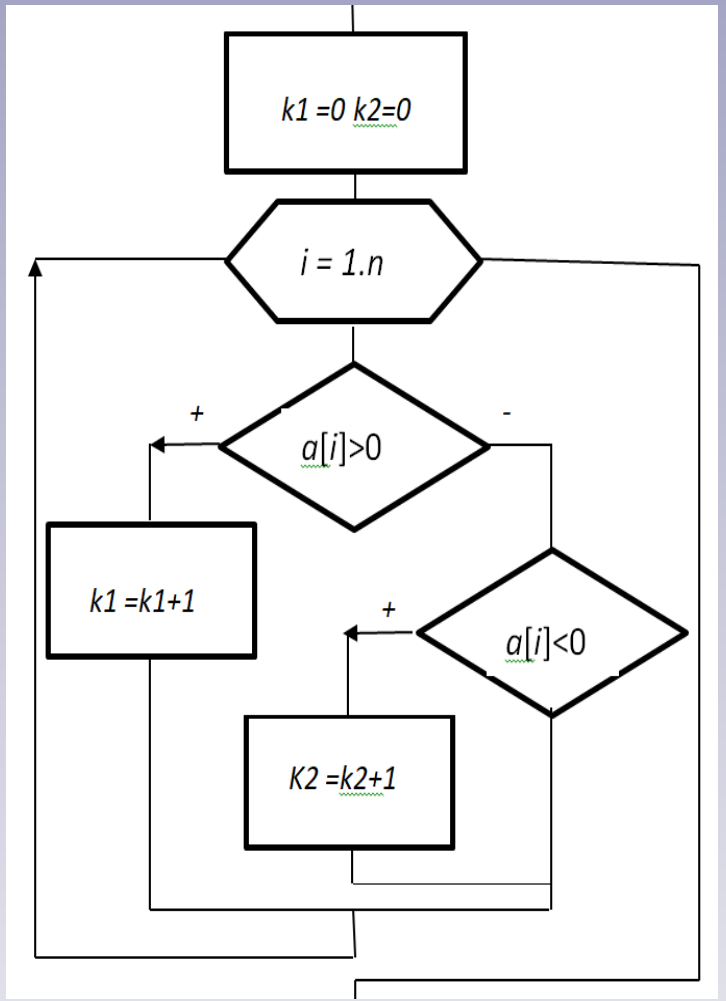
Статические массивы рекомендуется передавать по ссылке. Если массив не меняется внутри подпрограммы, то его следует передавать как ссылку на константу, если меняется - как ссылку на переменную.

```
const n=5;
type arr=array[1..n] of integer;
procedure ran_gen(var a:arr);
begin
  for var i:=1 to n do a[i]:=random(10);
end;
procedure print_arr(const a:arr);
begin
  for var i:=1 to n do write(a[i],' ');
end;
var b:arr;
begin
ran_gen(b);
print_arr(b);
end.
```

Output Window

```
4 1 6 0 2
```

Найти количество положительных и отрицательных чисел в данном массиве.



Найти наибольший элемент массива и определить его номер.

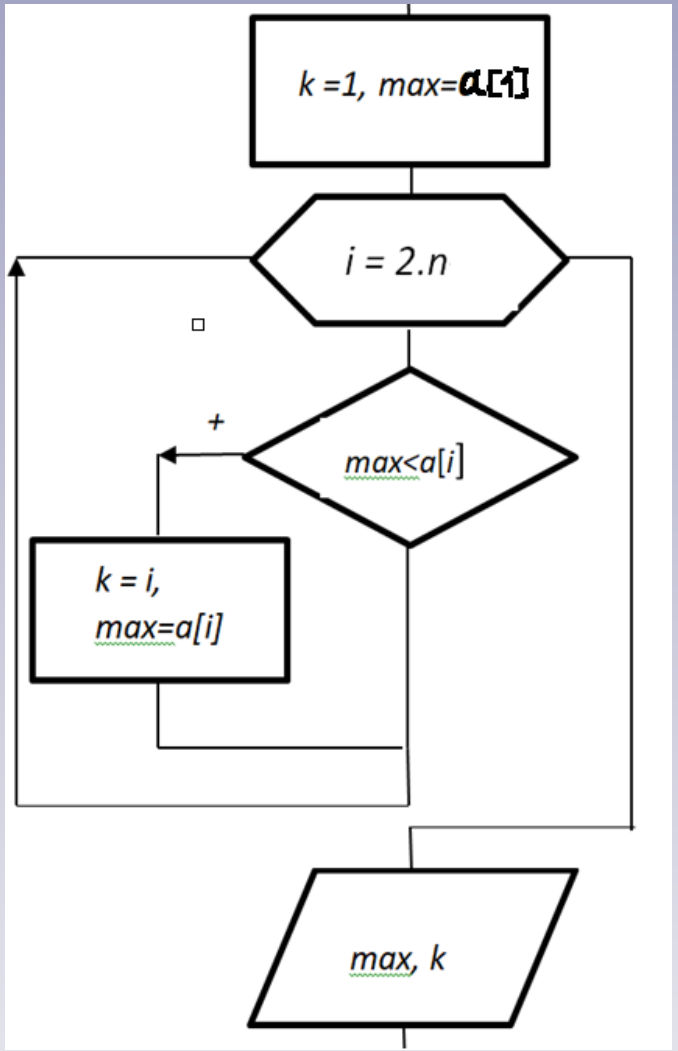


Табл. 7. Тестовый пример

Данные		Результат	
$n = 5$	$a = (3,-1, 10, 1, 6)$	max = 10	k = 3

Табл. 8. Выполнение алгоритма для тестового примера

i	$i \le n$?	$A_{max} < A[i]$?	A_{max}	k
			3	**1**
2	$2 \le 5$? Да.	$3 \le -1$? Нет.		
3	$3 \le 5$? Да.	$3 \le 10$? Да.	10	3
4	$4 \le 5$? Да.	$10 \le 1$? Нет.		
5	$5 \le 5$? Да.	$10 \le 6$? Нет.		
	$6 \le 6$? Нет.	КЦ.		

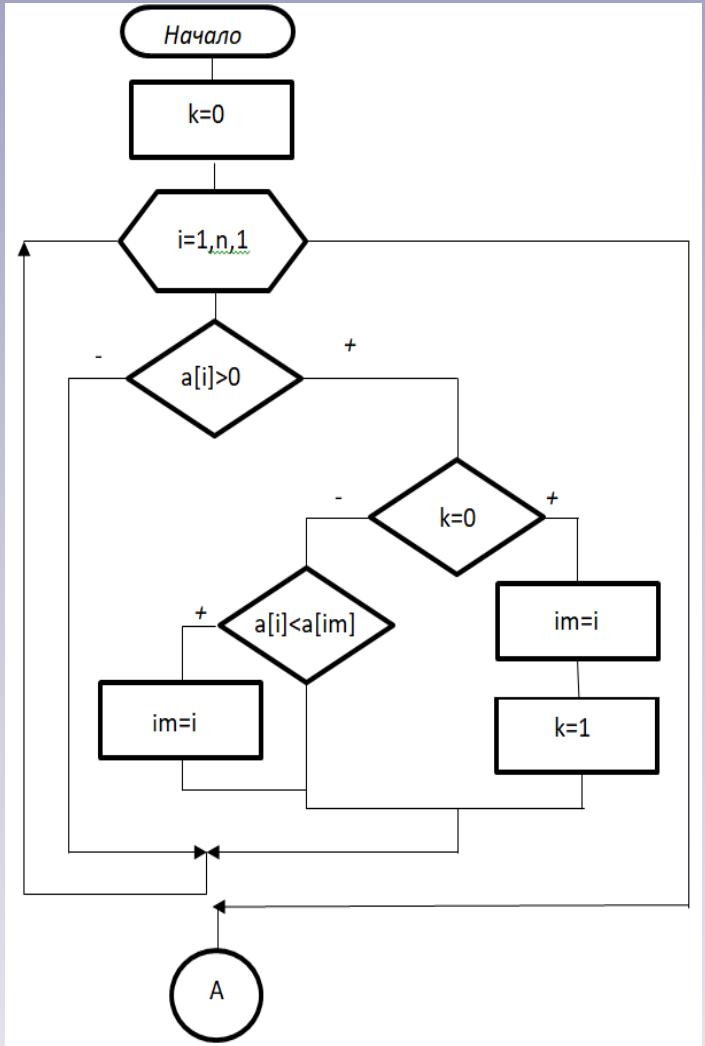
Можно искать не значение искомого элемента, максимума, минимума и т.п., а его индекс

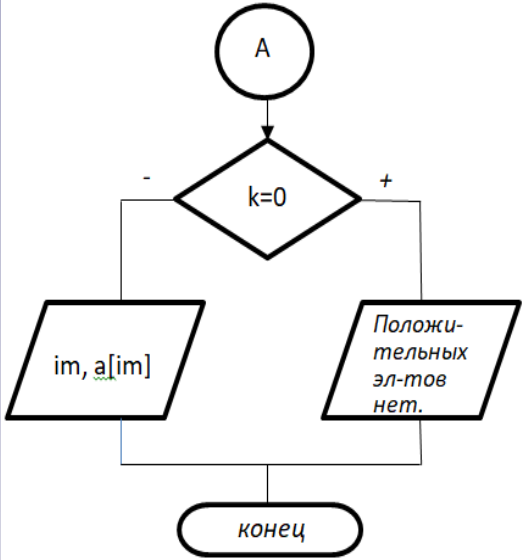
```
imax:=1;
imin:=1;
for i:=2 to N do
  if a[i]<a[imin] then
    imin:=i
  else
    if a[i]>a[imax] then
      imax:=i
```

Определить номер и значение минимального элемента среди положительных элементов массива.

Используем параметр **впервые**, определяющий, впервые ли встретилось в массиве число, удовлетворяющее заданному условию.

```
впервые := да
нц для i от 1 до n
| |
| ...
| если <условие>
| | то |Встретилось число, удовлетворяющее заданному условию
| |  если впервые        |Если оно встретилось впервые,
| |  | то
| |  |  максимальное:= a |принимаем его в качестве
| |  |                   |значения максимальное
| |  |  впервые := нет   |Следующие такие числа уже будут
| |  |                   |встречаться не впервые
| |  | иначе     |Если оно встретилось не впервые,
| |  |           |сравниваем число с величиной максимальное
| |  |  если a > максимальное
| |  |  |то
| |  |  | максимальное := a
| |  |  все
| |  все
| все
кц
```


Начало
k=0
i=1,n,1
a[i]>0
-
+
k=0
-
+
a[i]<a[im]
+
im=i
im=i
k=1
A


A
k=0
-
+
im, a[im]
Положи-
тельных
эл-тов
нет.
конец

Поиск второго по величине элемента массива

```
function max(a,b:integer):integer;
begin
 if(a>b) then result:=a else result:=b;
end;
function min(a,b:integer):integer;
begin
 if(a<b) then result:=a else result:=b;
end;
const n=10;
var i,max1,max2:integer;
    arr:array[1..n] of integer;
begin
  for i:=1 to n do arr[i]:=random(100);
  print(arr);
```

```
max1:=max(arr[1],arr[2]);
max2:=min(arr[1],arr[2]);
  for i:=3 to n do begin
    if (arr[i]>=max1) then begin
      max2:=max1;
      max1:=arr[i];
      continue;
    end;
    if (arr[i]>=max2) then
      max2:=arr[i];
  end;
  writeln(max1,'  ',max2);
end.
```

Output Window

```
[97,1,62,5,12,54,39,64,42,56] 97  64
```

1	2	3	4	5	6	7
8	5	7	3	10	9	2

max1=8;
max2=5;

i:=3 7>=8 нет max1=8;
      7>=5 да max2:=7 max2=7;

i:=4 3>=8 нет
      3>=5 нет

i:=5 10>=8 да max2:=max1:=8 max1=10;
                max1:=10   max2=8;

i:=6 9>=10 нет max1=10;
      9>=8 да max2:=9 max2=9;

i:=7 2>=10 нет
      2>=9 нет