

2016

Глава 4 Модульное программирование

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы
управления

Кафедра Компьютерные системы и сети

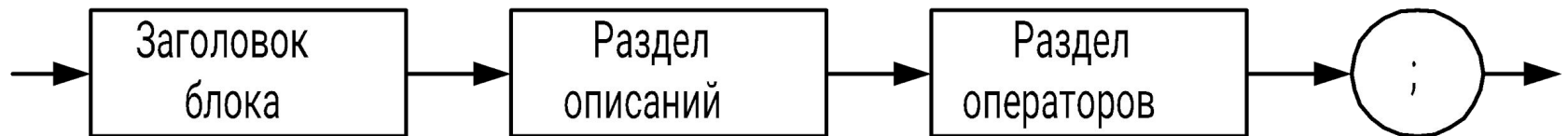
Лектор: д.т.н., проф.

Иванова Галина Сергеевна

4.1 Процедуры и функции

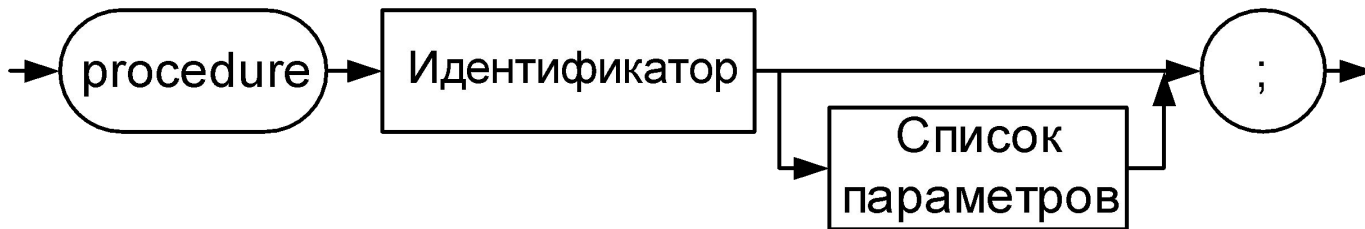
Процедуры и функции – самостоятельные фрагменты программы, соответствующим образом оформленные и вызываемые по имени (программные блоки).

Программный блок:



Заголовки процедуры и функции

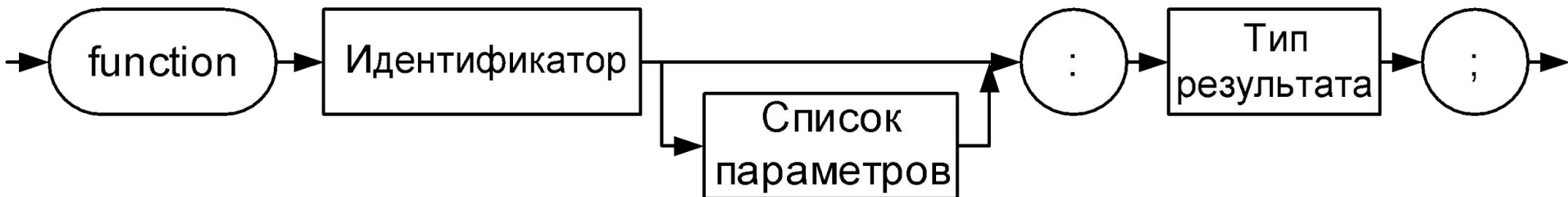
Процедура:



Пример:

```
Procedure RRR (a : integer ; b : real) ;
```

Функция:



Пример:

```
Function F23 (a : integer ; b : real) : boolean ;
```

Локальные и глобальные переменные. Передача данных в подпрограмму

| Классы переменных | Время жизни | Доступность |
|--|--|---|
| Глобальные – объявленные в основной программе | От запуска до завершения программы | Из любого места программы, включая подпрограммы* |
| Локальные – объявленные в подпрограмме | От вызова подпрограммы до возврата управления | Из подпрограммы и подпрограмм, вызываемых из нее* |

* - при отсутствии перекрытия имен

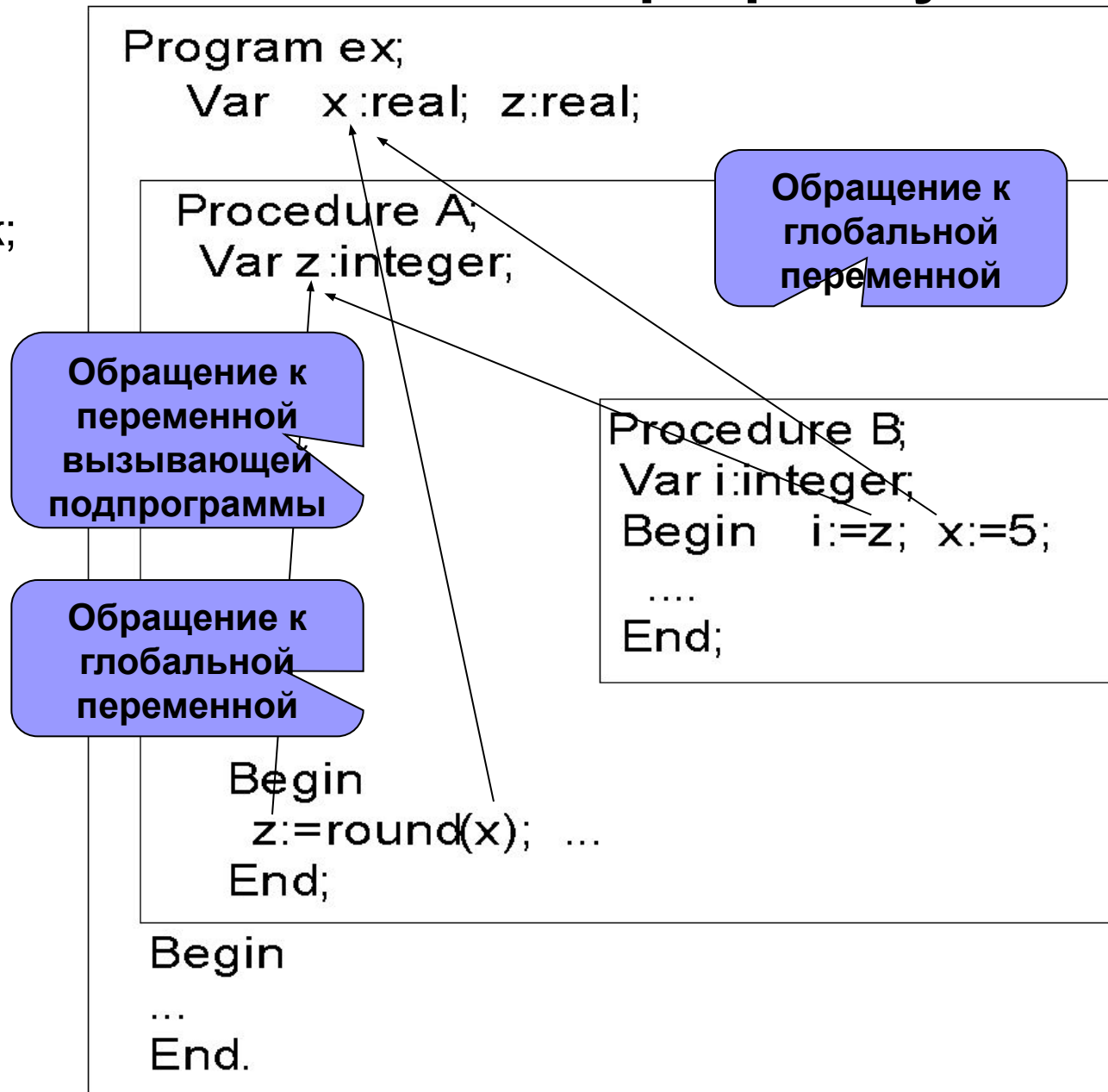
Подпрограмма может получать данные из основной программы:

- а) **неявно** – с использованием свойства доступности глобальных переменных;
- б) **явно** – через параметры.

Неявная передача данных в подпрограмму

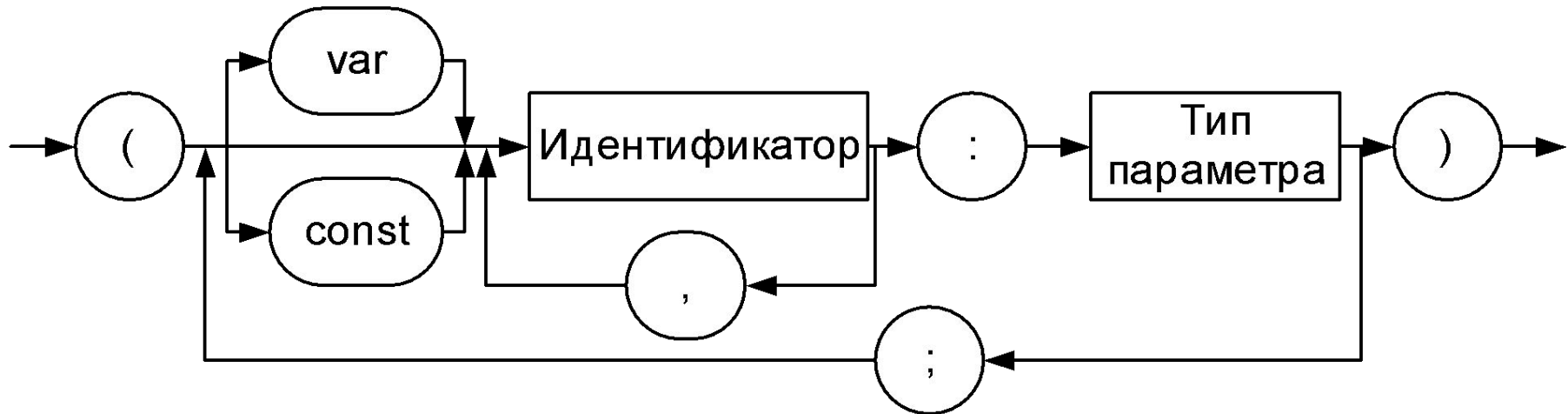
Неявная передача:

- 1) приводит к большому количеству ошибок;
- 2) жестко связывает подпрограмму и данные.



Передача данных через параметры

Список параметров описывается в заголовке:



Параметры, описанные в заголовке – *формальные*.

При вызове подпрограммы необходимо определить *фактические* значения этих параметров – аргументы (константы и переменные).

Формальные и фактические параметры должны соответствовать по количеству, типу и порядку:

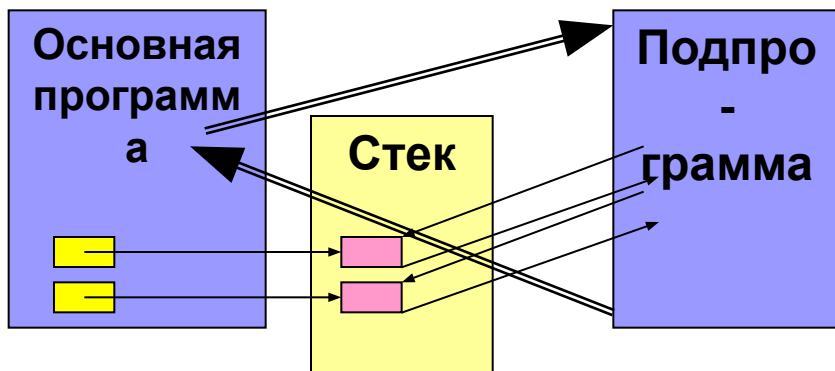
```
function proc(a:integer; b:single):byte; ...
```

```
n := proc(5, 2.1);
```

Arrows point from the number '5' to the parameter 'a' and from '2.1' to the parameter 'b' in the function definition above.

Способы передачи параметров

Передача по значению

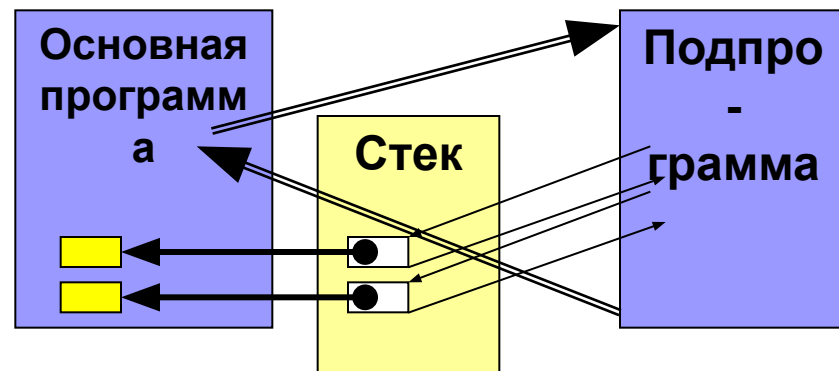


Копии параметров

Работа с копиями параметров

Параметры - **значения** – в подпрограмму передаются **копии фактических параметров**, и никакие изменения этих копий не возвращаются в вызывающую программу.

Передача по ссылке



Адреса параметров

Работа с параметрами через адреса

Параметры - **переменные** – в подпрограмму передаются **адреса фактических параметров**, соответственно все изменения этих параметров в подпрограмме происходят с переменными основной программы.

Способы передачи параметров (2)

- **Параметры-значения** при описании подпрограммы не помечаются, например:

```
function Beta(x:single; n:byte):integer; .
```

- **Параметры-переменные** при описании подпрограммы помечаются служебным словом `var`, например:

```
function Alpha(x:single; Var n:byte):integer; .
```

Ограничение: в качестве фактических значений параметров-переменных **нельзя использовать литералы**:

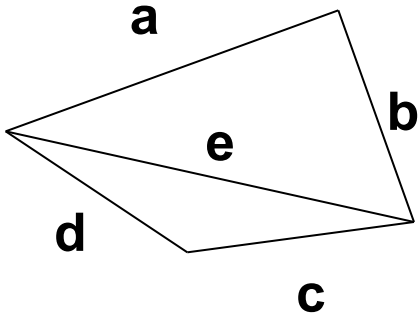
```
Alpha(2.5, 5); // ошибка!
```

правильно: `n:=5; Alpha(2.5, n);`

- **Параметры-константы** – в подпрограмму, так же как и в случае параметров-переменных, передаются **адреса фактических параметров**, но при попытке изменить значение параметра компилятор выдает сообщение об ошибке; такие параметры при описании подпрограммы помечаются служебным словом `const`, например:

```
function Alpha(const x:single; n:byte); .
```


Определение площади четырехугольника



Площадь четырехугольника определяем как сумму площадей треугольников.

Площадь треугольника определяем по формуле Герона.

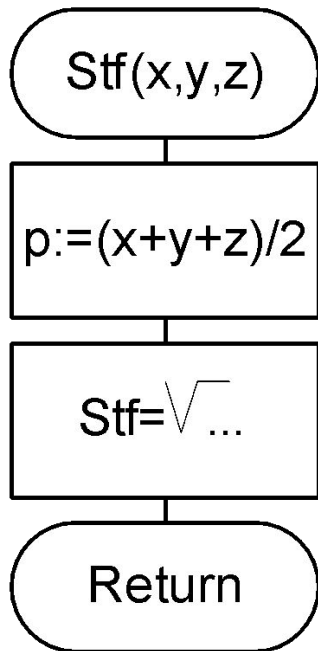
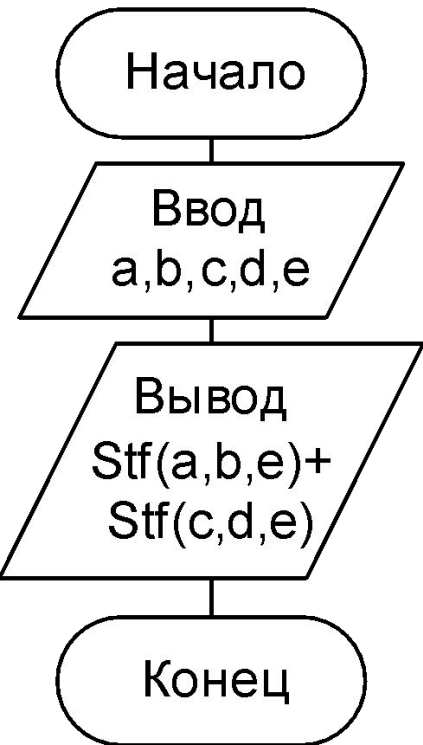
В качестве подпрограммы реализуем вычисление площади треугольника, поскольку эта операция выполняется два раза с разными параметрами.

Схемы алгоритмов подпрограмм

Подпрограмма-функция

Начало алгоритма подпрограммы

Формальные параметры

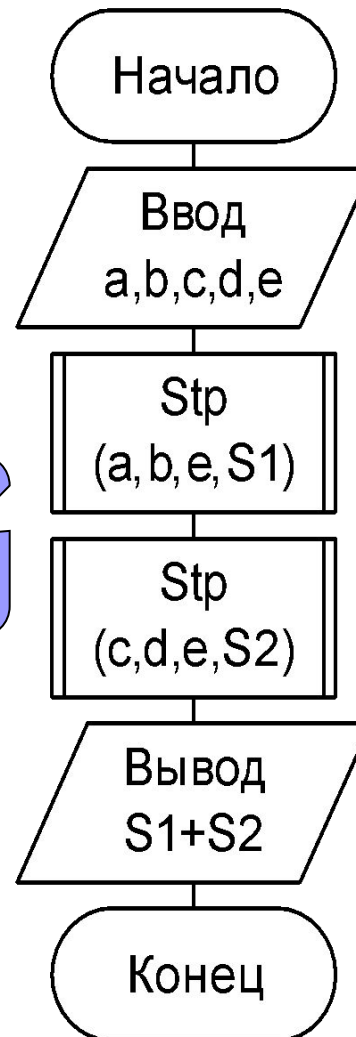


Фактические параметры

Завершение подпрограммы

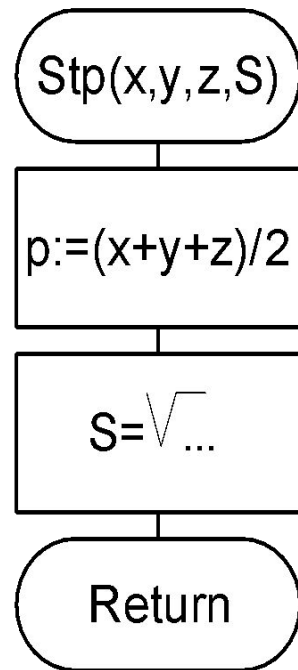
Подпрограмма-процедура

Формальный параметр-переменная в заголовке на схеме не выделяется



Вызов процедуры

Фактическое значение параметра-переменной



Функция

```
Program Ex4_1;  
{$APPTYPE CONSOLE}  
Uses SysUtils;  
Var A,B,C,D,E:single;  
Function Stf(const X,Y,Z: single): single;  
  Var p:single;  
  begin  
    p:=(X+Y+Z)/2;  
    Result:=sqrt(p*(p-X)*(p-Y)*(p-Z)); // или Stf:=..  
  end;  
Begin  
  WriteLn('Input a,b,c,d,e:');  
  ReadLn(A,B,C,D,E);  
  WriteLn('S=',Stf(A,B,E)+Stf(C,D,E):7:3);  
  ReadLn;  
End.
```

Глобальные
переменные

Тип
возвращаемого
значения

Локальная
переменная

Вычисление
возвращаемого
значения

Вызов
функции из
выражения

Процедура

```
Program Ex4_2;  
{$APPTYPE CONSOLE}  
uses SysUtils;  
Var A,B,C,D,E:real; S1,S2:single;  
Procedure Stp(const X,Y,Z:single;var S:single);  
    Var p:single;  
    begin    p:=(X+Y+Z)/2;  
            S:=sqrt(p*(p-X)*(p-Y)*(p-Z));  
    end;  
Begin  
    WriteLn('Input a,b,c,d,e');    ReadLn(A,B,C,D,E);  
    Stp(A,B,E,S1);  
    Stp(C,D,E,S2);  
    WriteLn('S= ',S1+S2:7:3);  
    ReadLn;  
End.
```

Глобальные
переменные

Возвращаемое
значение

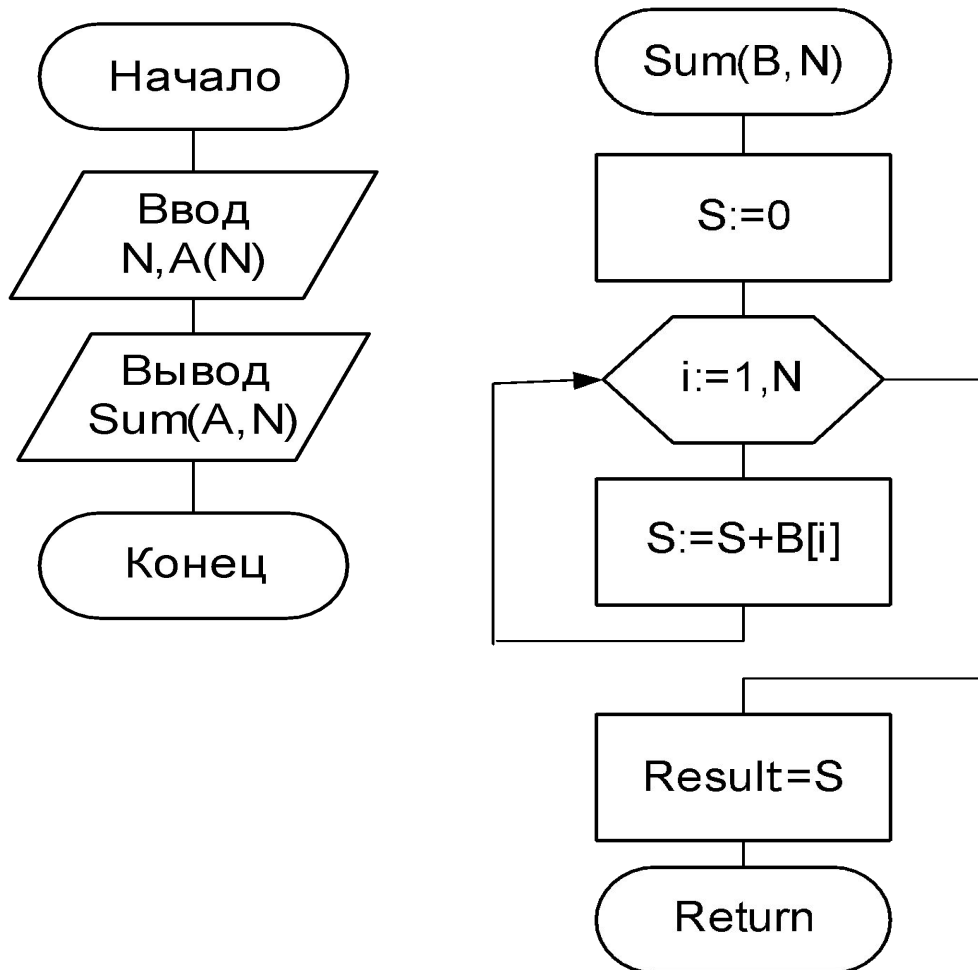
Локальная
переменная

Вызов
процедуры

Параметры структурных типов

Структурные типы параметров должны быть **предварительно объявлены**.

Пример. Функция вычисления суммы элементов массива.



Программа

```
Program Ex4_3;  
{ $APPTYPE CONSOLE }  
Uses SysUtils;  
Type mas=array[1..10] of integer;  
Var a:mas;    i,n:integer;  
Function sum(b:mas; n:integer):integer;  
    Var s:integer; i:integer;  
    Begin s:=0;  
        for i:=1 to n do s:=s+b[i];  
    Result:=s;  
End;  
Begin    Write('Input n:');  
        ReadLn(n);  
        for i:=1 to n do Read(a[i]);  
        ReadLn;  
        WriteLn('Sum =', sum(a,n));  
        ReadLn;  
End.
```

Предварительное
объявление типа
параметра

Объявление
параметра
структурного типа

Фактический
параметр
структурного типа

4.2 Модули

Модуль – это автономно компилируемая коллекция программных ресурсов, предназначенных для использования другими модулями и программами.

Ресурсы – переменные, константы, описания типов и подпрограммы. Все ресурсы, определенные в модуле делят на:

- 1) **внешние** – предназначенные для использования другими программами и модулями.
- 2) **внутренние** – предназначенные для использования внутри модуля.

Структура модуля:

Unit <Имя модуля>;

Interface

<Интерфейсная секция>

Implementation

<Секция реализации>

[Initialization

<Секция инициализации>

[Finalization

<Секция завершения>]]

End.

Имя модуля должно совпадать с именем файла, в котором он описан.

Подключение модуля к программе

Подключение модуля к программе осуществляется по имени:

Uses <Имя модуля1>, <Имя модуля2>, ...;

Объявление модулей в файле проекта

Если:

- к проекту подключается модуль, который находится в каталоге, не совпадающем с каталогом проекта и не указанном в путях компилятора;
- в путях компилятора имеется несколько модулей с одинаковыми именами,

то необходимо указать местонахождение модуля:

```
Uses Strings in 'C:\Classes\Strings.pas' ;
```

```
Uses Strings in '..\Strings.pas' ; {относительно текущего кат.}
```

Модули, объявленные в файле проекта с указанием `in ...`, считаются частью проекта, т. е. доступны через средства работы с проектом среды.

Использование указания `in ...` в файлах с расширением `pas` не допустимо.

Модуль с функцией вычисления суммы

```
Unit Summa; {должен находиться в файле Summa.pas}
```

```
Interface
```

```
    type mas=array[1..10] of integer;
```

```
    function sum(b:mas;n:integer):integer;
```

```
Implementation
```

```
    Function sum;
```

```
        Var s:integer;i:integer;
```

```
        begin
```

```
            s:=0;
```

```
            for i:=1 to n do s:=s+b[i];
```

```
            Result:=s;
```

```
        end;
```

```
End.
```

Программа вычисления суммы

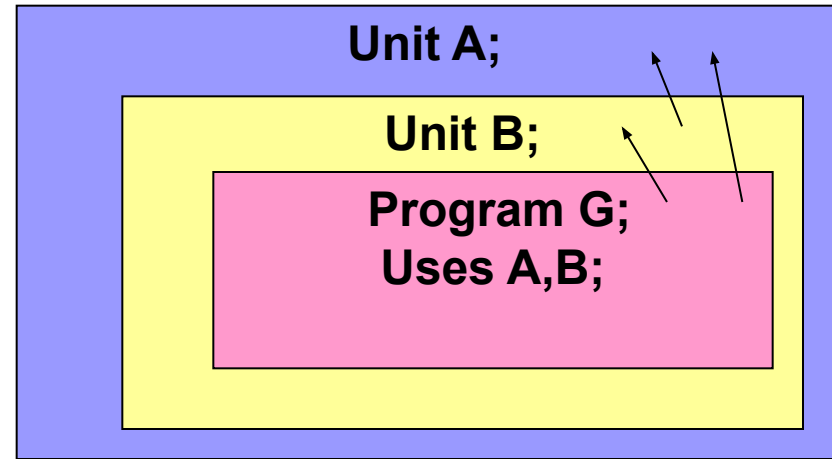
```
Program Ex4_4;  
{$APPTYPE CONSOLE}  
Uses SysUtils,  
      Summa in 'Summa.pas';  
Var a:mas;  
      i,n:integer;  
Begin  
    Write('Input n:');  
    Readln(n);  
    for i:=1 to n do Read(a[i]);  
    ReadLn;  
    WriteLn('Sum =',sum(a,n));  
    ReadLn;  
End.
```

Правило видимости имен ресурсов модуля

Ресурсы модуля перекрываются ресурсами программы и ранее указанных модулей.

Для доступа к перекрытым ресурсам модуля используют точечную нотацию:

<Имя модуля>.<Имя ресурса>



Пример:

```
Unit A;  
    Interface  
        Var X:real; ...  
End.
```

```
Program ex;  
    Uses A;  
        Var X:integer;  
    Begin  
        X:=10;  
        A.X:=0.45; ...
```

4.3 Создание универсальных подпрограмм

4.3.1 Открытые массивы и строки

Открытый массив – конструкция описания типа массива без указания типа индексов. **Используется только при объявлении формальных параметров.**

Примеры:

```
array of single;  
array of integer;
```

Индексы открытого массива всегда начинаются с 0.

Размер можно:

- передать через дополнительный параметр;
- получить, используя функцию High(<Идентификатор массива>).

Функция с открытым массивом

```
Unit Summa2;
```

```
Interface
```

```
Function sum(b:array of integer; n:integer):integer;
```

```
Implementation
```

```
Function sum;
```

```
var s:integer;
```

```
i:integer;
```

```
begin
```

```
    s:=0;
```

```
    for i:=0 to n-1 do s:=s+b[i];
```

```
    Result:=s;
```

```
end;
```

```
End.
```

Размер
массива

Тестирующая программа

```
Program Ex4_5;  
{ $APPTYPE CONSOLE }  
  
Uses  
    SysUtils,  
    Summa2 in 'Summa2.pas';  
  
Var a:array[1..10] of integer;  
    i,n:integer;  
Begin  
    Write('Input n:');  
    ReadLn(n);  
    for i:=1 to n do Read(a[i]);  
    ReadLn;  
    WriteLn('Sum=', sum(a,n));  
    ReadLn;  
  
End.
```

Открытые строки

Для строк, передаваемых в подпрограмму как параметр-переменная, Паскаль осуществляет контроль длины строки.
Чтобы избежать его необходимо использовать «открытые» строки.

Пример. Программа, формирующая строку из букв латинского алфавита.

```
Unit Stroka;  
Interface  
    Procedure Add(var s:openstring);  
Implementation  
    Procedure Add;  
    Var Ch:char;  
        begin  
            Ch:=s[length(s)];  
            s:=s+chr(succ(Ord(Ch)));  
        end;  
End.
```

Тестирующая программа

```
program Ex4_6;  
  {$APPTYPE CONSOLE}  
uses  SysUtils,  
      Stroka in 'Stroka.pas';  
  
Var S:string[26];i:integer;  
Begin  
    s:='A';  
    for i:=2 to 26 do Add(s);  
    WriteLn(s);  
    ReadLn;  
end.
```


4.3.2 Нетипизированные параметры

Нетипизированные параметры – параметры-переменные, тип которых при объявлении не указан.

Для приведения нетипизированного параметра к определенному типу можно использовать:

1) автоопределенное преобразование типов:

```
Procedure Proc (Var a) ; ...  
    ...b:= Integer (a)+10; ...
```

2) наложенное описание переменной определенного типа:

```
Procedure Proc (Var a) ; ...  
    Var r:real absolute a;...
```

Суммирование чисел различных типов

```
Unit Summa4;
```

```
Interface
```

```
  type ttype=(treal,tinteger);
```

```
  function sum(var x;n:integer;t:ttype):real;
```

```
Implementation
```

```
function sum;
```

```
  Var  mr:array[1..3000] of real absolute x;
```

```
      mi:array[1..3000] of integer absolute x;
```

```
      s:real;i:integer;
```

```
begin s:=0;
```

```
  if t=treal then
```

```
    for i:=1 to n do s:=s+mr[i]
```

```
  else for i:=1 to n do s:=s+mi[i];
```

```
  sum:=s;
```

```
end;
```

```
End.
```

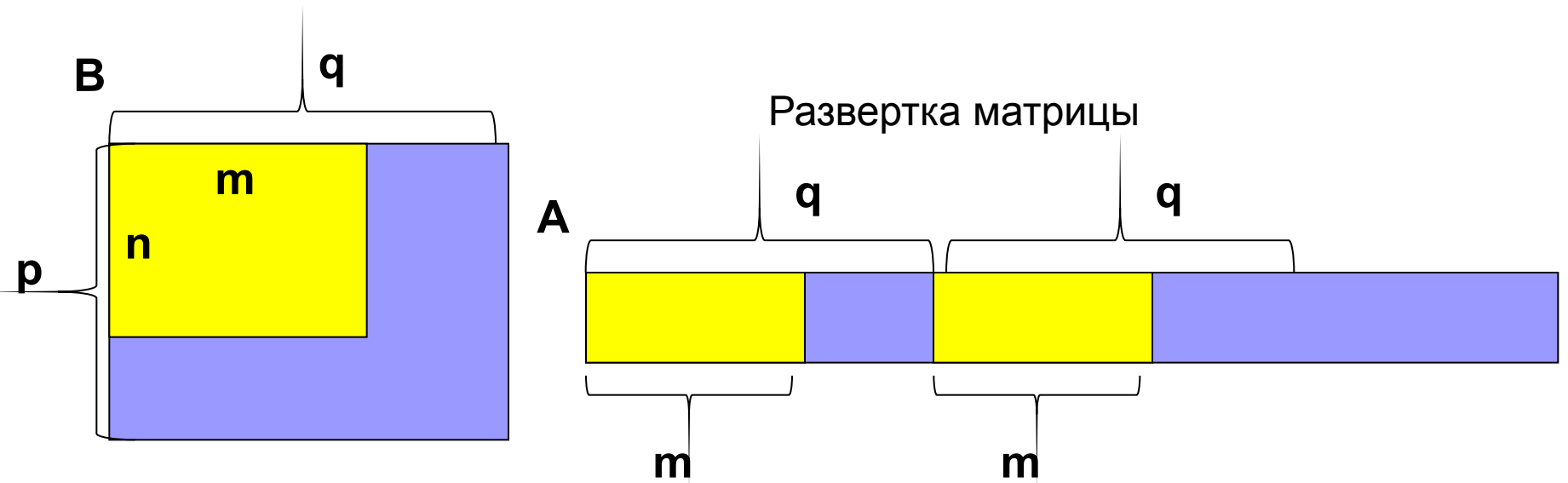
Параметр
перечисляемого типа,
определяющий тип
элементов массива

Описанный массив
накладывается по
адресу параметра

Тестирующая программа

```
program Ex4_7;  
  {$APPTYPE CONSOLE}  
uses SysUtils,  
     Summa4 in 'Summa4.pas';  
Var a:array[1..10] of integer;  
     b:array[1..15] of real;  
     i,n:integer;  
Begin  
  for i:=1 to 10 do Read(a[i]);  
  ReadLn;  
  WriteLn('Sum=',sum(a,10,tinteger):8:1);  
  for i:=1 to 15 do Read(b[i]);  
  ReadLn;  
  WriteLn('Sum=',sum(b,15,treal):8:1);  
  ReadLn;  
end.
```

Универсальные подпрограммы с многомерными массивами



$B[i, j]$

\Leftrightarrow

$A[(i-1) * q + j]$

Транспонирование матрицы

В транспонированной матрице B: $b[i, j] = a[j, i]$

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |

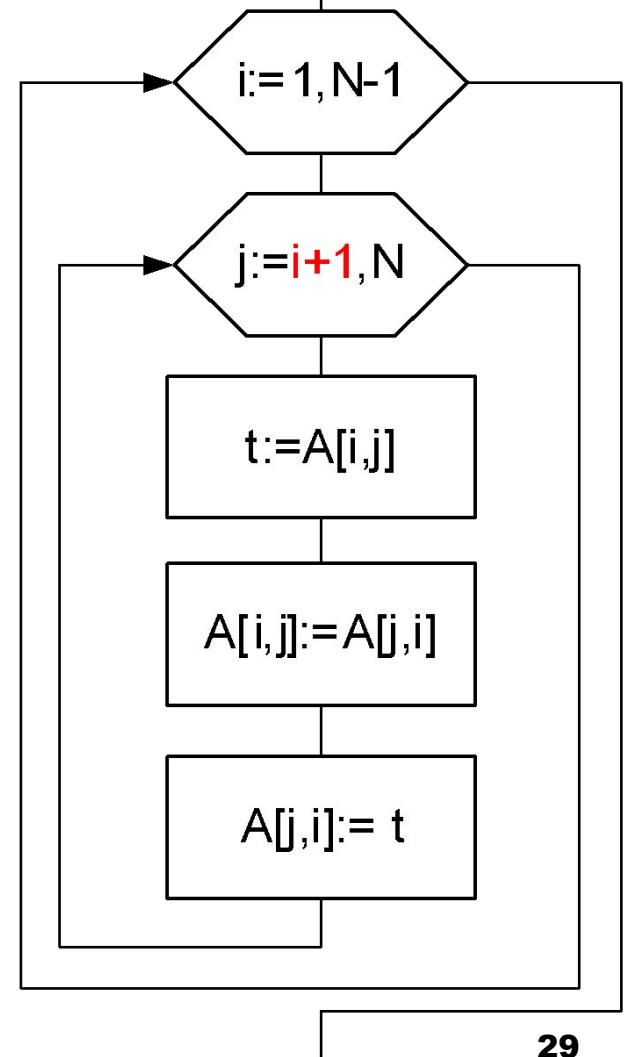
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

Если $i=1$, то первый номер столбца $j=2$

$i=2 \Rightarrow j=3$

$i=3 \Rightarrow j=4$

$i=4 \Rightarrow j=5$



Универсальная подпрограмма

Unit Matrica;

Interface

```
procedure Tran(Var x;n,q:integer);
```

Implementation

```
procedure Tran;
```

```
Var a:array[1..3000] of real absolute x;
```

```
    i,j:integer;          t:single;
```

```
begin
```

```
    for i:=1 to n-1 do
```

```
        for j:= i+1 to n do
```

```
            begin t:=a[(i-1)*q+j];
```

```
                a[(i-1)*q+j]:=a[(j-1)*q+i];
```

```
                a[(j-1)*q+i]:=t;
```

```
            end;
```

```
end;
```

End.

Тестирующая программа

```
Program Ex4_8;  
{$APPTYPE CONSOLE}  
Uses SysUtils,  
      Matrica in 'Matrica.pas';  
Var   a:array[1..10,1..10] of single;      i,j:integer;  
Begin WriteLn('Input a(5*5):');  
      for i:=1 to 5 do  
        begin for j:=1 to 5 do Read(a[i,j]);  
              ReadLn;  
        end;  
      tran(a,5,10);  
      WriteLn('Result:');  
      for i:=1 to 5 do  
        begin for j:=1 to 5 do Write(a[i,j]:6:2);  
              WriteLn;  
        end;  
      ReadLn;  
End.
```

4.3.3 Параметры процедурного типа

Параметры процедурного типа используются для передачи в подпрограмму имен процедур и функций.

Для объявления процедурного типа используется заголовок подпрограммы, в котором отсутствует имя:

```
Type  proc=procedure (a,b,c:real;Var d:real);  
      func=function(x:real):real;
```

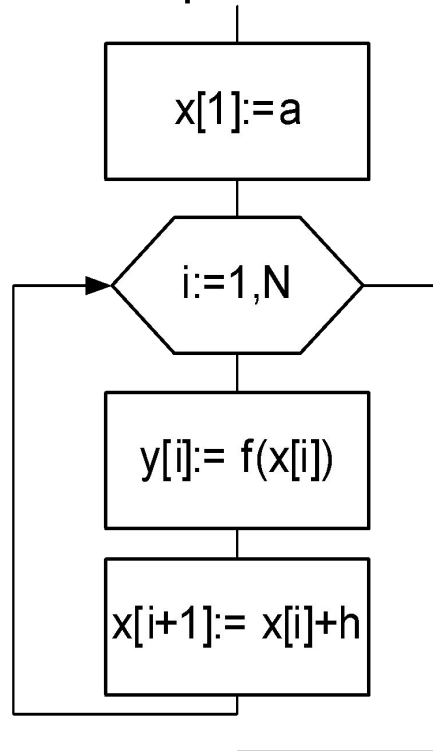
Значениями переменных процедурных типов являются идентификаторы процедур и функций с соответствующими заголовками:

```
Var  f:func;  
...  
f:=fun1;...
```


Табулирование функций

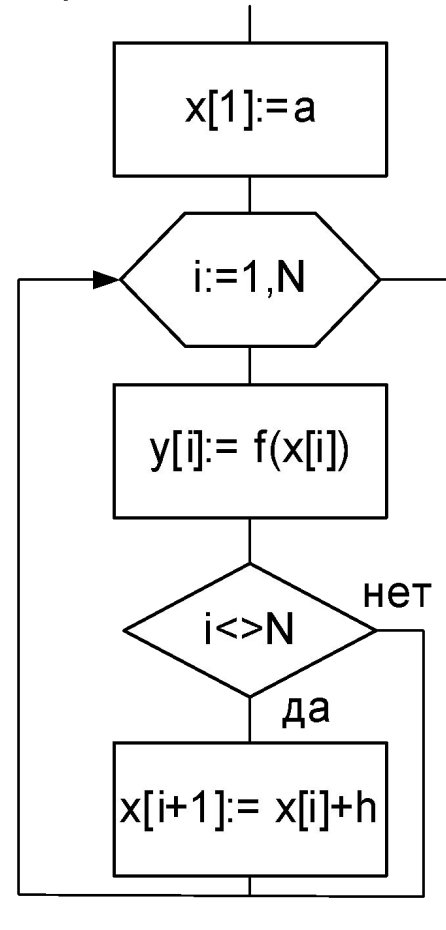
Табулирование – построение таблицы значений:

| x | y |
|------|------|
| 0.01 | 5.56 |
| 0.02 | 6.34 |
| 0.03 | 7.56 |
| ... | |

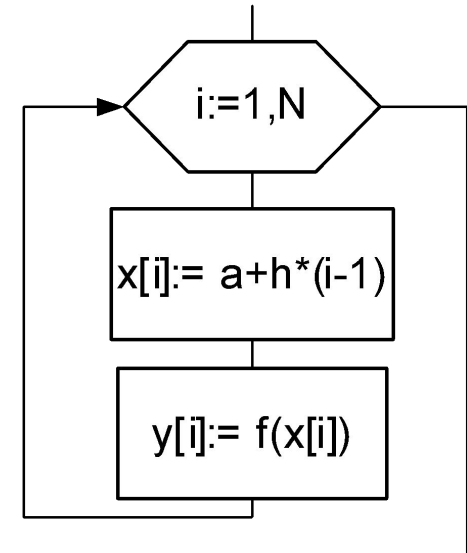


Рассчитывается лишний $N+1$ элемент

Исключение расчета лишнего элемента за счет дополнительной проверки



Расчет значения аргумента требует больше времени



Подпрограмма табулирования функции

```
Unit SFun;  
Interface  
    Type func=function(x:Single):Single;  
    Procedure TabFun(f:func;a,b:Single;n:integer;  
                    var Masf,MasX:array of Single);  
Implementation  
Procedure TabFun;  
    Var h,x:Single;    i:integer;  
    Begin  
        h:=(b-a)/(n-1);  
        for i:=0 to n-1 do  
            begin MasX[i]:= a+h*i;  
                  Masf[i]:=f(MasX[i]);  
            end;  
    End;  
End.
```

Тестирующая программа

```
Program Ex4_9;  
{ $APPTYPE CONSOLE }  
Uses SysUtils,  
      SFun in 'SFun.pas';  
Var masF1,masX1:array[1..10] of Single;  
    masF2,masX2:array[1..20] of Single;  
    i:integer;  
function F1(x:Single):Single;  
  Begin  
    F1:=sin(x);  
  end;  
function F2(x: Single):Single;  
  Begin  
    F2:=exp(x)+cos(x);  
  end;
```

Тестирующая программа. Раздел операторов

Begin

```
TabFun (F1, 0, 2, 10, masF1, masX1) ;
```

```
WriteLn ('Table 1') ;
```

```
for i:=1 to 10 do
```

```
    WriteLn (masX1:4:1, masF1[i]:7:1) ;
```

```
WriteLn ('Table 2') ;
```

```
TabFun (F2, 0, 2, 20, masF2, masX2) ;
```

```
for i:=1 to 20 do
```

```
    WriteLn (masX2:4:1, masF2[i]:7:1) ;
```

```
ReadLn ;
```

End.

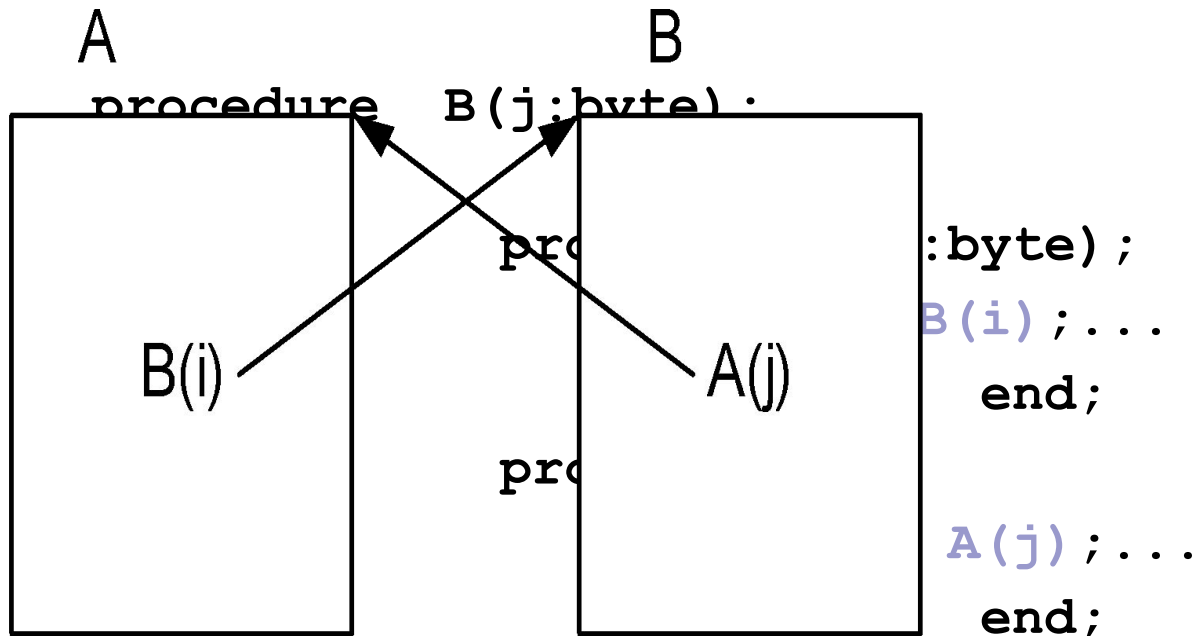
4.4 Рекурсия

4.4.1 Основные понятия

Рекурсия – организация вычислений, при которой процедура или функция обращаются к самим себе.

Различают явную и косвенную рекурсии. При явной – в теле подпрограммы существует вызов самой себя, при косвенной – вызов осуществляется в подпрограммах, вызываемых из рассматриваемой.

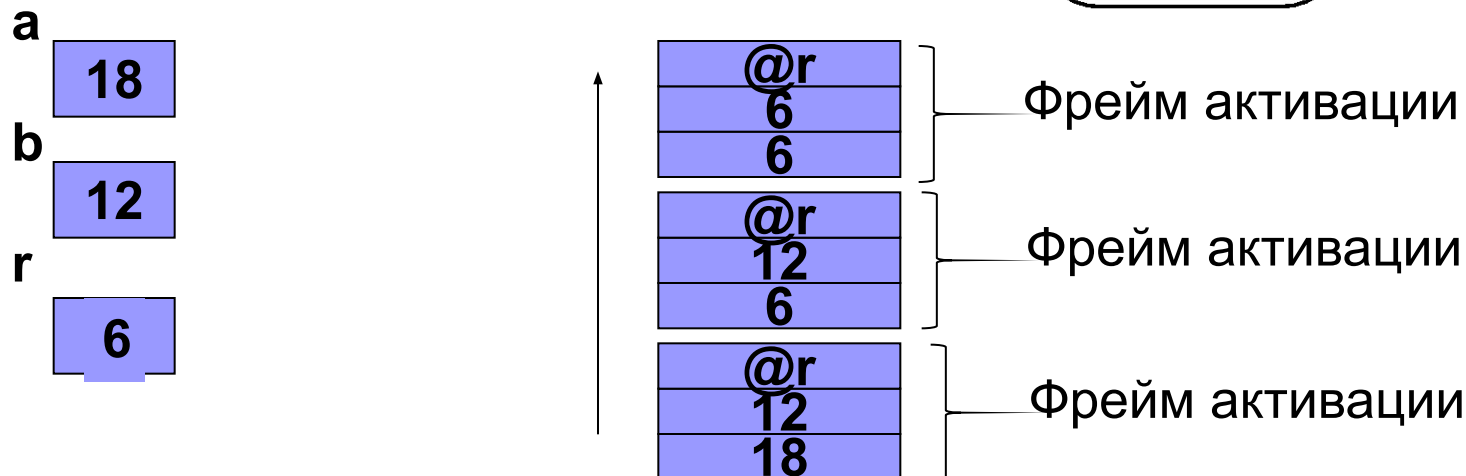
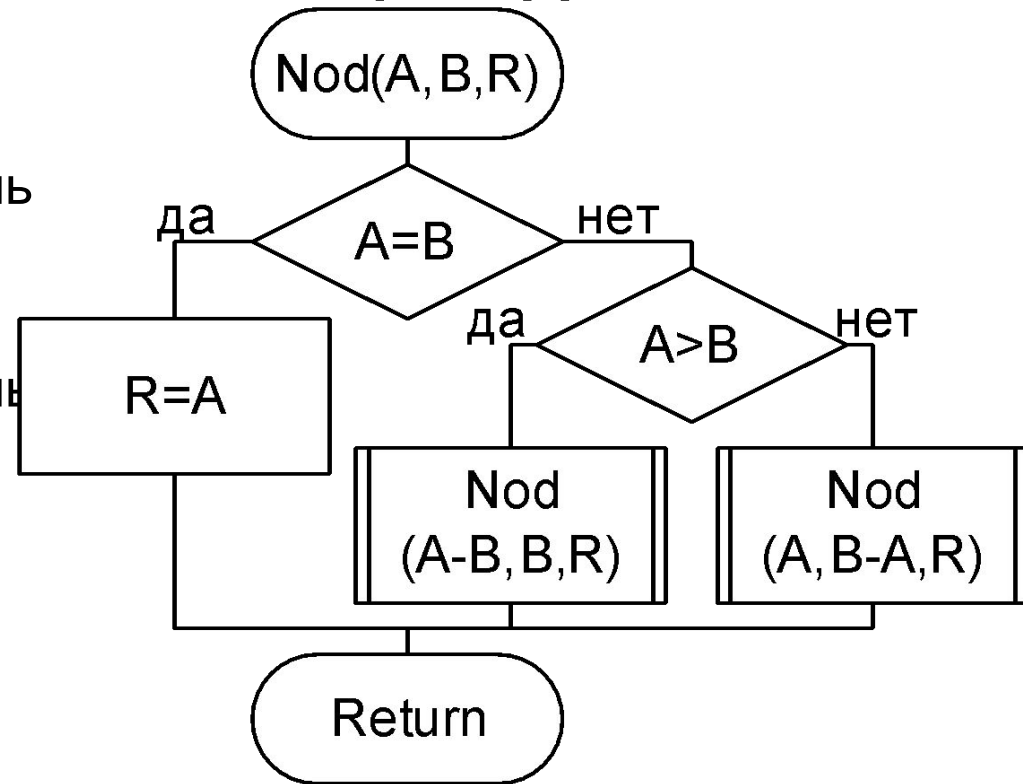
Косвенная рекурсия требует **предопределения forward**:



Вычисление наибольшего общего делителя

Базисное утверждение: если два числа равны, то их наибольший общий делитель равен этим числам.

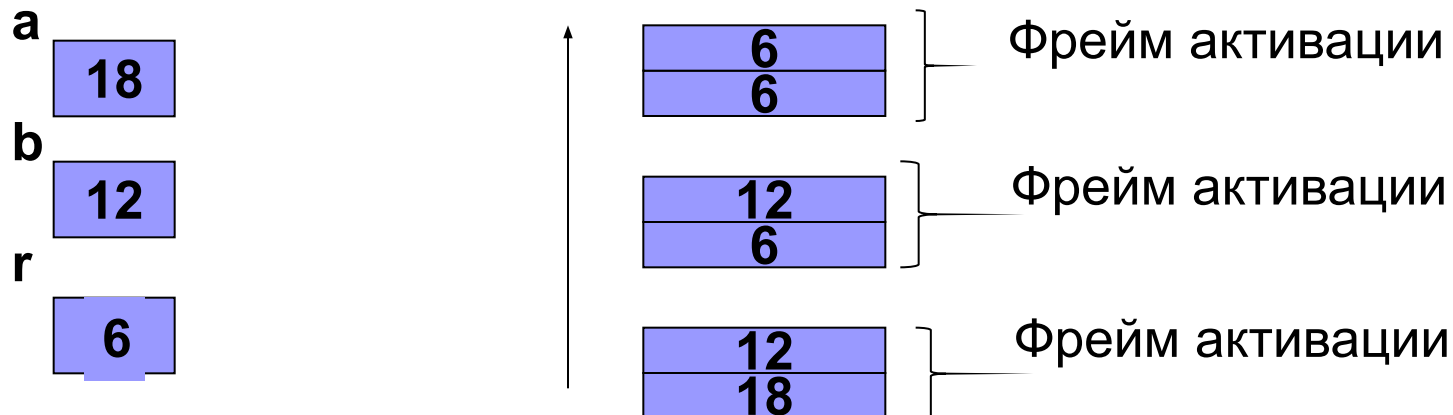
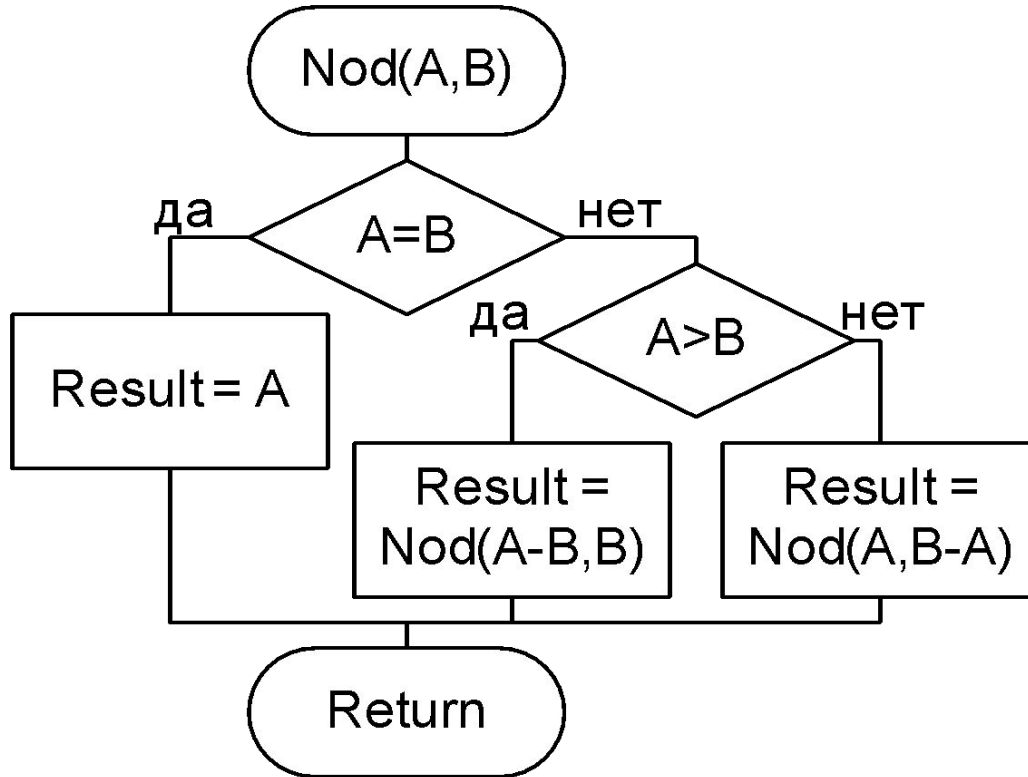
Рекурсивное утверждение: наибольший общий делитель двух чисел равен наибольшему общему делителю их разности и меньшего из чисел.



Вычисление наибольшего общего делителя (2)

```
Program Ex4_10a;  
{ $APPTYPE CONSOLE }  
Uses SysUtils;  
Var a,b,r:integer;  
Procedure nod(a,b:integer; var r:integer);  
  Begin  
    if a=b then r:=a  
    else if a>b then nod(a-b,b,r)  
           else nod(a,b-a,r)  
  End;  
Begin  
  WriteLn('Input A,B');  
  ReadLn(a,b);  
  nod(a,b,r);  
  WriteLn(r);  
  ReadLn;  
End.
```

Вычисление наибольшего общего делителя (3)



Вычисление наибольшего общего делителя (4)

```
Program Ex4_10b;  
{$APPTYPE CONSOLE}  
Uses SysUtils;  
Var a,b,r:integer;  
Function nod(a,b:integer):integer;  
    begin if a=b then Result:=a  
           else  
               if a>b then Result:=nod(a-b,b)  
                   else Result:=nod(a,b-a)  
           end;  
Begin WriteLn('Input A,B');  
    ReadLn(a,b);  
    r:=nod(a,b);  
    WriteLn(r);  
    ReadLn;  
End.
```

4.4.2 Фрейм активации.

Структура рекурсивной подпрограммы

Каждое обращение к рекурсивной подпрограмме вызывает независимую *активацию* этой подпрограммы.

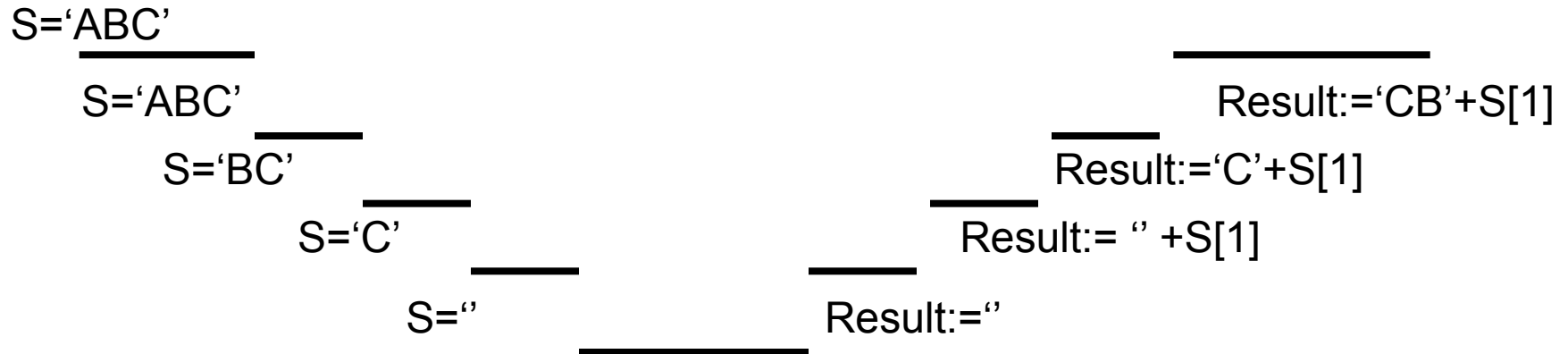
Совокупность данных, необходимых для *одной* активации рекурсивной подпрограммы, называется *фреймом активации*.

Фрейм активации включает

- локальные переменные подпрограммы;
- копии параметров-значений;
- адреса параметров-переменных и параметров-констант (4 байта);
- копию строки результата (для функций типа string);
- служебную информацию (≈ 12 байт, точный размер этой области зависит от способа передачи параметров).

Переворот строки

1) последовательное отсечение начального элемента и добавление его в конец результирующей строки:



```
Function reverse1(const st:string):string;
```

```
Begin
```

```
  if length(st)=0 then Result:=''
```

```
  else
```

```
    Result:= reverse1(copy(st,2,length(st)-1))+ st[1];
```

```
End;
```

Фрейм активации: $V=4 + 256 + \text{<служебная область>} \approx 272$.

Переворот строки (2)

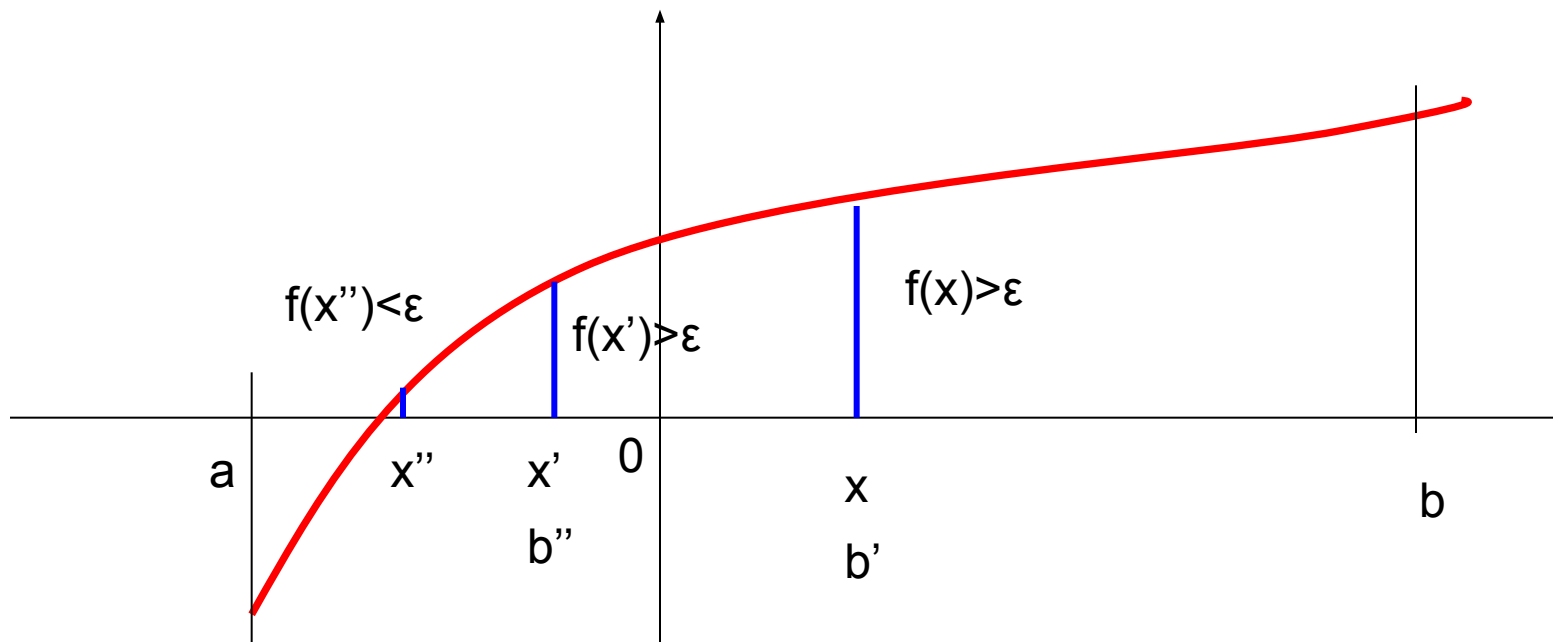
2) последовательная перестановка элементов,
например

ABCDE \Rightarrow EBCDA \Rightarrow EDCBA

```
Procedure reverse2 (var ss:string; n:integer);  
Var temp:char;  
Begin if n<=length(ss) div 2 then  
    begin temp:=ss[n];  
        ss[n]:=ss[length(ss)-n+1];  
        ss[length(ss)-n+1]:=temp;  
        reverse2(ss,n+1);  
    end;  
End;
```

Фрейм активации: $V=4+4+1+\langle\text{служебная область}\rangle\approx 21$

Определение корней уравнения на заданном отрезке. Метод деления пополам

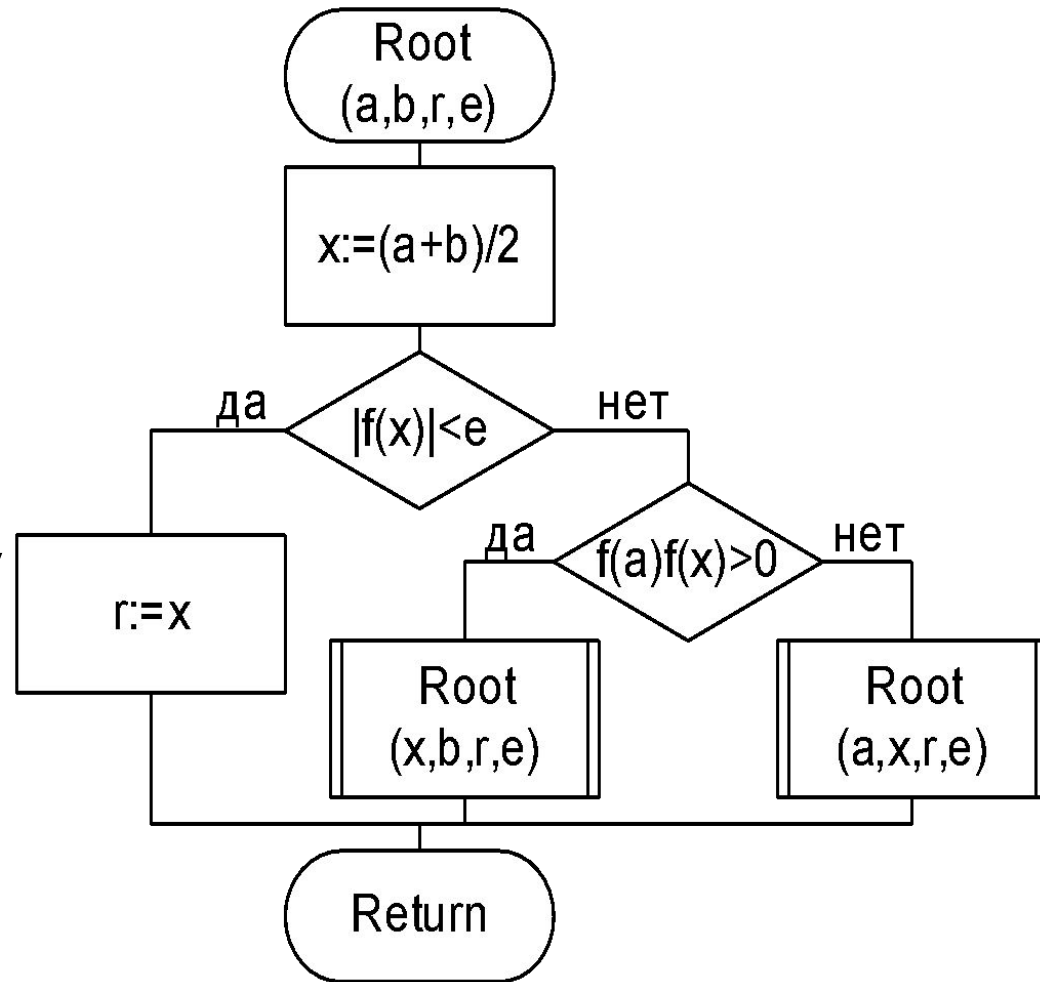


$$x = (b-a)/2$$

Определение корней уравнения на заданном отрезке (2)

Базисное утверждение: Если абсолютная величина функции в середине отрезка не превышает заданного значения погрешности, то координата середины отрезка и есть корень.

Рекурсивное утверждение: Корень расположен между серединой отрезка и тем концом, значение функции в котором по знаку не совпадает со значением функции в середине отрезка.

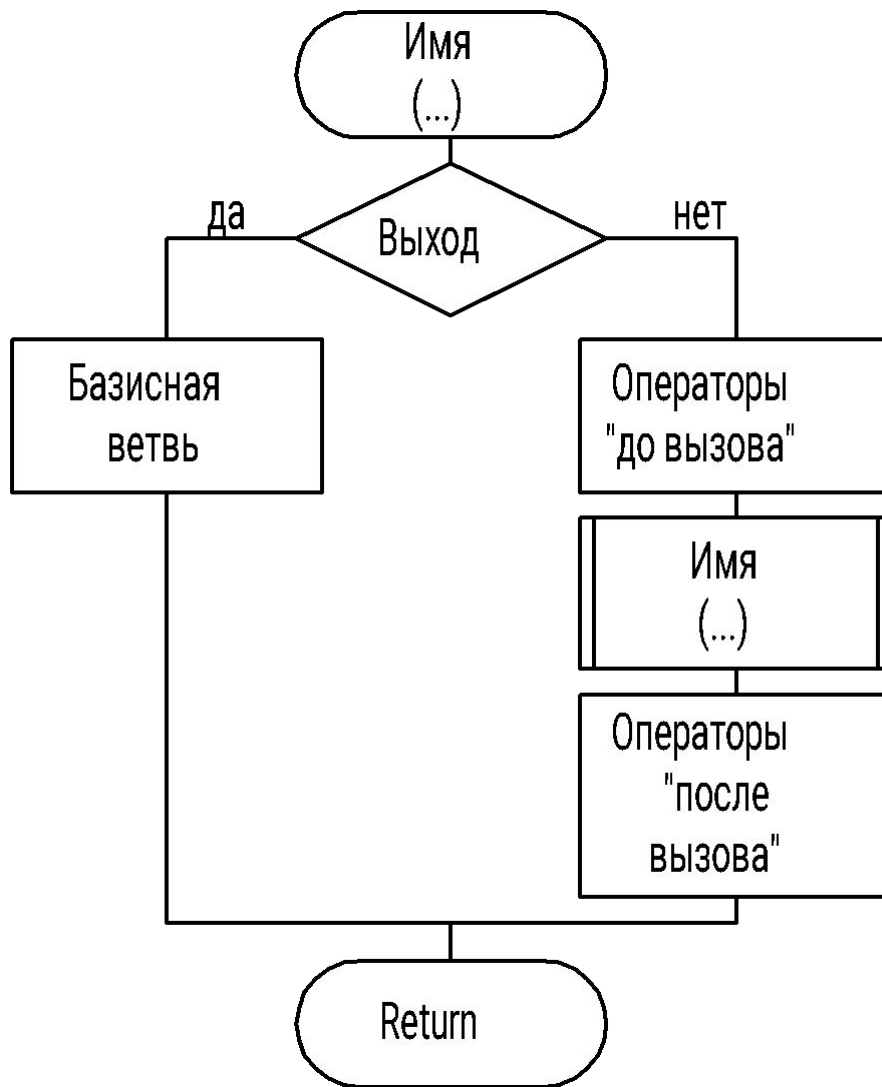


Определение корней уравнения на заданном отрезке (3)

```
Program Ex4_11;  
{ $APPTYPE CONSOLE }  
Uses SysUtils;  
Var a,b,eps,x:real;  
Procedure root(a,b,eps:real;var r:real);  
  Var f,x:real;  
  Begin  x:=(a+b)/2; f:=x*x-1;  
         if abs(f)>=eps then  
           if (a*a-1)*f>0 then root(x,b,eps,r)  
                               else root(a,x,eps,r)  
         else r:=x;  
  End;  
Begin  WriteLn('Input a,b,eps'); ReadLn(a,b,eps);  
       root(a,b,eps,x);  
       WriteLn('Root x=',x:9:7); ReadLn;  
End.
```

Если корней на заданном отрезке нет, то произойдет зацикливание!

Структура рекурсивной подпрограммы



«Операторы после вызова», выполняются *после возврата управления* из рекурсивно вызванной подпрограммы.

Пример. Распечатать положительные элементы массива в порядке следования, а отрицательные элементы – в обратном порядке. Признак конца массива – 0.

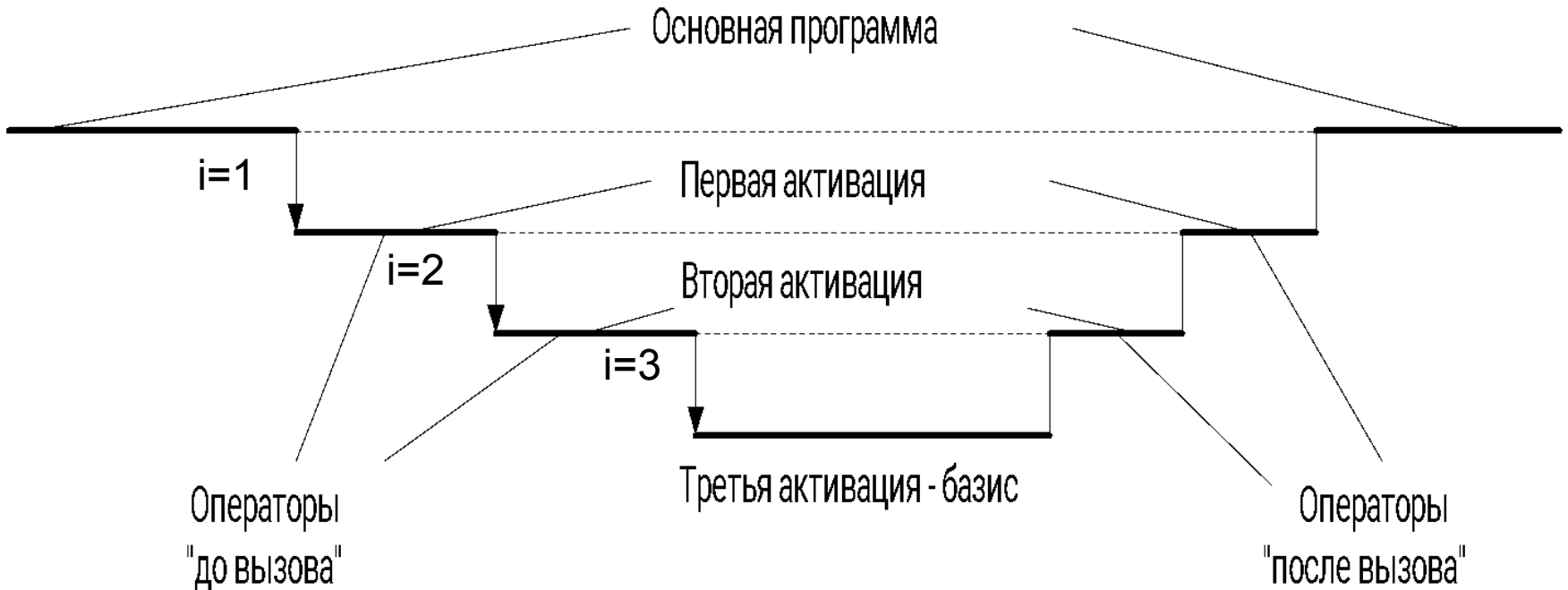
Просмотр массива

Дан массив, завершающийся нулем и не содержащий нулей в середине, например:

4 -5 8 9 -3 0.

Необходимо напечатать положительные элементы в том порядке, как они встречаются в массиве и отрицательные элементы в обратном порядке:

4 8 9 -3 -5



Просмотр массива. Программа

```
Program Ex4_12;  
{ $APPTYPE CONSOLE }  
Uses SysUtils;  
Type mas=array[1..10] of real;  
Var x:mas;    i:integer;  
Procedure print(const x:mas;i:integer);  
    Begin if x[i]=0 then WriteLn('***')  
          else  
            begin  
              if x[i]>0 then WriteLn(i,x[i]);  
                print(x,i+1);  
              if x[i]<0 then WriteLn(i,' ', x[i]);  
            end  
    End;  
End;
```

Просмотр массива. Программа (2)

Begin

 i:=0;

 repeat

 i:=i+1;

 Read(x[i])

 until x[i]=0;

 print(x,1);

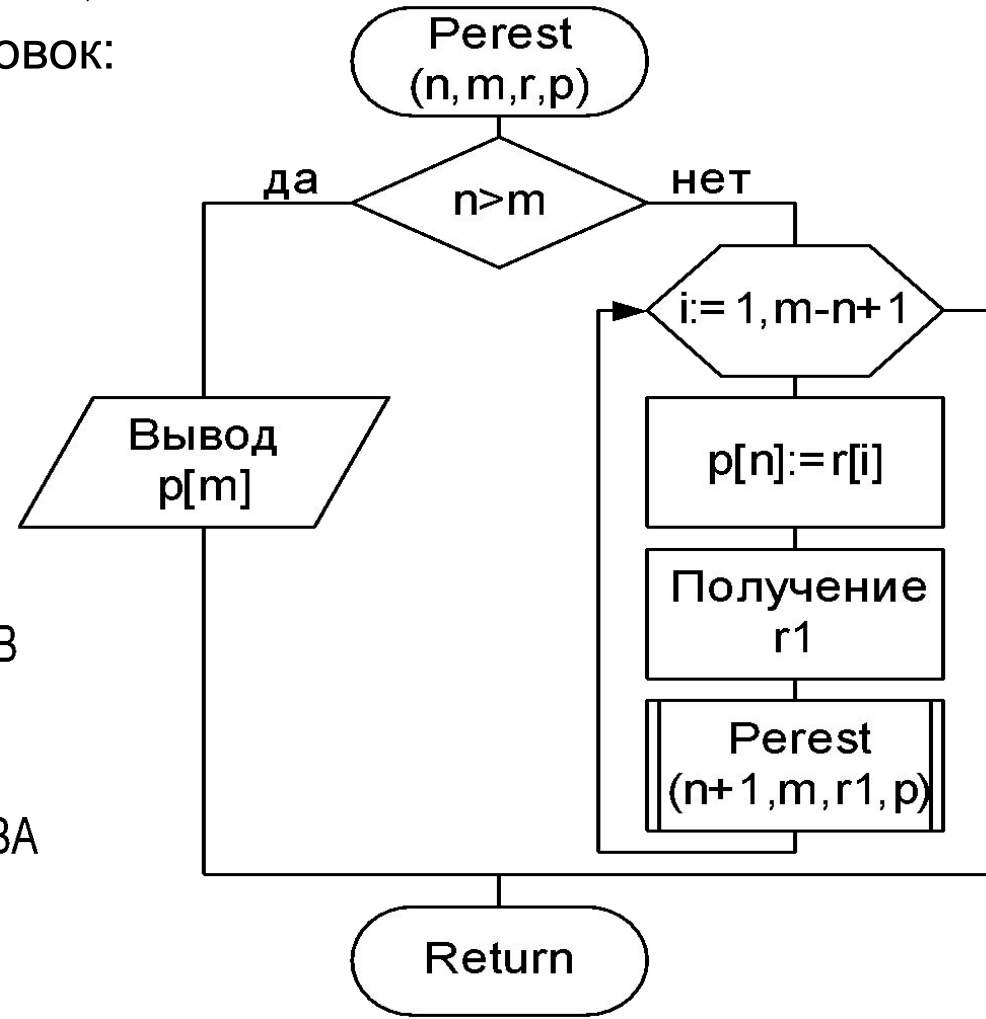
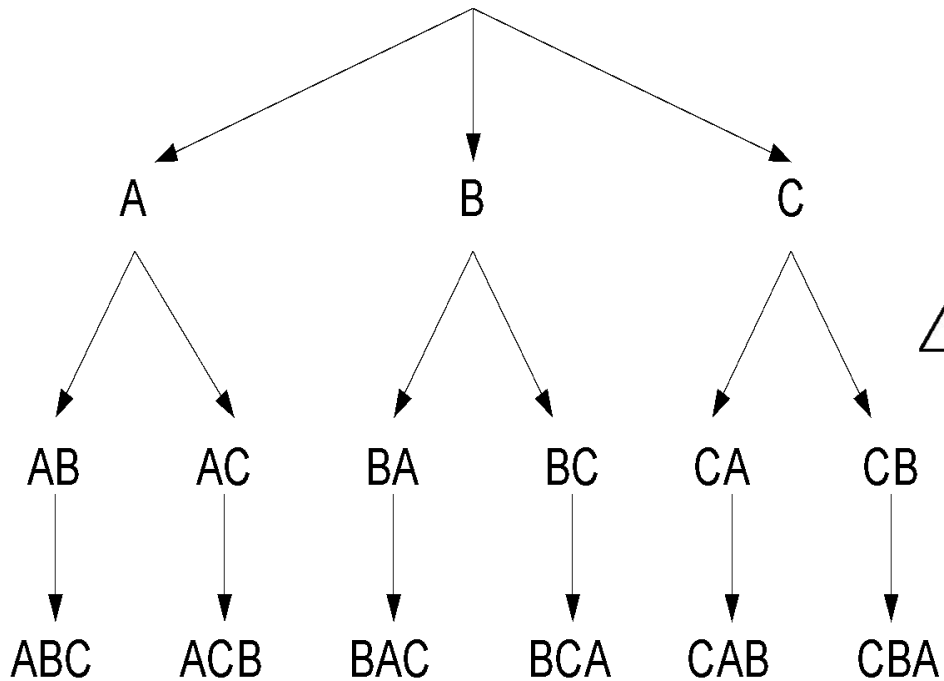
 ReadLn;

End.

4.4.3 Древовидная рекурсия. Перестановки

A,B,C \Rightarrow ABC, ACB, BAC, BCA, CAB, CBA.

Схема формирования перестановок:



Перестановки (2)

```
Program Ex4_13;  
{$APPTYPE CONSOLE}  
Uses SysUtils;  
Type mas=array[1..3] of char;  
Var a:mas='ABC'; Var pole:mas;  
procedure Perest(n,m:integer; Const r:mas;  
                Var pole:mas);  
    Var r1:mas; k,j,i:integer;  
    Begin  
        if n>m then  
            begin  
                for i:=1 to m do Write(pole[i]); WriteLn;  
            end  
        else
```

Перестановки (3)

```
    for i:=1 to m-n+1 do
        begin
            pole[n]:=r[i];
            k:=1;
            for j:=1 to m-n+1 do
                if j<>i then
                    begin    r1[k]:=r[j];    k:=k+1; end;
                Perest(n+1,m,r1,pole);
            end;
        end;
    End;
Begin
    Perest(1,3,a,pole);
    ReadLn;
End.
```