

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 Функции

Функция – самостоятельный фрагмент программы, соответствующим образом оформленный и вызываемый по имени. Функция вычисляет и возвращает в точку вызова скалярное значение или адрес, поэтому вызывается в выражениях соответствующего типа.

```
Тип_результата Имя ([Список_параметров]) {  
    [< Объявление переменных и констант >]  
    {Оператор}  
}
```

Функция, не возвращающая значения в точку вызова, описывается с результатом типа void, называется **процедурой** и вызывается отдельным оператором.

Пример:

```
int max(int a, int b);  
int max(int a, int b) {  
    if (a>b) return a;  
    else return b;  
}  
k = max(x,y)+5;
```

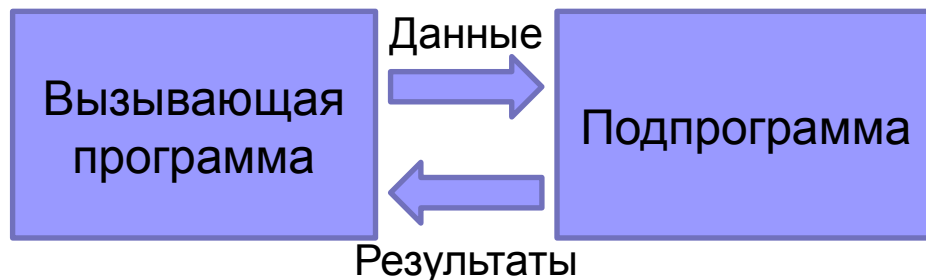
Объявление функции – **прототип** – позволяет описывать функции в любом порядке

Описание (тело) функции

Вызов функции в выражении

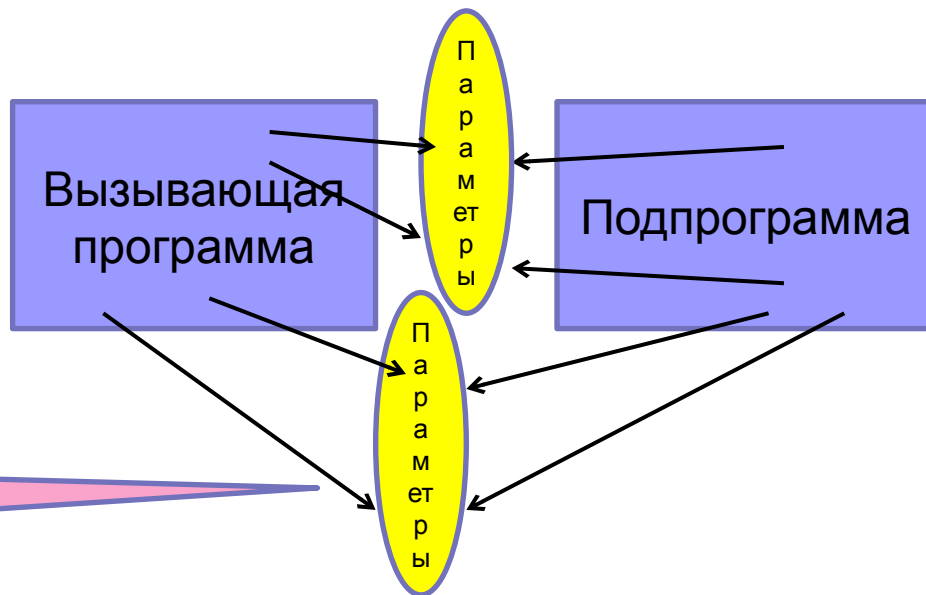
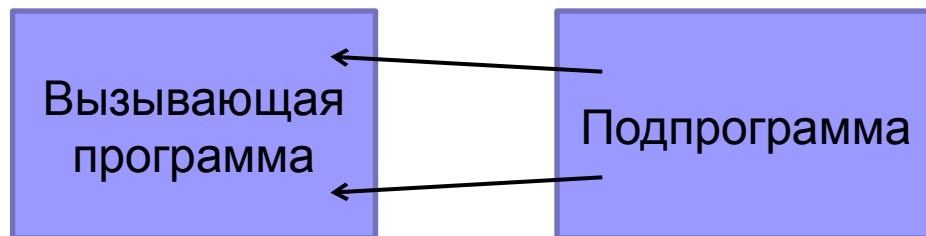
Передача данных в подпрограмму

Подпрограммы (процедуры и функции) могут получать данные из основной программы и возвращать результаты.



Способы передачи:

- а) **неявно** — с использованием свойства доступности некоторых видов переменных из подпрограмм (жесткая связь);
- б) **явно** — через параметры (гибкая связь).



Универсальность -
возможен вызов
подпрограмм с другими
параметрами !!!

4.2 Классы памяти переменных

Класс памяти	Время жизни	Доступность	Место
Автоматические (локальные), объявляются внутри блоков (подпрограмм, программных блоков {...})	От момента вызова блока до его завершения	В пределах блока, в котором она объявлена	Стек
Внешние (глобальные), объявляются вне подпрограмм и/или со спецификатором extern	От момента вызова программы до ее завершения	Вся программа, кроме подпрограмм, в которых переменная перекрыта локальной	Область данных программы
Статические , объявляются внутри подпрограмм со спецификатором static	то же самое	В пределах подпрограммы, в которой она объявлена	Область внутри области данных
Внешние статические , объявляются static вне подпрограмм и/или со спецификаторами extern static – в подпрограмме	то же самое	В пределах файла	Область данных файла

Примеры

1. Автоматические (локальные) переменные

```
int main()
{ int a;...}
void abc()
{ int a;...}
```

Автоматические (локальные) переменные – две разные переменные, которые **временно** размещаются в стеке

2. Внешние переменные (**extern**)

```
[extern] int a;
int main()
{extern int a;...}
void abc()
{extern int a;...}
int bcd()
{int a;...}
```

Одна и та же переменная – размещается в сегменте данных программы

Автоматическая переменная, которая внутри функции "перекрывает" вызова внешней

Классы памяти (2)

3. Статические переменные (**static**)

```
abc() {  
    int a=1;  
    static int b=1;  
    ... a++; b++; ...  
}
```

В отличие от автоматической статическая переменная хранит предыдущее значение, которое при каждом запуске увеличивается на 1. Размещается в специальной области сегмента данных

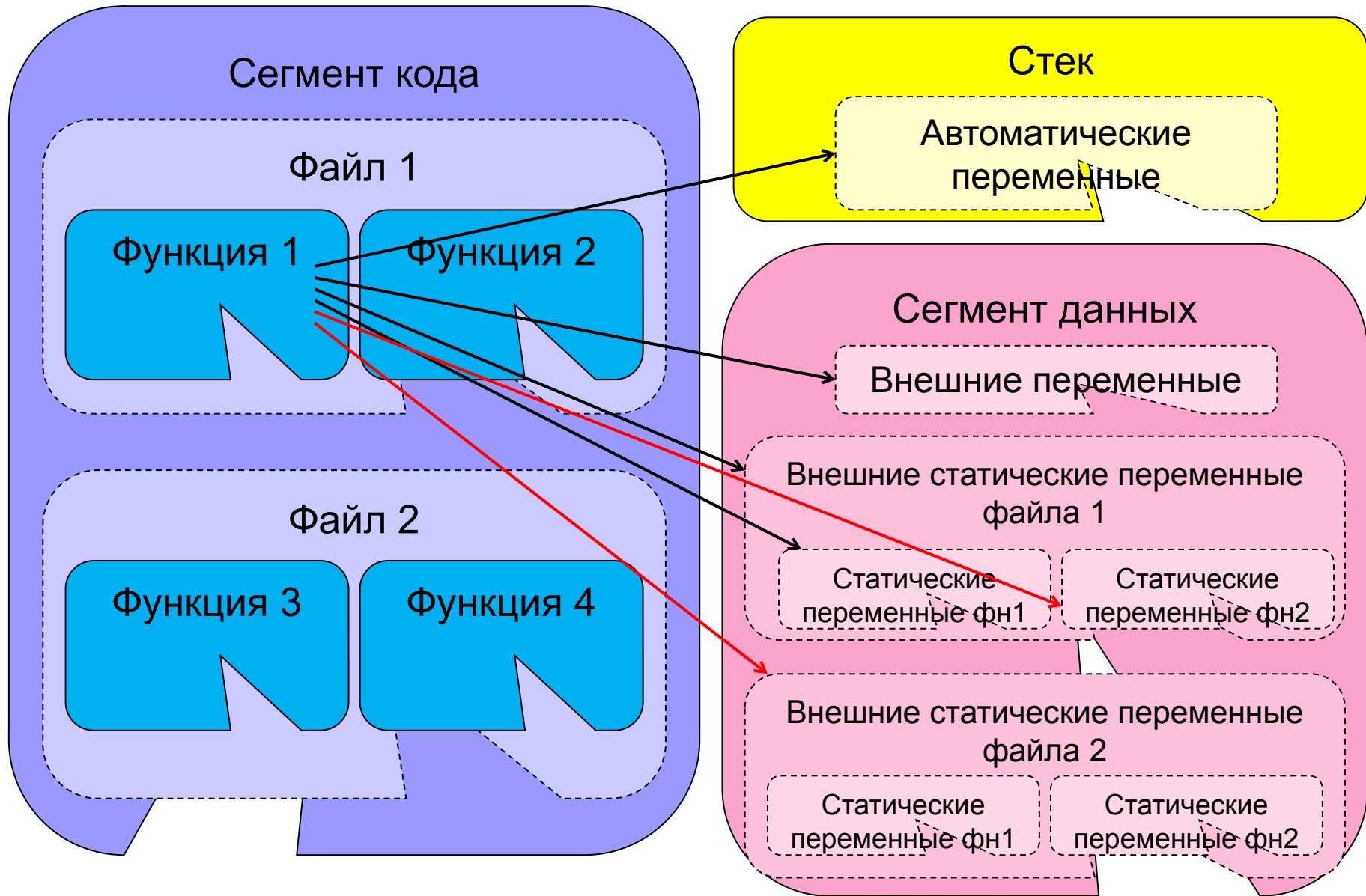
4. Внешние статические переменные (**extern static**)

```
int a;  
[extern] static int b;
```

Файл

Внешняя переменная доступна во всех файлах программы, а внешняя статическая - только в том файле, где описана

Расположение переменных разных классов



4.3 Параметры подпрограмм

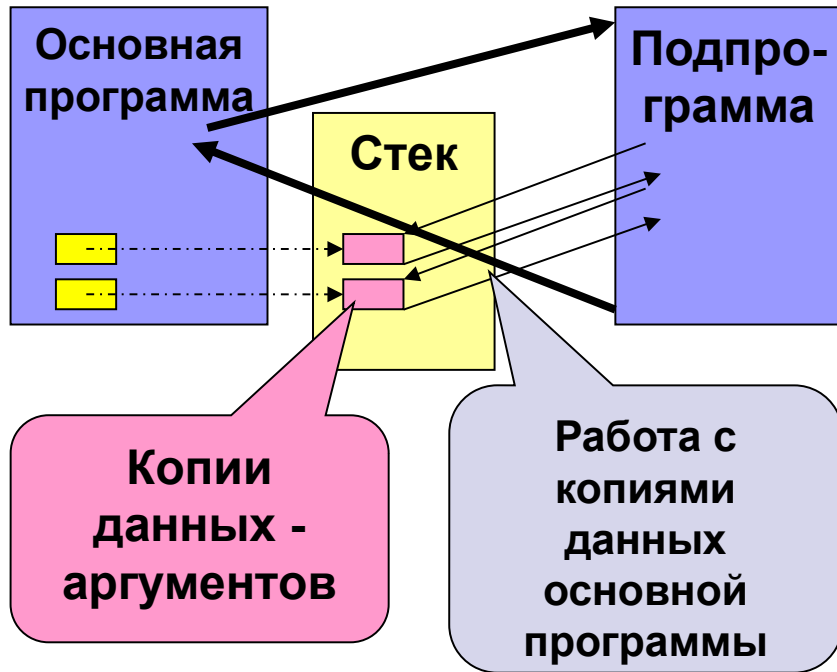
Информация о параметрах при вызове подпрограммы копируется в стек!

Различают следующие варианты передачи параметров:

- **передача значения** – в стек копируется значение и все операции подпрограмма выполняет с копией, в вызывающую программу *изменения не возвращается*;
- **передача адреса (указателя или ссылки)** – в стек копируется адрес параметра, все операции подпрограмма выполняет по указанному адресу, т.е. с самим значением в вызывающей программе, соответственно это *значение изменяется*;
- **константная передача** – в стек копируется адрес параметра, но подпрограмме *запрещено изменять* значение параметра по переданному адресу.

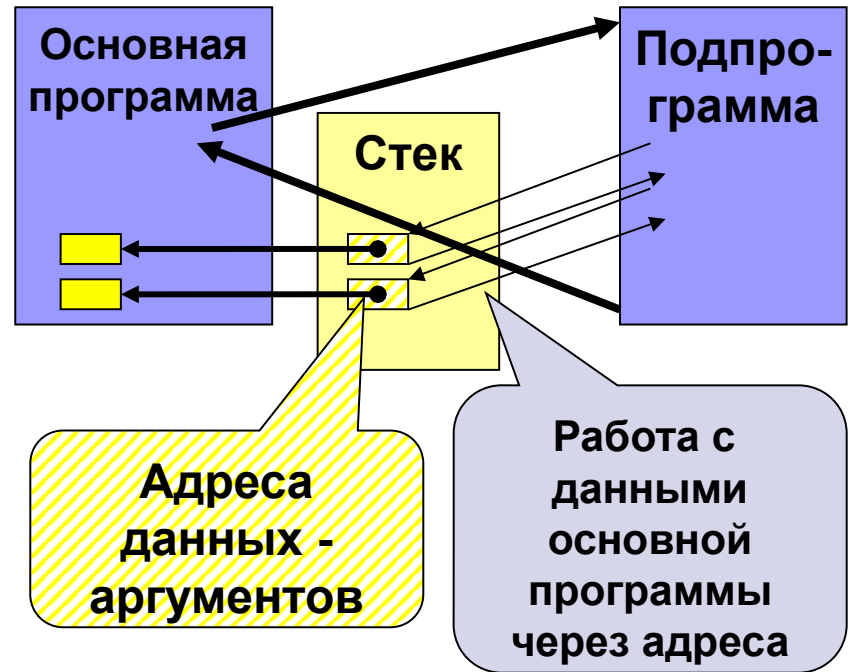
Способы передачи параметров

Передача значения



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

Передача адреса



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

Обозначения способов передачи параметров

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

```
int Beta(float x, int n) {...}
```

```
ВЫЗОВ: k = Beta(z1,i);
```

- **Параметры-адреса** при описании подпрограммы помечаются в зависимости от типа (указатель или ссылка), например:

а) указатель

```
void prog(int a, int *b) { *b = a; }
```

```
ВЫЗОВ: prog(c,&d);
```

б) ссылка

```
void prog(int a, int &b) { b = a; }
```

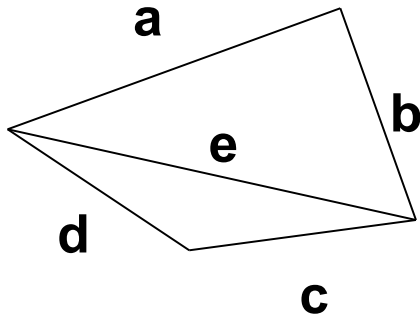
```
ВЫЗОВ: prog(c,d);
```

- **Параметры-константы** при описании подпрограммы помечаются служебным словом `const`, например:

```
int prog(const int *a) {...};
```

```
ВЫЗОВ: k = prog(&c);
```

Определение площади четырехугольника



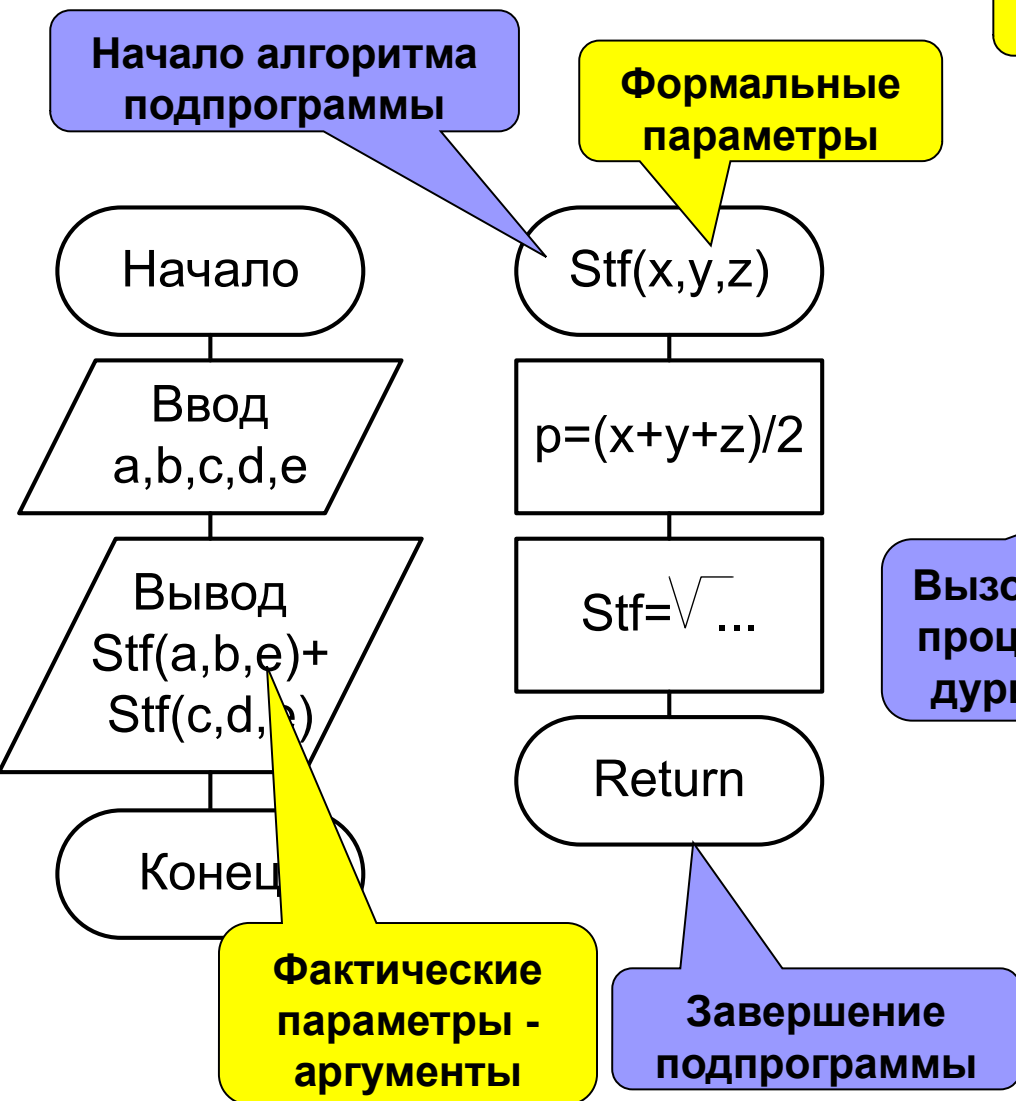
Площадь четырехугольника определяем, как сумму площадей двух треугольников. Площадь треугольника со сторонами a , b , c определяем по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \text{ где } p = \frac{a+b+c}{2}.$$

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

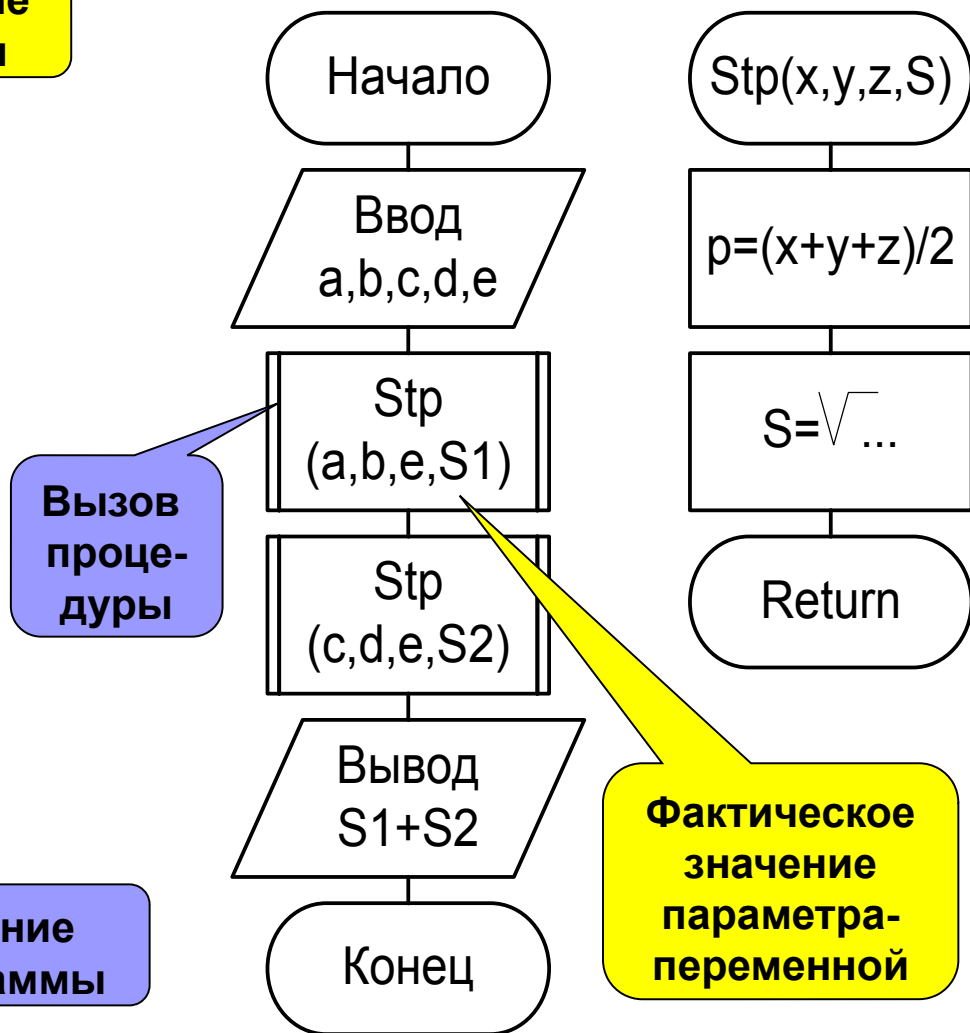
Схемы алгоритмов подпрограмм

Подпрограмма-функция



Подпрограмма-процедура

Формальный параметр-переменная в заголовке на схеме не выделяется



Функция

```
#include <iostream>
```

```
#include <math.h>
```

```
using namespace std;
```

Ex04_01

Тип
возвращаемого
значения

Формальные
параметры

```
float stf(float x, float y, float z) {
```

```
    float p = (x+y+z)/2;
```

Автоматическая
переменная

```
    return sqrt(p*(p-x)*(p-y)*(p-z));
```

Вычисление
возвращаемого
значения

```
}
```

```
int main() {
```

```
    float a,b,c,d,e;
```

Автоматические
переменные

```
    cout << "Enter a,b,c,d,e:";
```

```
    cin >> a >> b >> c >> d >> e;
```

```
    cout << "S = " << stf(a,b,e)+stf(c,d,e) << endl;
```

```
    return 0;
```

Вызов
функции из
выражения

Фактические
параметры

```
}
```

Процедура

Ex04_02

```
#include <iostream>
#include <math.h>
using namespace std;
void stp(float x, float y, float z, float &s) {
    float p = (x+y+z)/2;
    s = sqrt(p*(p-x)*(p-y)*(p-z));
}
int main() {
    float a,b,c,d,e,s1,s2;
    cout << "Enter a,b,c,d,e:";
    cin >> a >> b >> c >> d >> e;
    stp(a,b,e,s1);
    stp(c,d,e,s2);
    cout << "S = " << s1+s2 << endl;
    return 0;
}
```

Возвращаемое
значение

Локальная
переменная

Автоматические
переменные

Вызов
процедуры

Способы возврата значений из подпрограмм

- *как возвращаемое значение функции* – так может вернуть значение только функция и только скалярное значение (в том числе адрес), например:

```
int z(int a, int b) {  
    return a + b;    // возвращаемое значение  
}
```

```
y = z(k, t);          // возвращаемое значение заносим в y
```

- *через параметры-переменные* – и функция, и процедура без ограничений могут вернуть необходимое количество параметров, например:

```
1) void d(int a, int b, int *c) {  
    *c = a + b;        // c вернется через параметр-  
указатель  
}
```

```
d(n, p, &s);          // s будет содержать сумму
```

```
2) int z(int a, int b, int &k) {  
    k = a + b;          // k вернется через параметр-ссылку  
    return a * b;       // возвращаемое значение  
}
```

4.4 Параметры структурных типов

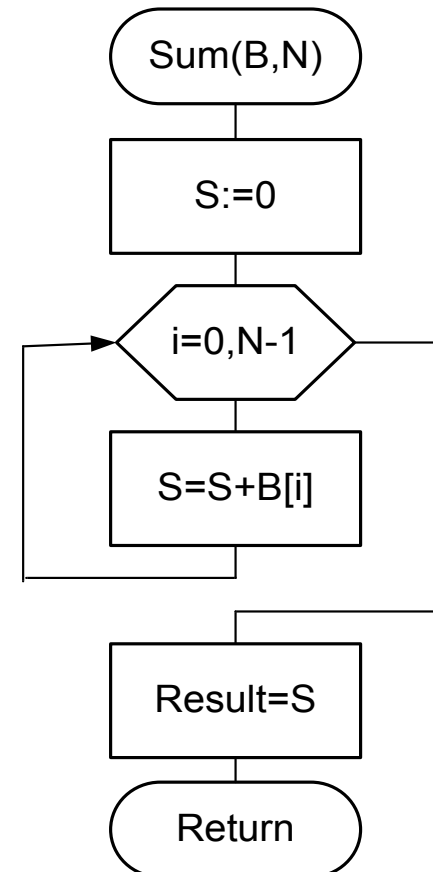
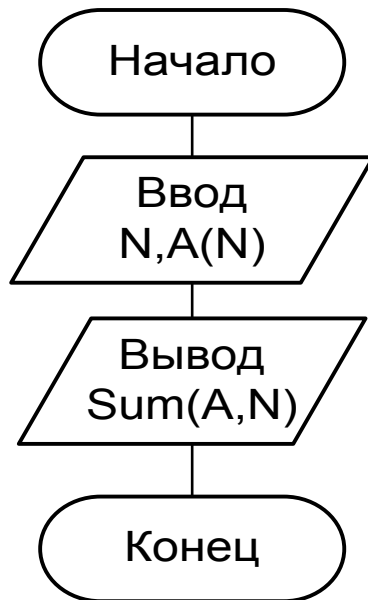
А Параметры-массивы

В C++ отсутствует контроль размера массива по первому индексу!

а) `int x[5] ⇔ int *x ⇔ int x[]`

б) `int y[][8] ⇔ int y[4][8]`

Пример. Функция вычисления суммы элементов массива.



Программа

Ex04_03

Объявление
параметра-
массива

```
#include <iostream>
using namespace std;
float sum(float a[],int n){
    float s = 0.0;
    for (int i = 0; i<n; i++)
        s +=a[i];
    return s;
}

int main() {
    int n;
    cout << "Enter n: ";      cin >> n;
    float m[n];
    cout << "Enter array:" << endl;
    for (int i = 0; i<n; i++)
        cin >> m[i];
    cout << "s = " << sum(m,n) << endl;
    return 0;
}
```

Количество
элементов
массива

Фактический
параметр
структурного типа

Б Параметры-строки

Функции с возвращаемым значением типа «строка» целесообразно писать как процедуры-функции.

Пример. Функция удаления «лишних» пробелов между словами.

Ex04_04

```
char *strdel(const char *source, char *result)
{
    char *ptr;
    strcpy (result, source);
    while ((ptr=strstr(result, " ")) != nullptr)
        strcpy(ptr, ptr+1);
    return result;
}
```

ВЫЗОВЫ: puts (strdel (str, strres)) ; или
strdel (str, strres) ;

В Параметры-структуры

Имя структуры не является указателем на нее.

Пример 1. Сумма элементов массива (указатель).

Ex04_05

```
struct mas{int n; int a[10]; int s;} massiv;
```

```
int summa(struct mas *x)
```

```
{ int i,s=0;
```

```
  for(i=0;i<x->n;i++) s+=x->a[i];
```

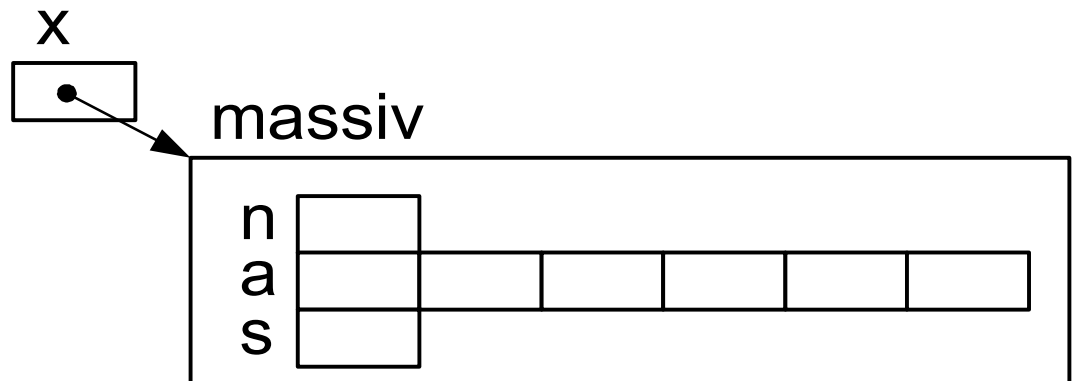
```
  x->s=s;
```

```
  return s;
```

```
}
```

ВЫЗОВ:

```
summa (&massiv) ;
```



Параметры-структуры (2)

Пример 2. Сумма элементов массива (ссылка).

Ex04_05b

```
struct mas{int n; int a[10]; int sum;} massiv;
```

```
int summa(struct mas &x)
```

```
{ int i,s=0;
```

```
  for(i=0;i<x.n;i++) s+=x.a[i];
```

```
  x.s=s;
```

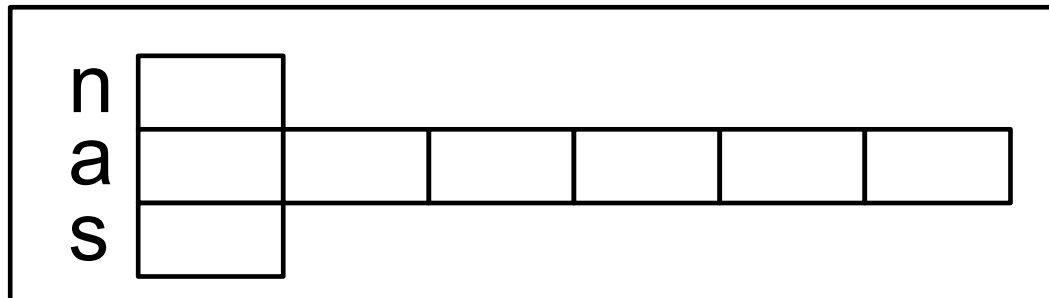
```
  return s;
```

```
}
```

x=massiv

ВЫЗОВ:

```
summa(massiv);
```



Параметры-структуры (3)

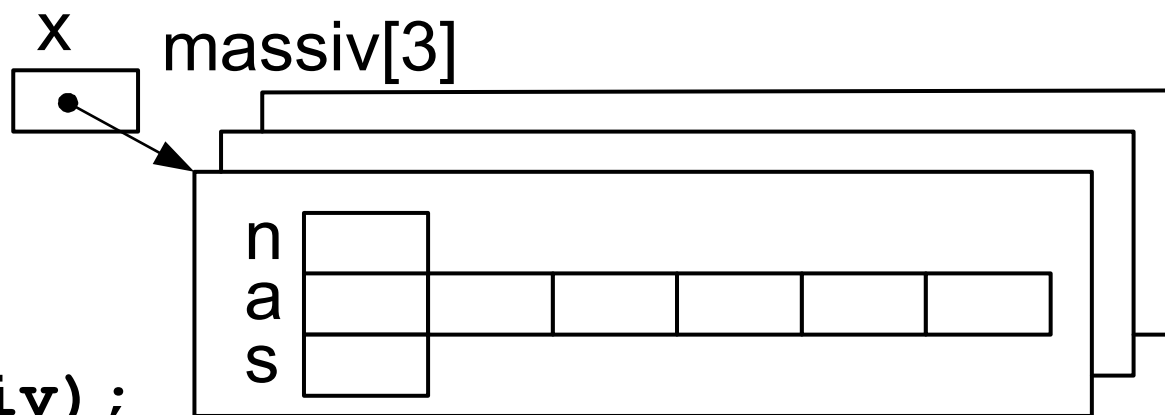
Ex04_05c

Пример 3. Сумма элементов массива (массив структур).

```
struct mas{int n;int a[10];int sum;} massiv[3];

int summa(struct mas *x)
{ int i,k,s,ss=0;
  for(k=0;k<3;k++,x++)
    { for(s=0,i=0;i<x->n;i++) s+=x->a[i];
      x->s=s;
      ss+=s;
    }
  return ss;
}
```

ВЫЗОВ: `summa (massiv);`



4.5 Многофайловые программы C++

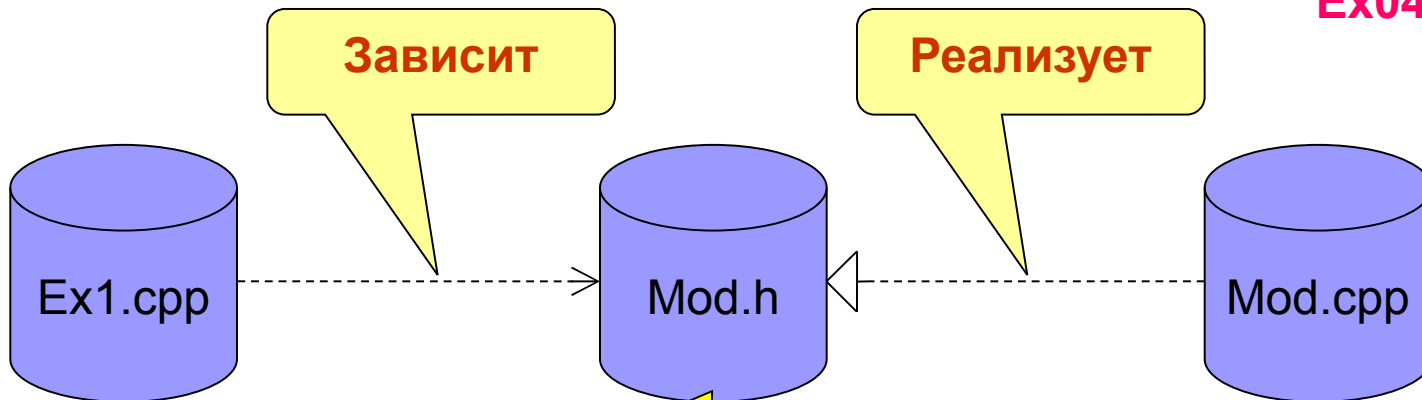
Модуль в современных языках программирования – это автономно компилируемая коллекция программных ресурсов (подпрограмм, переменных, констант, типов и др.).

В C++ до версии C++20 не существовало модулей. Вместо них использовалась **многофайловая структура программы**, что в сочетании с пространствами имен, заголовочными файлами и внешними статическими переменными позволяло **имитировать** модульную организацию программы.

Согласно рекомендациям разработчиков языка **заголовочные файлы** с расширением .h должны включать объявления ресурсов псевдомодуля, используемых в других частях программы, а соответствующие файлы с расширением .cpp - содержать реализацию объявленных в заголовке подпрограмм.

Пример многофайловой программы

Ex04_06



```
#include <stdio.h>
#include "Mod.h"
int main()
{
    int a=18,b=24,c;
    c=nod(a,b);
    printf("nod=%d\n",c);
    return 0;
}
```

```
#ifndef MOD_H
#define MOD_H
int nod(int a,int b);
#endif
```

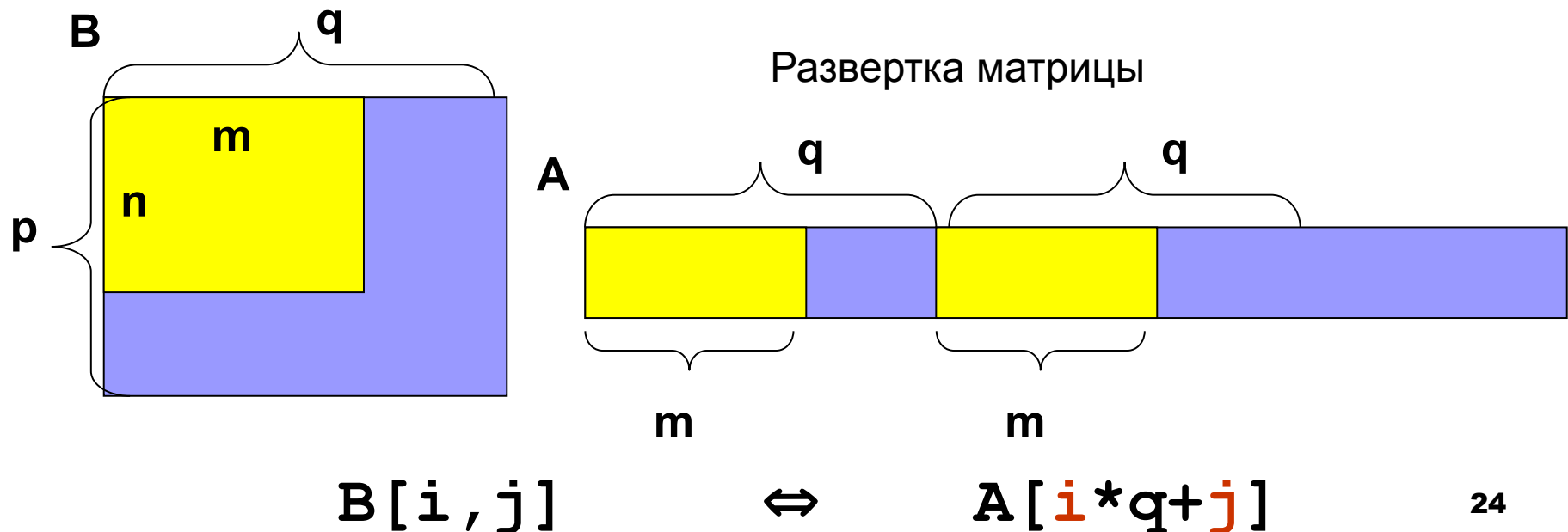
```
#include "Mod.h"
int nod(int a,int b)
{
    while (a!=b)
        if (a>b) a=a-b;
        else b=b-a;
    return a;
}
```

4.6 Создание универсальных подпрограмм

4.6.1 Универсальные подпрограммы с многомерными массивами

Существует две проблемы создания универсальных подпрограмм с параметрами – многомерными массивами:

- 1) компилятор C++ контролирует вторую и последующие размерности многомерных массивов;
- 2) при использовании "развертки" многомерного массива необходимо учитывать (если оно есть) несоответствие максимальной и реальной размерностей массива:



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

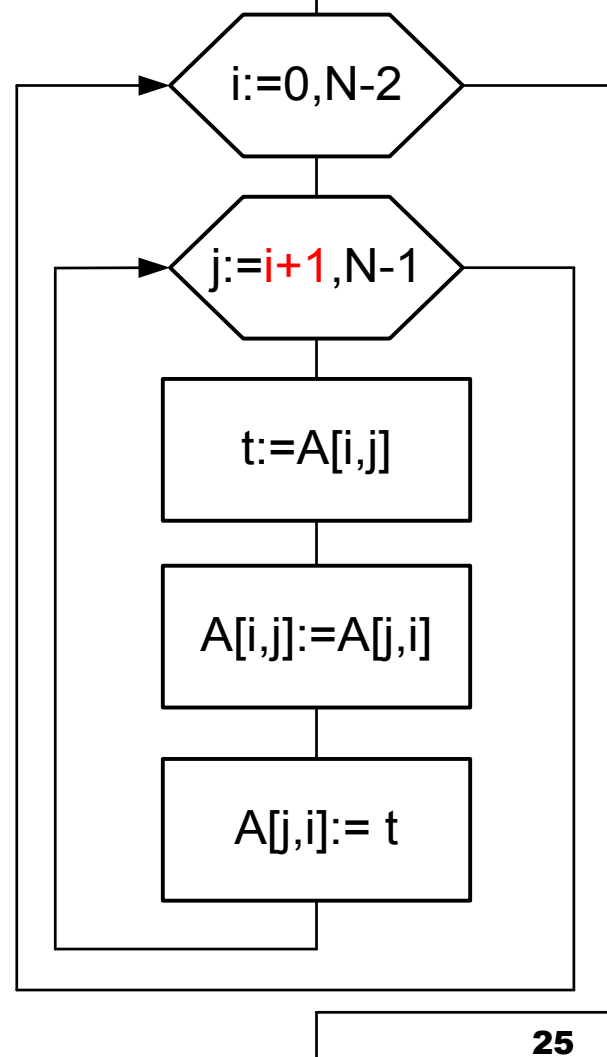
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Если $i=0$, то первый номер столбца $j=1$

$i=1 \Rightarrow j=2$

$i=2 \Rightarrow j=3$

$i=3 \Rightarrow j=4$



Универсальная подпрограмма

Ex04_07

Файл Array.h:

```
#ifndef ARRAY_H
#define ARRAY_H
void trans(float *r,int n,int q);
#endif
```

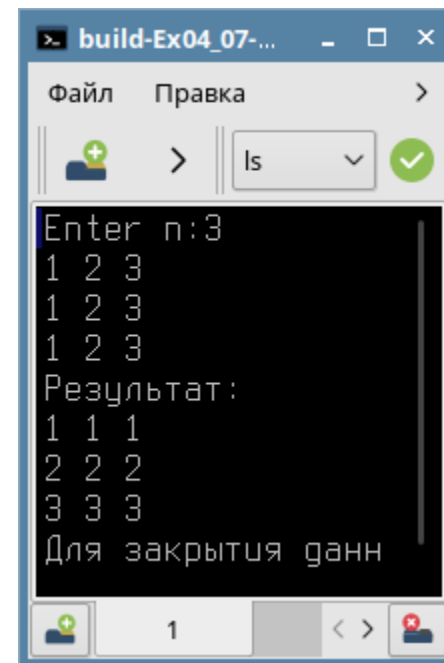
Реальный размер строки

Зарезервированный
размер строки

Файл Array.cpp:

```
#include <Array.h>
void trans(float *r,int n,int q) {
    for (int i=0;i<n-1;i++)
        for (int j=i+1;j<n;j++){
            float t = r[i*q+j];
            r[i*q+j] = r[j*q+i];
            r[j*q+i] = t;
        }
}
```

Развертка матрицы



Подключение файла: #include "Array.h"

Матрица: float a[10][10];

Тип float**

Вызов: trans((float *)a,n,10);

Явное переопределение типа

4.6.2 Универсальные подпрограммы с указателями на функции

Указатель на функцию позволяет работать с адресами функций.

Формат объявления указателя:

Тип_результата (*Имя)(Список_параметров);

Для того, чтобы указателю можно было присвоить адрес функции, должны совпадать *сигнатуры* указателя и функции (типы параметров и типы возвращаемых значений).

Пример:

```
int add(int n,int m) {return n+m;}
```

```
int sub(int n,int m) {return n-m;}
```

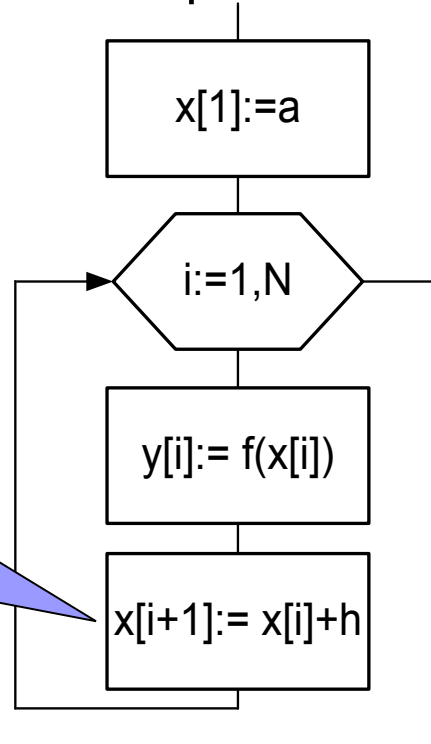
```
int (*ptr) (int,int) ;    // объявление указателя
ptr = add;                // присваивание значения указателю
a = ptr(a,b) ;            // вызов функции через указатель
```

Пример. Табулирование функций

Ex04_08

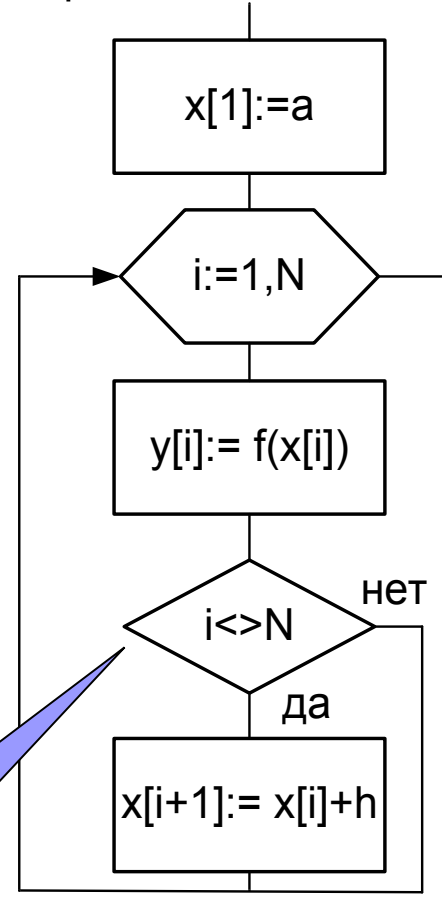
Табулирование – построение таблицы значений:

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

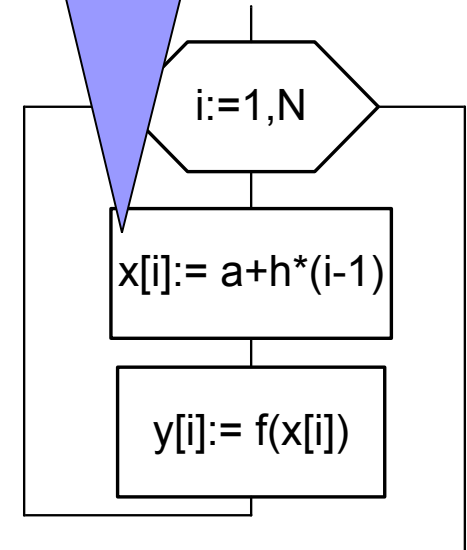
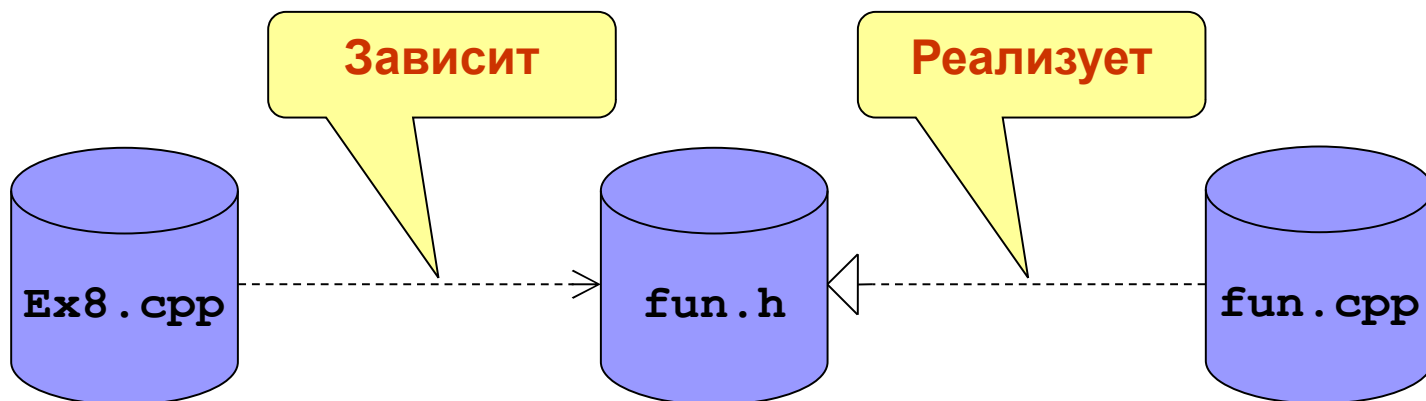


Диаграмма компоновки. Прототип функции с параметром-указателем на функцию



Файл fun.h:

```
#ifndef FUN_H
#define FUN_H
void table(float(*)
(float), float, float, int, float*, float*);
#endif
```

Количество точек

Границы отрезка

Массивы x и y

Указатель на функцию

Подпрограмма табулирования функции

Файл *fun.cpp*:

Ex04_09

```
#include <Fun.h>

void table(float(*ptr)(float), // указатель на функцию
          float a, float b,    // границы отрезка
          int n,               // количество точек
          float *masX, float *masY) // массивы результатов
{
    float h = (b-a)/(n-1);
    float x = a;
    for (int i=0;i<n;i++){
        masX[i] = x;
        masY[i] = ptr(x);
        x += h;
    }
}
```

Тестирующая программа

```
#include <iostream>
using namespace std;
#include <iomanip>
#include "fun.h"
#include <math.h>
```

```
float f1(float x)
{
    return x*x-1;
}
```

Функция 1

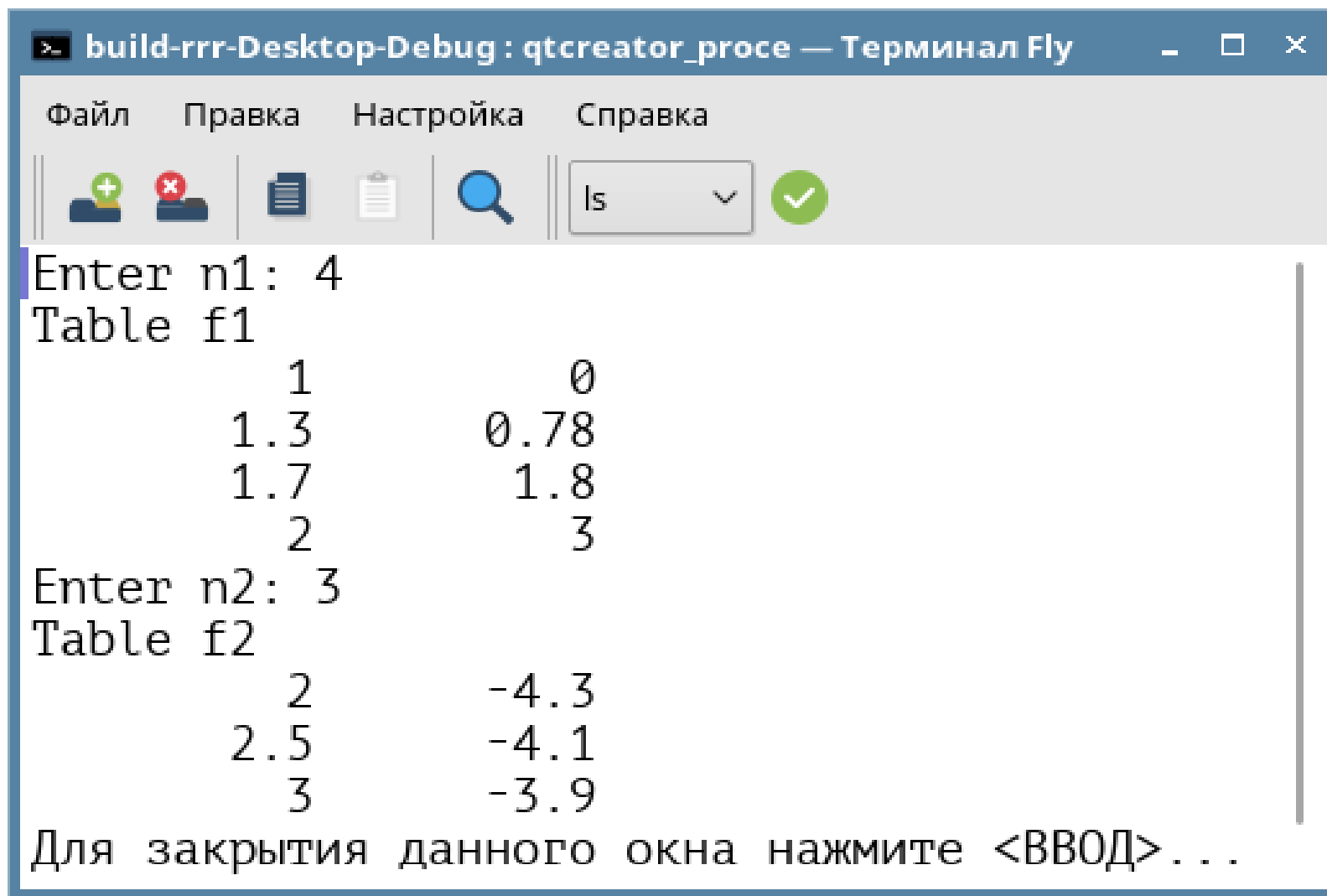
```
float f2(float x)
{
    return log(x)-5;
}
```

Функция 2

Тестирующая программа. Функция main

```
int main() {  
    int n;  
    cout << "Enter n1: ";    cin >> n;  
    float my1[n],mx1[n];  
    table(f1,1,2,n,mx1,my1);  
    cout << "Table f1" << endl;  
    for (int i=0;i<n;i++)  
        cout << setw(10)<< setprecision(2)<< mx1[i] <<  
            setw(10)<< setprecision(2)<< my1[i] << endl;  
    cout << "Enter n2: ";    cin >> n;  
    float my2[n],mx2[n];  
    table(f2,2,3,n,mx2,my2);  
    cout << "Table f2" << endl;  
    for (int i=0;i<n;i++)  
        cout << setw(10)<< setprecision(2)<<mx2[i]<<  
            setw(10)<< setprecision(2)<< my2[i]<< endl;  
    return 0;  
}
```


Результат



The screenshot shows a terminal window titled "build-rrr-Desktop-Debug : qtcreeator_proce — Терминал Fly". The window has a menu bar with "Файл", "Правка", "Настройка", and "Справка". Below the menu is a toolbar with icons for adding/removing files, clipboard, search, and a dropdown menu currently showing "ls" with a green checkmark icon. The terminal output is as follows:

```
Enter n1: 4
Table f1
      1      0
    1.3    0.78
    1.7    1.8
      2      3
Enter n2: 3
Table f2
      2    -4.3
    2.5    -4.1
      3    -3.9
Для закрытия данного окна нажмите <ВВОД>...
```

4.7 Пространство имен

Большинство приложений состоит более чем из одного исходного файла. При этом возникает вероятность дублирования имен, что препятствует сборке программы из частей. Для снятия проблемы в С++ был введен механизм логического разделения области глобальных имен программы, который был назван *пространством имен*.

Имена, определенные в пространстве имен, становятся локальными внутри него и могут использоваться независимо от имен, определенных в других пространствах.

```
namespace [Имя] { Объявления_и_определения }
```

Пример:

```
namespace ALPHA {                               // ALPHA – имя пространства имен
    long double LD;                             // объявление переменной
    float f(float y) { return y*LD; }           // описание функции
}
```

Пространство имен определяет область видимости, следовательно, функции, определенные в пространстве имен, могут без ограничений использовать другие ресурсы, объявленные там же (переменные, типы и т.д.).

Если имя пространства опущено, то считается, что определено пространство имен с именем *unique*, которое не упоминается в программе.

Пример:

```
namespace { int asd;}
```

Доступ к элементам пространства имен

Доступ к элементам **других** пространств имен может осуществляться:

1) с использованием **квалификатора доступа**, например:

ALPHA::LD или **ALPHA::f()**

2) с использованием **объявления using**, которое указывает, что некоторое имя доступно в другом пространстве имен:

```
namespace BETA {  
    ...  
    using ALPHA::LD; /* имя ALPHA::LD доступно в BETA*/ }
```

3) с использованием **директивы using**, которая объявляет все имена одного пространства имен доступными в другом пространстве:

```
namespace BETA {  
    ...  
    using ALPHA; /* все имена ALPHA доступны в BETA*/  
}
```

Пространства имен приложения

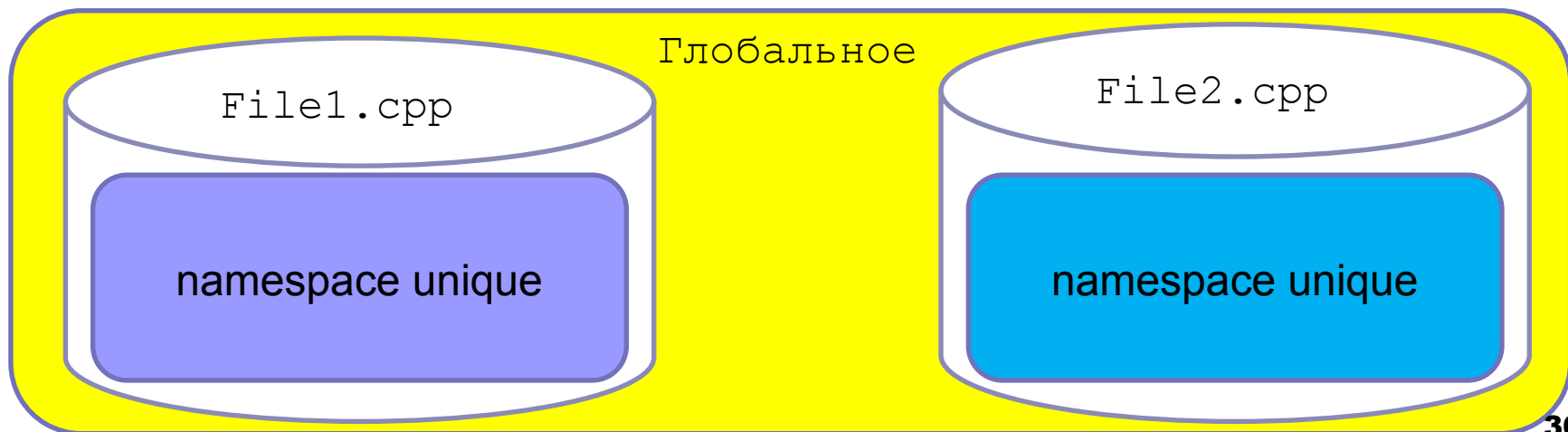
Приложение включает одно непоименованное *глобальное пространство имен*. Имена, входящие в это пространство, объявляются без указания пространства имен. Их видимость определяется классом памяти переменной. При необходимости уточнить ссылку на такое имя указывают "::".

Пространство имен, объявленное без имени, *невидимо в других файлах*:

```
namespace { namespace-body }
```

По умолчанию оно именуется "unique" и доступно в своем файле, т.е.:

```
namespace unique { namespace-body } // не пишем!  
using namespace unique;              // не пишем!
```

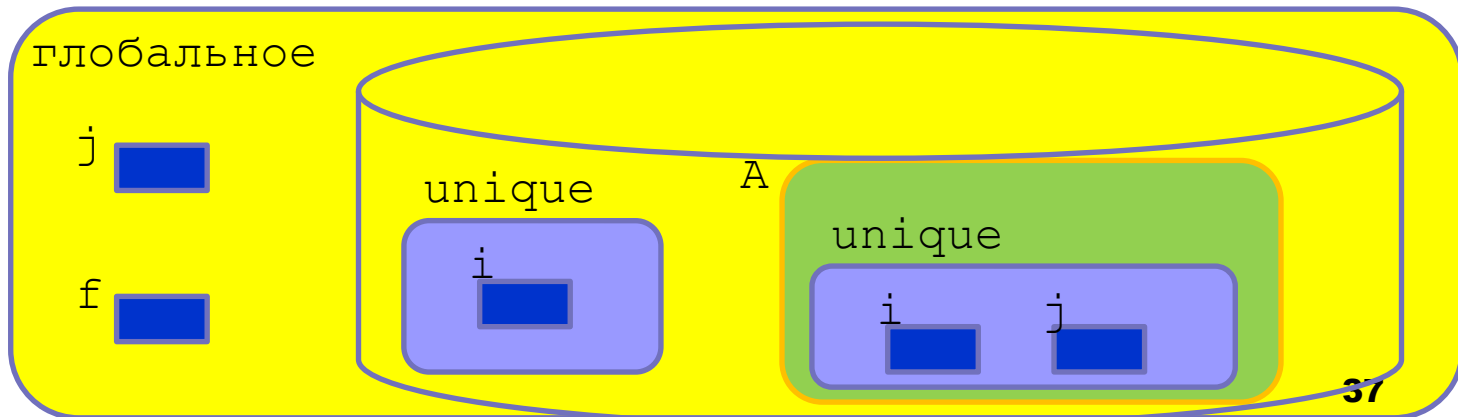


Пример определения пространства имен

```
int j;                // ::j                Ex04_10
namespace { int i; }  // unique::i
    void f() { i++; }  // ::f    unique::i++
namespace A {
    namespace {int i,j;}} // A::unique::i A::unique::j

using namespace A;
void h()
{    i++;              // unique::i или A::unique::i ???????
  A::i++;              // A::unique::i
  j++;                 // A::unique::j или ::j ???????
}
```

using namespace unique;
подразумевается по умолчанию



Использование квалификаторов доступа

Пример:

```
int i; Ex04_11
```

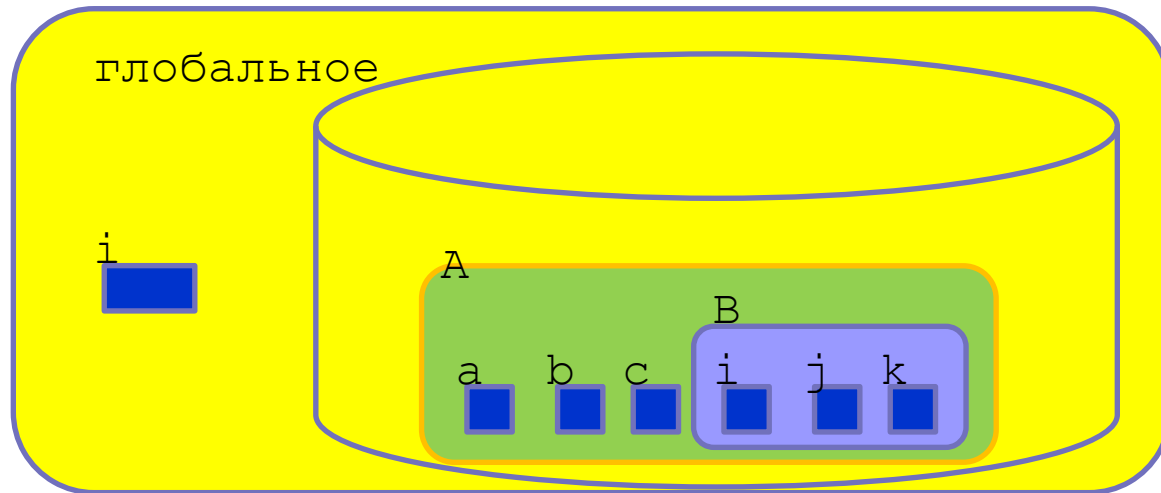
```
namespace A
{
    int a, b, c;
    namespace B {int i, j, k;}
}
```

```
int main()
{
```

```
    A::a++; // обратиться без A:: нельзя, т.к.  
           // отсутствует using
```

```
    A::B::i++;  
    ::i++; // глобальное i
```

```
}
```



Имена стандартных библиотек C++

Согласно стандарту ANSI/ISO в C++ все имена ресурсов стандартных библиотек определены в пространстве std. При использовании этого пространства автоматически подключаются библиотеки <cstdio>, <cmath> и т.д.

Пример:

1-й вариант

```
#include <iostream>
int main()
{
    std::cout << "Hello ";
}
```

2-й вариант

```
#include <iostream>
int main()
{
    using namespace std;
    cout << "World." << endl;
}
```

Однако можно по-прежнему использовать определение ресурсов стандартных библиотек в глобальном пространстве. Для этого необходимо подключать <stdio.h>, <conio.h>, <math.h> и т.д. (кроме <iostream.h>, которая больше не существует).

Список доступных стандартных библиотек в старой и новой формах можно посмотреть в среде.

4.8 Аргументы командной строки

Командная строка – текстовый интерфейс, обеспечивающий связь между пользователем компьютера и операционной системой Windows, например вызов программы записывается как:

```
C:\> E:\ivv\proq.exe a1.dat 36 vvv.txt
```

Текущий
каталог

Каталог
программы

Имя
программы

Три параметра,
записанных через пробел

Описание основной программы (функции) С или С++:

```
int main(int argc, char *argv[]) { ... }
```

Массив текстовых строк, через
который передаются параметры

Применительно к примеру командной строки параметры содержат:

argc - количество параметров командной строки +1 = 4;

argv[0] – полное имя файла программы: "E:\ivv\proq.exe";

argv[1] - первый параметр из командной строки – "a1.dat";

argv[2] - второй параметр из командной строки – "36";

argv[3] - третий параметр из командной строки – "vvv.txt";

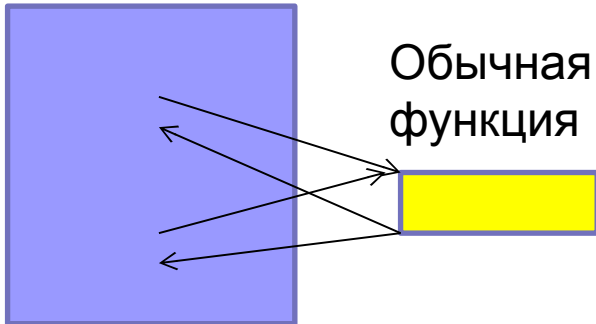
argv[4] - содержит **NULL**.

4.9 Подставляемые функции

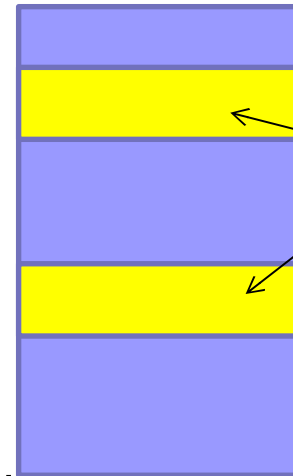
```
inline int abs(int a) {return a>0?a:-a;}
```

Текст подставляемой функции при компиляции вставляется в текст программы в точку вызова столько раз, сколько функция вызывается.

Основная
программа



Основная
программа



Подставляемая
функция

Нельзя "подставлять" функции, содержащие:
циклы и ассемблерные вставки, а также виртуальные методы.

Достоинство: уменьшается время вызова подпрограммы.

Недостаток: увеличивается объем программы;

4.10 Параметрическая перегрузка функций

Параметрическая перегрузка функций — механизм, позволяющий описывать несколько функций с одинаковыми именами, но **разными списками параметров**, например:

```
int lenght(int x,int y){return sqrt(x*x+y*y);}  
int lenght(int x,int y,int z)  
    {return sqrt(x*x+y*y+z*z);}  
int lenght(char *s)  
    {return charwidth*strlen(s);}
```

Разными могут быть количество параметров и/или их типы, тип возвращаемого значения не учитывается

Какую функцию вызвать компилятор определяет по типам и количеству аргументов, например:

```
int a=5,b=3;  
k=length(a,b); // будет вызвана функция с двумя целочисленными  
               // параметрами, т.е. первая из перечисленных выше
```

4.11 Параметры функций по умолчанию

Параметры функции, принимаемые по умолчанию – механизм, позволяющий описать параметры функции с наиболее часто встречающимися значениями аргументов, например:

```
void InitWindow(char *windowname,  
                int xSize=80, int ySize=25,  
                int barColor=BLUE,  
                int frameColor=CYAN) { ... }
```

При вызове функции параметры со значениями по умолчанию можно не указывать, например:

```
InitWindow(pname, 20, 10); // barColor=BLUE,  
                           // frameColor=CYAN  
                           // по умолчанию
```

Пропускать аргументы при вызове нельзя, поэтому часто изменяемые параметры при объявлении функции указывают в начале списка параметров.

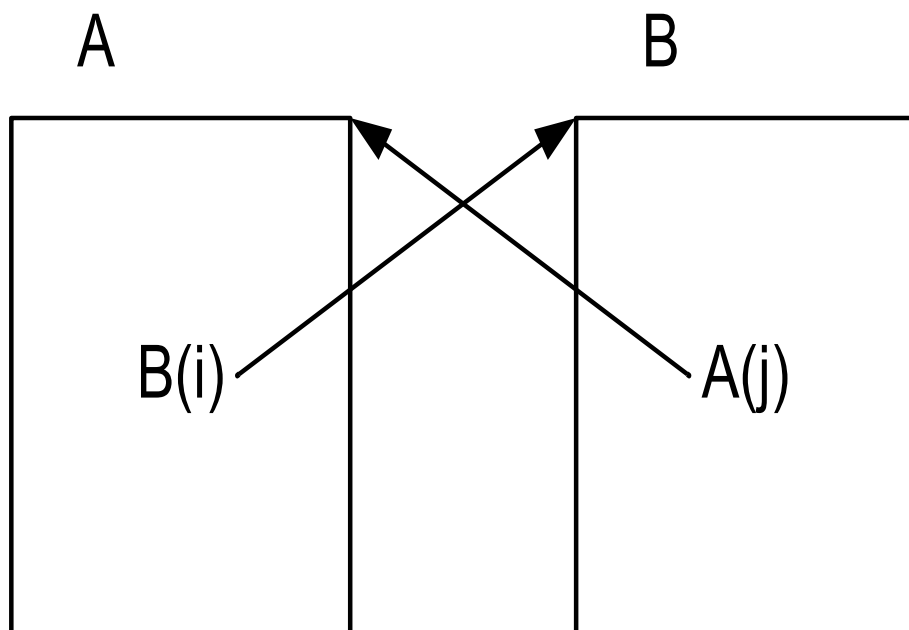
4.12 Рекурсия

4.12.1 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 обязательного использования прототипа:

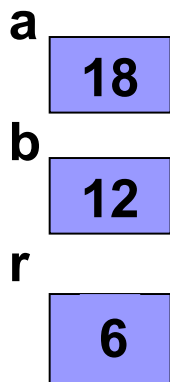
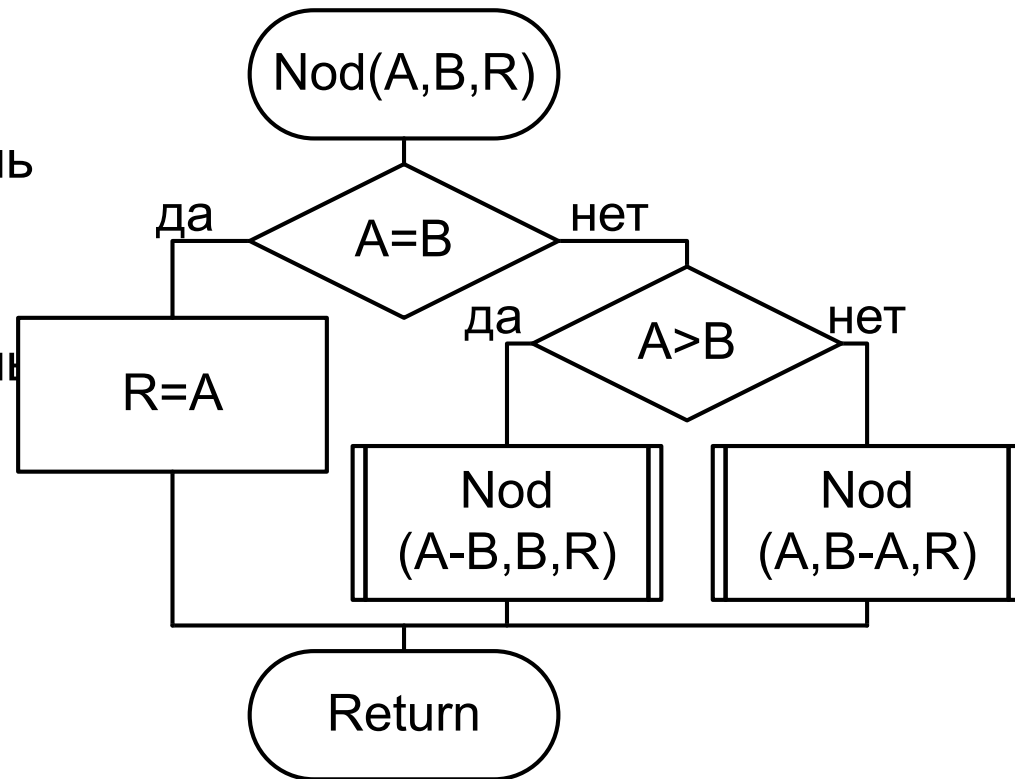


```
void B(int j);  
  
void A(int j){  
    .B(i); ...  
}  
  
void B(int i){  
    . A(j); ...  
}
```

Вычисление NOD. Рекурсив. процедура (схема)

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.

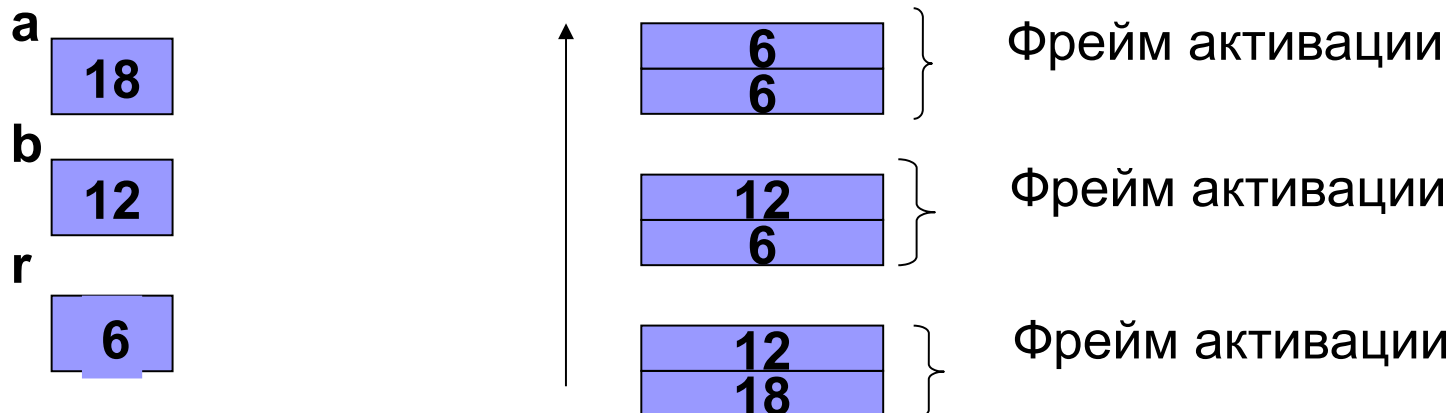
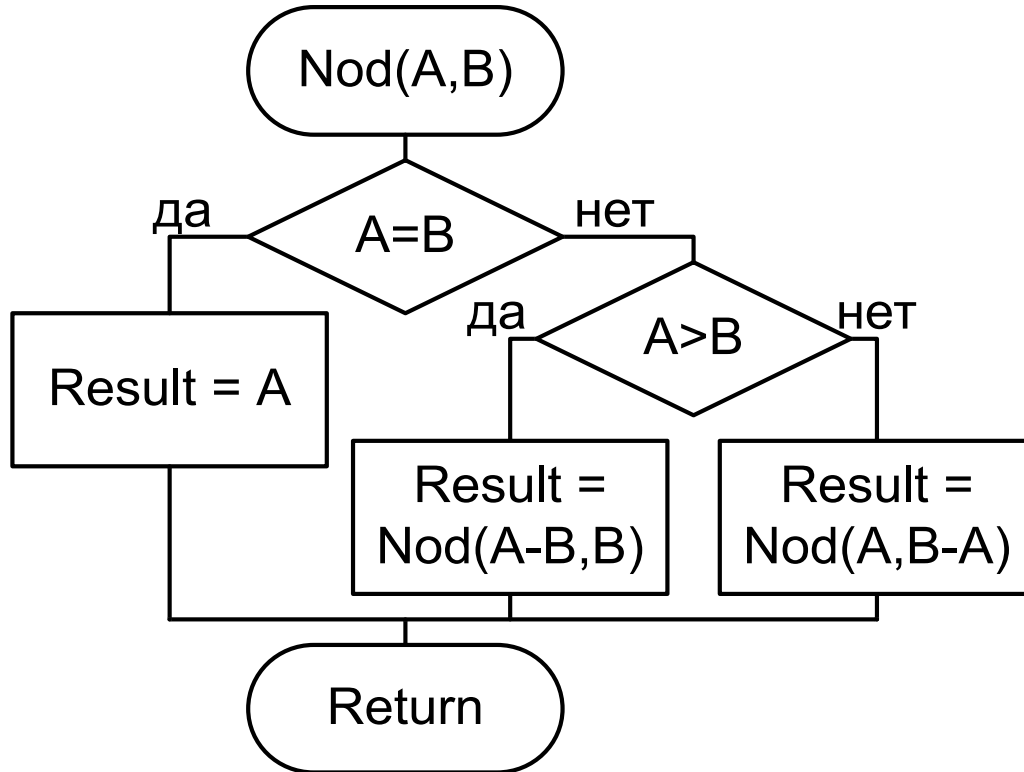


Вычисление NOD. Рекурсивная процедура

Ex04_12

```
#include <iostream>
using namespace std;
void nod(int a,int b,int *r){
    if (a==b) *r = a; // нерекурсивная ветвь
    else
        if (a>b) nod(a-b,b,r);
        else nod(a,b-a,r);
}
int main() {
    int a,b,r;
    cout << "Enter a,b: ";
    cin >> a >> b;
    nod(a,b,&r);
    cout << "nod = " << r << endl;
    return 0;
}
```

Вычисление NOD. Рекурсивная функция (схема)



Вычисление NOD. Рекурсивная функция

Ex04_13

```
#include <iostream>
using namespace std;
int nod(int a,int b){
    if (a==b) return a;  // нерекурсивная ветвь
    else
        if (a>b) return nod(a-b,b) ;
        else return nod(a,b-a) ;
}
int main() {
    int a,b;
    cout << "Enter a,b: ";
    cin >> a >> b;
    cout << "nod = " << nod(a,b) << endl;
    return 0;
}
```


4.12.2 Фрейм активации.

Структура рекурсивной подпрограммы

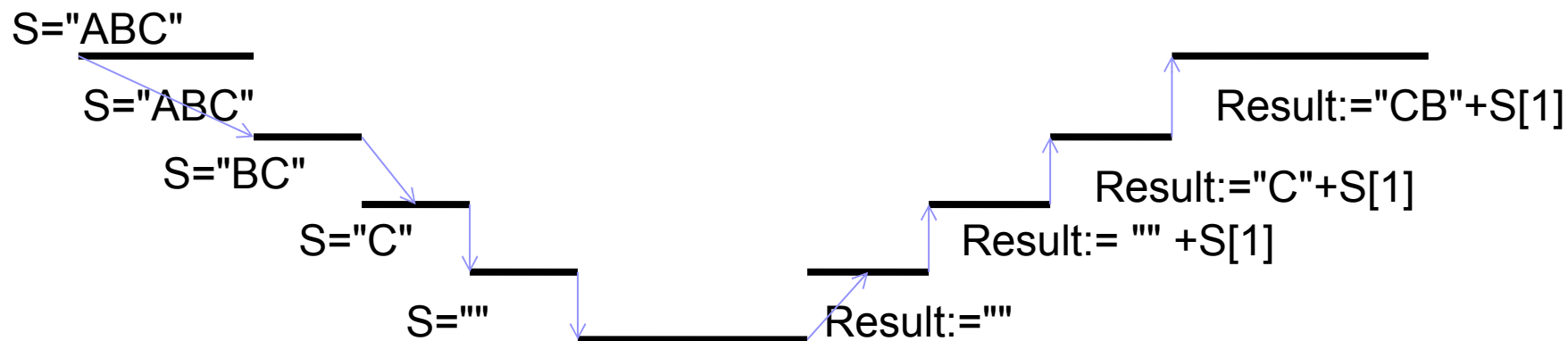
Каждое обращение к рекурсивной подпрограмме вызывает независимую *активацию* этой подпрограммы.

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 последовательным отсечением начального элемента и добавлением его в конец результирующей строки



Переворот строки отсечением первого символа

Ex04_14

```
#include <stdio.h>
#include <string.h>
void reverser(const char s[],char sr[]) {
    int k;
    if (!strlen(s))    sr[0]='\0';
    else {
        reverser(s+1,sr);
        k=strlen(sr);
        sr[k]=s[0]; sr[k+1]='\0'; }
    }
int main() {
    char s[256],sr[256];
    printf("Enter string: ");    scanf("%s",s);
    reverser(s,sr);
    printf("Output string: %s\n",sr);
    return 0;
}
```

A	S	D	B	\0
---	---	---	---	----

\0

B	\0
---	----

B	D	\0
---	---	----

B	D	S	\0
---	---	---	----

B	D	S	A	\0
---	---	---	---	----

Фрейм активации: $V=4 + 4 + 4 + \text{<служебная область>} \approx 24$.

+ нужна строка результата того же размера, что и исходная

Переворот строки перестановкой элементов

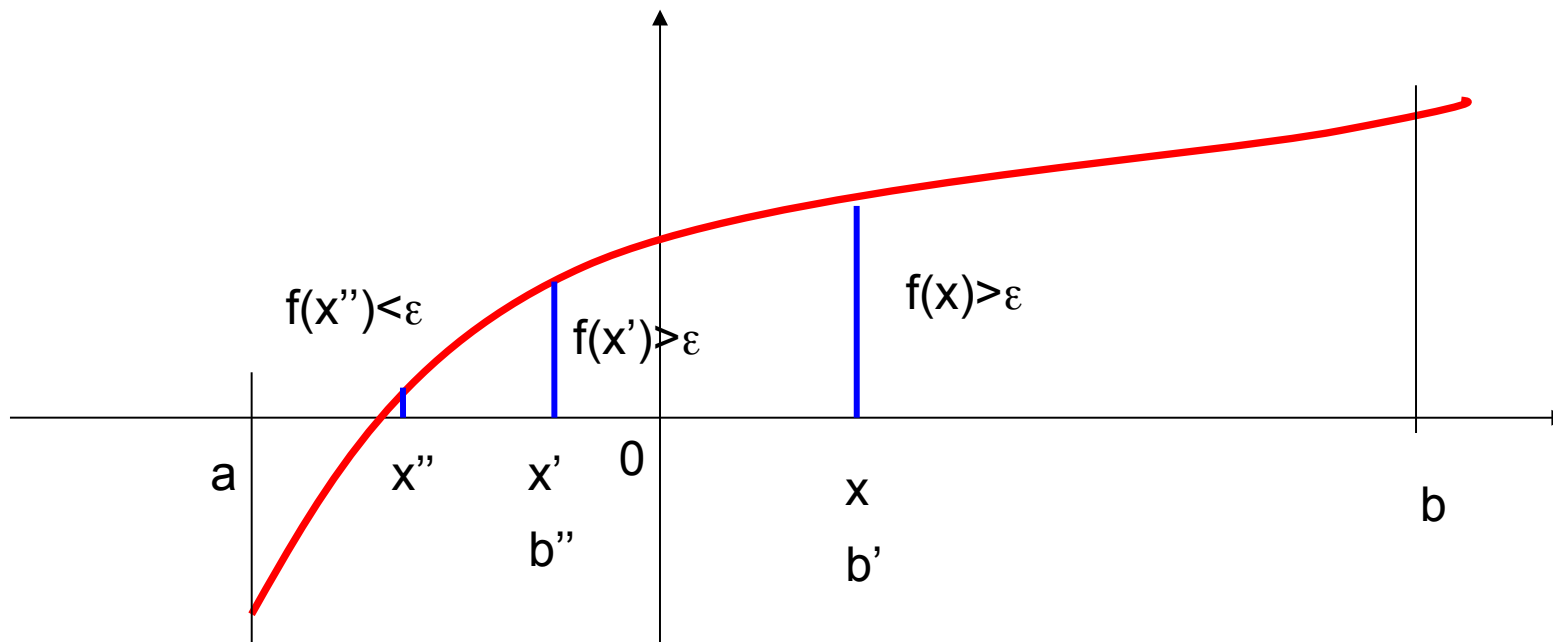
ABCDE \Rightarrow EBCDA \Rightarrow EDCBA

Ex04_15

```
#include <stdio.h>
#include <string.h>
void reverser2(char s[],int n) {
    if (n<strlen(s)/2) {
        char temp = s[n];
        s[n] = s[strlen(s)-n-1];
        s[strlen(s)-n-1] = temp;
        reverser2(s,n+1);
    }
}
int main() {
    char s[20];
    printf("Enter string: ");      scanf("%s",s);
    reverser2(s,0);
    printf("Output string: %s\n",s);
    return 0;
}
```

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

Определение корней уравнения на заданном отрезке. Метод деления пополам

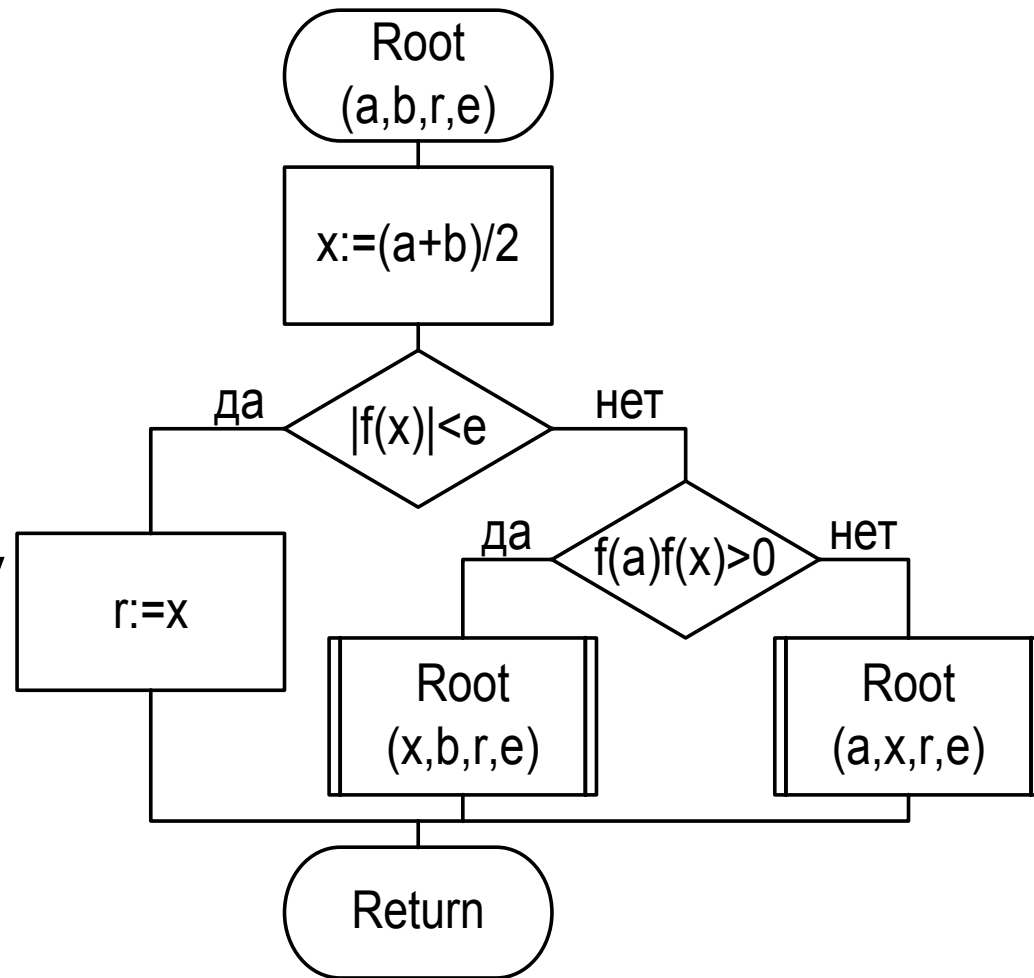


$$x = (b-a)/2$$

Определение корней уравнения на отрезке. Схема

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)

```
#include <iostream>
using namespace std;
#include <math.h>

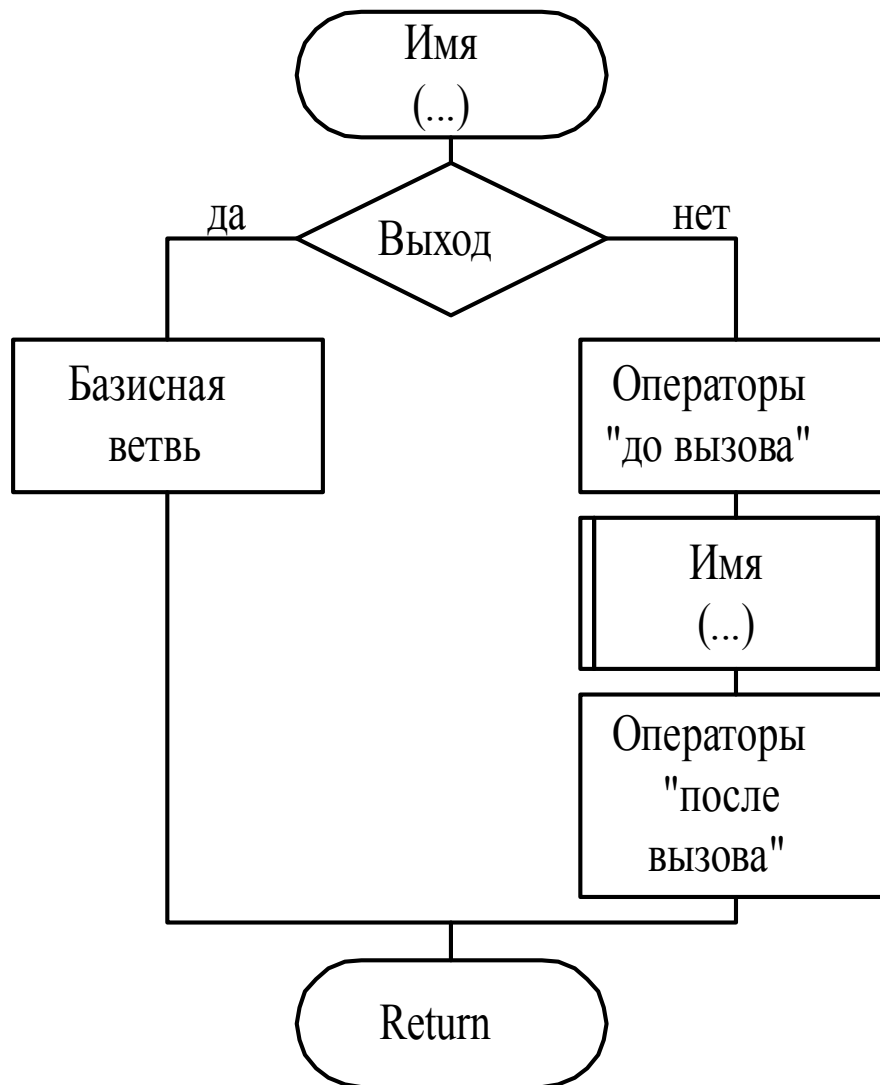
void root(float a, float b, float eps, float &r) {
    float x = (a+b)/2;    float f = x*x-1;
    if (fabs(f)>= eps)
        if ((a*a-1)*f>0) root(x,b,eps,r);
        else root(a,x,eps,r);
    else r = x;
}

int main() {
    float a,b,r,eps;
    cout << "Enter a,b,eps";    cin >> a >> b >>
    eps;
    root(a,b,eps,r);
    cout << "r = " << r << endl;
    return 0;
}
```

Ex04_16

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.

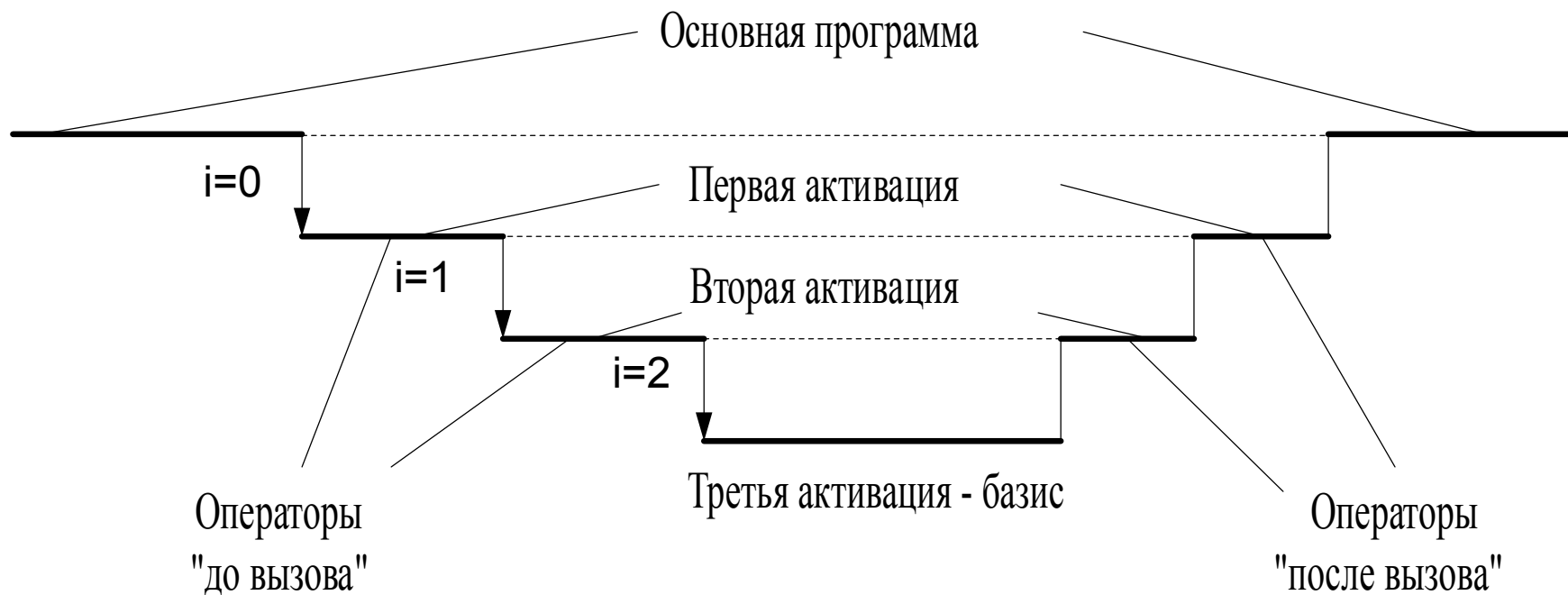
Просмотр массива

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

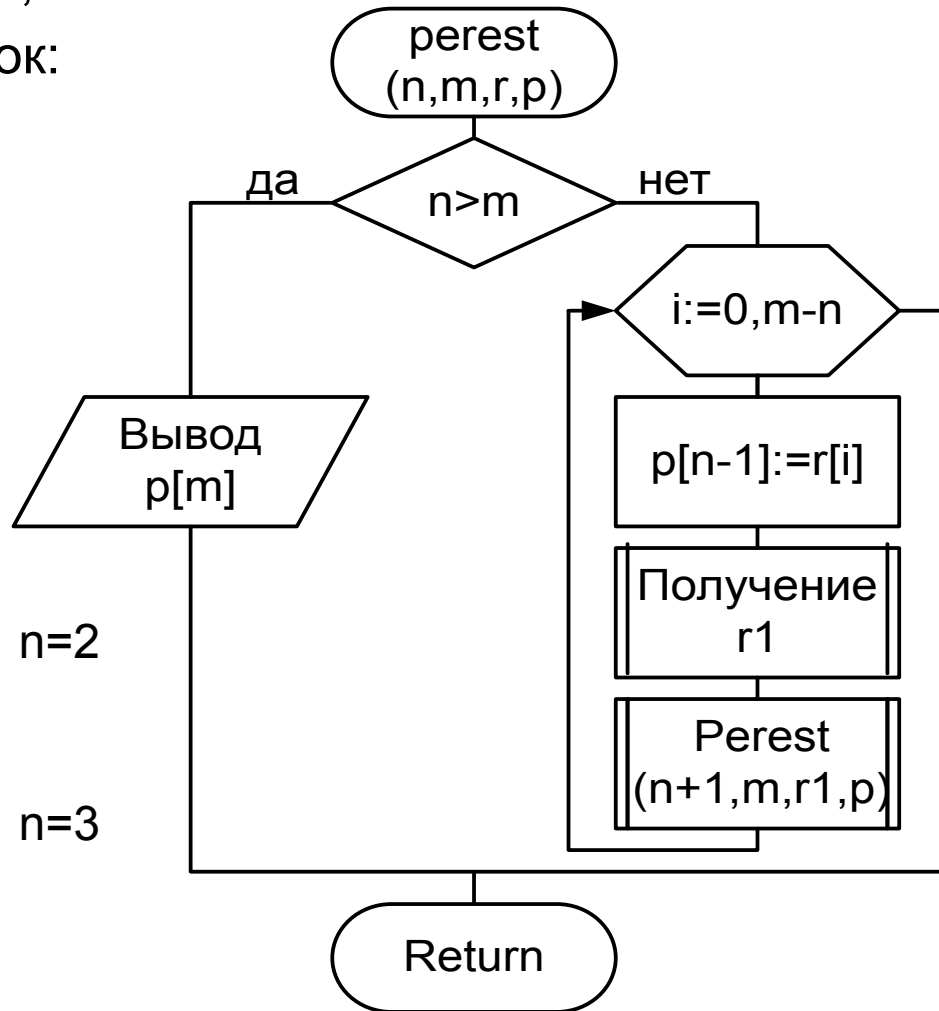
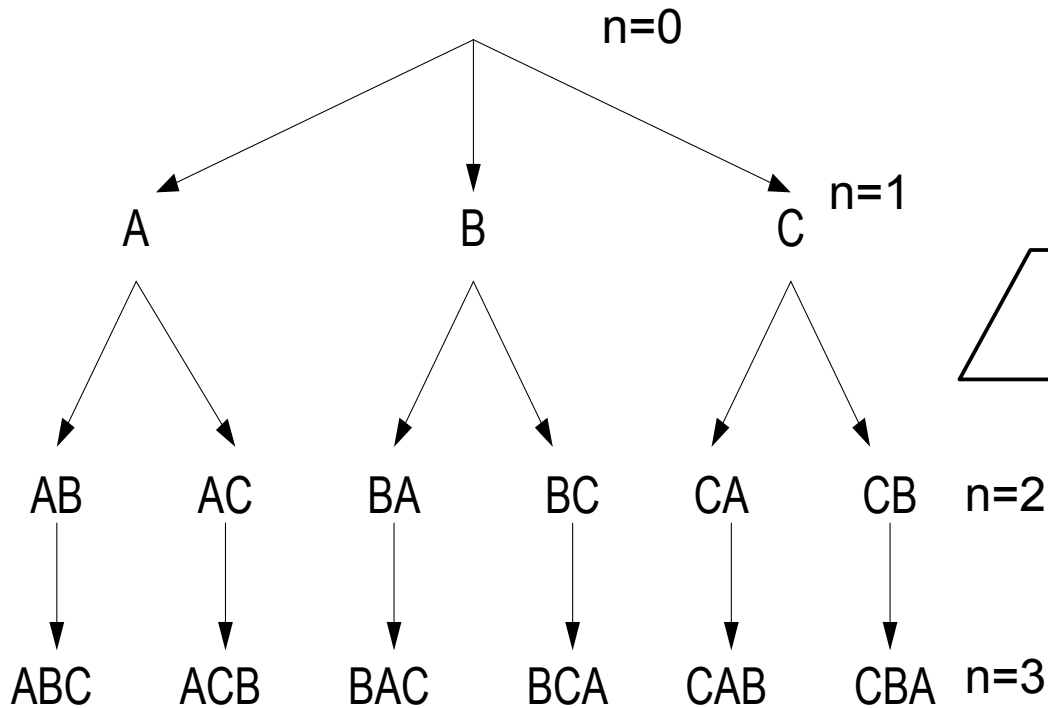
Ex04_17

```
#include <iostream>
using namespace std;
#include <math.h>
void print(const float *x,int i) {
    if (x[i] == 0) cout << "***";
    else {
        if (x[i]>0) cout << x[i];
        print(x,++i);
        if (x[i]<0) cout << x[i];
    }
}
int main() {
    float x[20];      int i = 0;
    do {
        cin >> x[i++];
    } while (x[i-1] != 0);
    print(x,0);
    return 0;
}
```

4.12.3 Древовидная рекурсия. Перестановки

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

Схема формирования перестановок:



Пусть:

m – количество символов в перестановке;

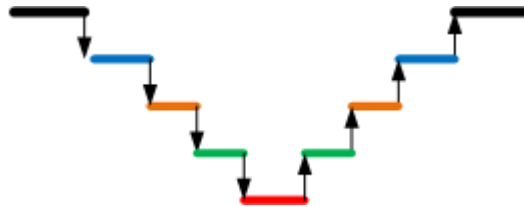
n – номер уровня дерева формирования перестановок

Временная диаграмма работы программы с древовидной рекурсией

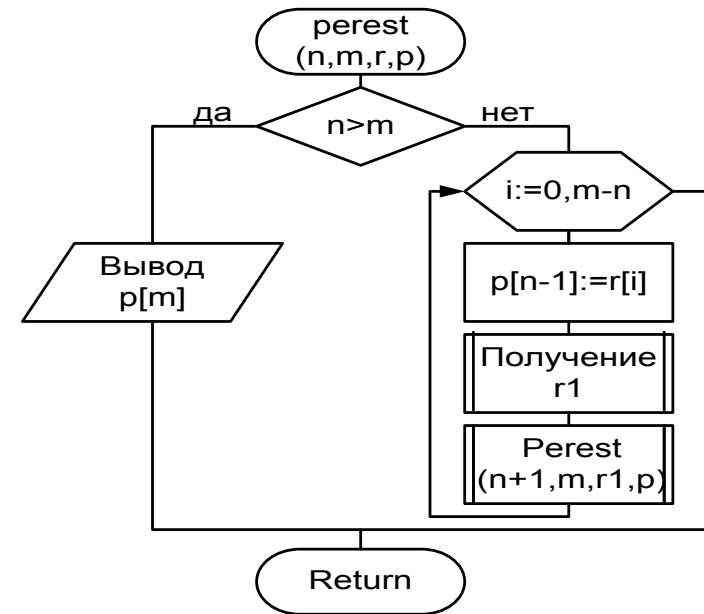
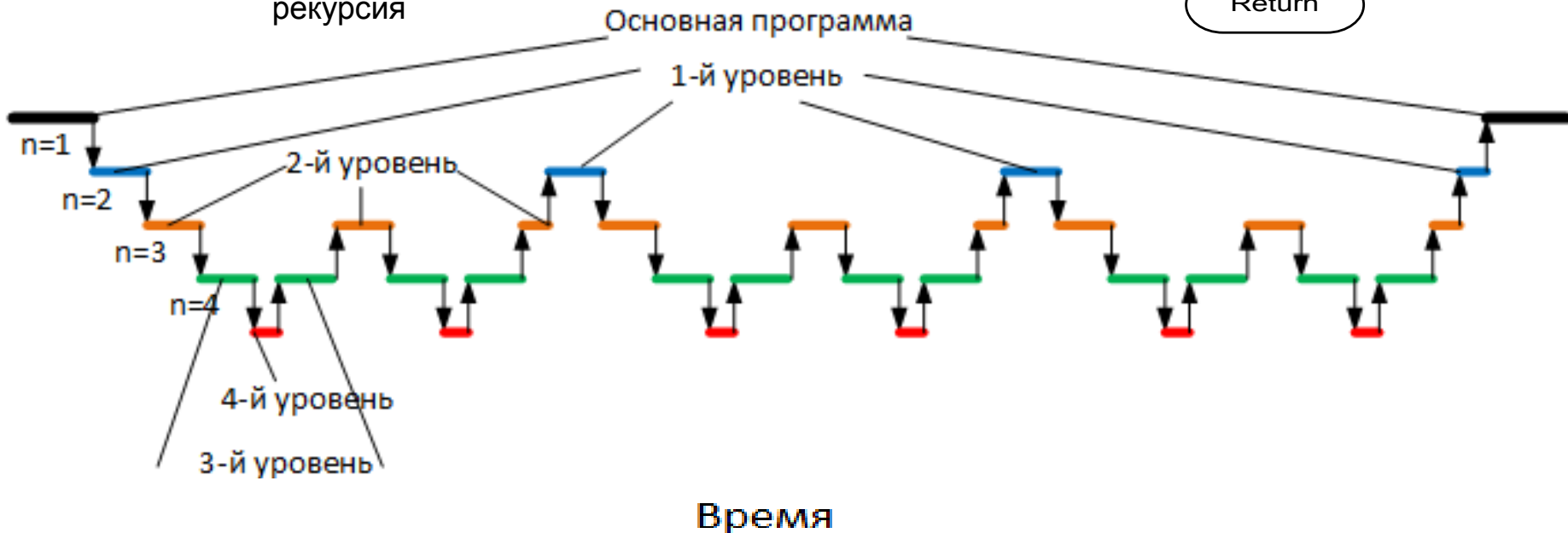
Обычная
подпрограмма



Линейная
рекурсия



Древовидная
рекурсия



Программа Перестановки (начало)

Ex04_18

```
#include <iostream>
using namespace std;
#include <math.h>
void perest(int n,int m,const char *r,char *pole) {
    if (n>m) {
        for (int i=0;i<m;i++)
            cout << pole[i];
        cout <<endl;
    }
    else
```

Программа Перестановки (конец)

```
        for(int i = 0;i<m-n+1;i++){
            pole[n-1] = r[i];
            int k = 0;
            char r1[m];
            for (int j = 0;j<m-n+1;j++)
                if (j != i){
                    r1[k++] = r[j];
                }
            perest(n+1,m,r1,pole);
        }
    }

int main() {
    char a[] = "ABC",pole[3];
    perest(1,3,a,pole);
    return 0;
}
```