

# Глава 4 Средства модульного программирования

С увеличением объема и сложности программ, появилось большое количество задач, выполнение которых повторяется как внутри одной программы, так и в других программах.

Для улучшения эффективности программ, в языках высокого уровня были разработаны средства модульного программирования, предусматривающие использование подпрограмм.

**Подпрограмма – это относительно самостоятельный фрагмент алгоритма, соответствующим образом оформленный и снабженный именем.**

В зависимости от способа описания и вызова, известны подпрограммы двух видов **процедуры и функции.**

Процедуры предназначены для выполнения некоторых действий (например, печать строки), а функция – позволяет получить некоторую величину, которую возвращает в качестве результата.

Однако, принципы программирования C++ основаны на понятии функции. Поэтому, в C++ нет процедур, как элементов языка, однако средства языка позволяют создавать функции, которые не возвращают значения и реализуют конструкцию, аналогичную процедурам.

## 4.1 Функции C++.

При программировании на C++ функция – это основное понятие.

1. Каждая программа обязательно должна включать единственную функцию с именем **main** (главная функция).
2. В программу может входить произвольное количество функций, выполнение которых прямо или косвенно инициируется функцией **main**.
3. Для доступности в программе, функция должна быть в ней определена или описана до первого вызова.
4. В определении функции указывается последовательность действий, выполняемых при ее вызове, имя функции, тип функции (тип возвращаемого ею результата ) и, если необходимо, список параметров (для обмена данными между подпрограммами).

Таким образом, для использования функций необходимо знать, как их можно определять, как к ним обращаться и как устанавливать связь между функцией и программой, ее вызывающей.

## 4.1.1 Описание функции

```
<Тип результата> <Имя> ([<Список параметров>])  
{ [< Объявление локальных переменных и констант >]  
  <Операторы>  
}
```

Пример:

```
int max(int a, int b);  
int max(int a, int b)  
{ if (a>b) return a;  
  else return b;  
}
```

Объявление функции-  
прототип

Описание  
функции

Заголовок  
функции

Тело функции

По правилам C++ подпрограмму можно описывать в любом месте программы и даже в другом файле, но только не внутри другой функции.

При описании функции после функции main или другой функции, в которой она используется, необходимо в начале программы описать прототип этой функции или подключить файл с описанием прототипа.

## 4.1.2 Передача данных в подпрограмму

Подпрограмма может получать данные двумя способами:

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

Неявная передача:

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

Локальные данные

Обращение к локальной переменной

Обращение к глобальной переменной

Глобальные переменные

Обращение к глобальной переменной

Прототип функции max

Обращение к функции max

Обращение к глобальным переменным

Перекрывает глобальную

Обращение к функции sum

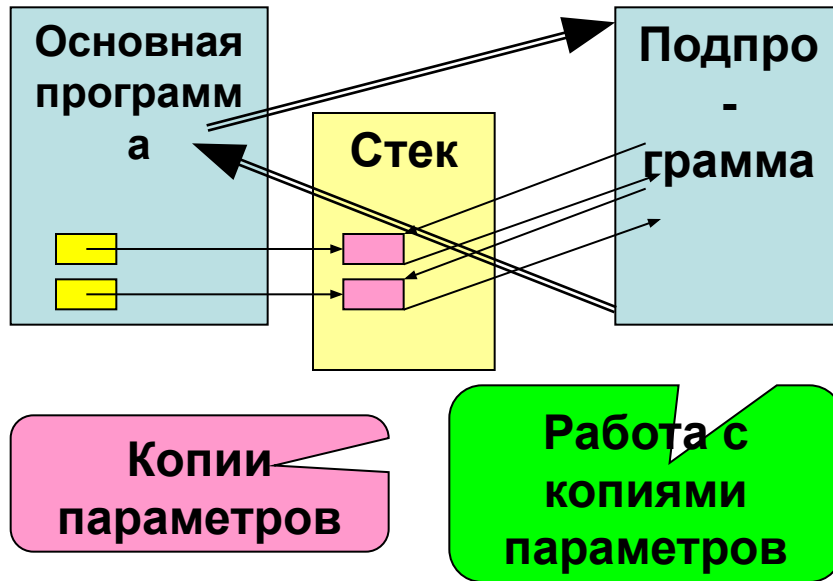
```
int a,c,k;  
int sum(int a,int b)  
{int k,l,j;  
...  
k=c+a;  
}
```

```
int max(int,int);  
  
void main()  
{int i,j,p;  
...  
p=max(k,c);  
}
```

```
int maxl(int b,int c);  
{int a;  
...  
a=sum(c,b);  
k=a;  
}
```

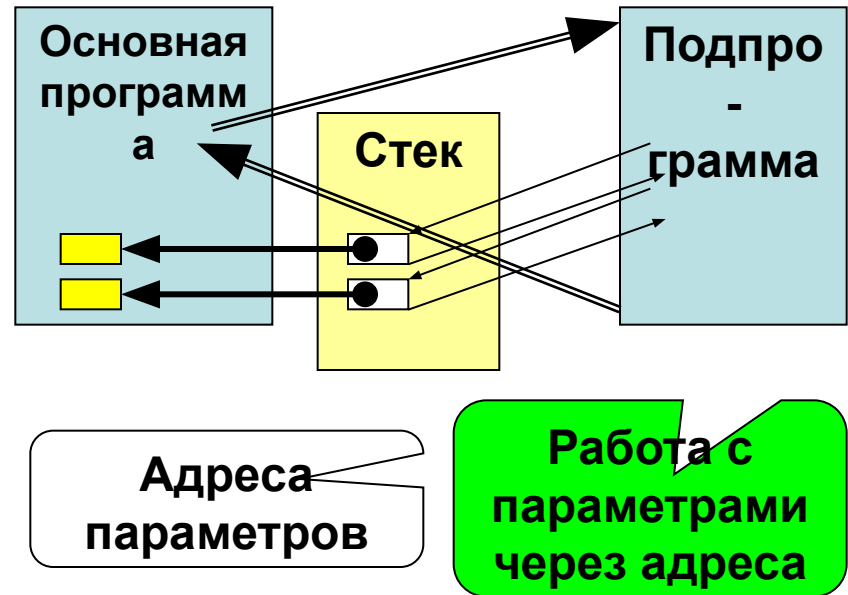
# 4.1.3 Способы передачи параметров

## Передача по значению



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

## Передача по ссылке



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

## 4.1.4 Формальные и фактические параметры

**Формальными** называются параметры, определенные в заголовке функции при ее описании .

Каждый формальный параметр не только перечисляется (именуется), но и специфицируется (для него задается тип) .

Совокупность формальных параметров определяет *сигнатуру функции*.

*Сигнатура функции* зависит от количества параметров, их типа и порядка размещения в спецификации формальных параметров.

Спецификация формальных параметров это либо пусто, либо void либо список отдельных параметров.

**Примеры:**

```
float max(float a, float b){....}
```

```
int fun1()
```

```
{.....}
```

```
char F2(void)
```

```
{.....}
```

**Формальные  
параметры a b**

**Формальные параметры  
отсутствуют(пусто)**

**Формальные параметры  
отсутствуют**

# Формальные и фактические параметры(2)

**Фактическими** называются параметры, задаваемые при вызове функции.

Формальные и фактические параметры должны совпадать:

- по количеству;
- по типу;
- по порядку следования.

Однако, имена формальных и фактических параметров **могут** не совпадать.

**Пример:**

```
int k,l,n=6; float d=567.5,m=90.45
```

Формальные  
параметры

```
void fun2(int a,float c,float b){....} // описание функции fun2
```

```
fun2(n,d,m); // Правильный зов
```

Фактические  
параметры

```
fun2(4,8.7); // Ошибка в количестве параметров
```

```
fun2(4.67, 5,7); // ошибка в типах параметров
```

```
fun2(3,m,d); // ошибка в порядке следования контролируется  
пользователем
```

# Формальные и фактические параметры(3)

Если в качестве параметров передаются параметры значения, то в качестве фактических можно передавать переменные, константы и выражения.

**Пример:**

```
int k,l,n=6;  
float d,m=90.45
```

Формальные параметры

```
int fun1(int a,float b){....} // описание функции fun1
```

```
// вызовы функции
```

```
k=fun1(n,m); // фактические параметры переменные
```

```
printf("f=%5d",fun1(5,78.9)); // фактические параметры константы
```

```
l=fun1(2-n%3,m/k-34.78); // фактические параметры выражения
```



# Формальные и фактические параметры(4)

**1. Все параметры передаются по значению!**

**2. Если надо вернуть значение, то передают указатель или ссылку:**

а) указатель

**void prog(int a, int \*b) { \*b=a; } // Будут описаны дальше**

**ВЫЗОВ: prog(c,&d);**

б) ссылка

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

**ВЫЗОВ: prog(c, d);**

**3. Если надо запретить изменение параметра, переданного адресом, то его описывают const**  
**int prog2(const int \*a) { ...}**

# Формальные и фактические параметры(5)

## Понятие ссылки

В C++ ссылка определена как другое имя уже существующего объекта. Основные достоинства ссылок проявляются при работе с функциями.

**<тип данных>& <имя ссылки> <инициализатор>**

В соответствии с синтаксисом определение может быть:

**<тип данных>& <имя ссылки>= <выражение>**

или

**<тип данных>& <имя ссылки>( <выражение>)**

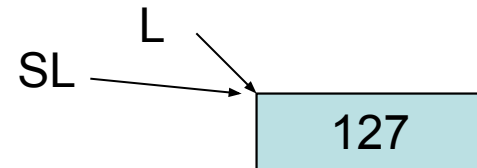
В качестве выражения может быть *имя* некоторого объекта, имеющего место в памяти.

Значением ссылки после инициализации становится *адрес* этого объекта.

**Пример определения ссылки:**

```
int L=127;
```

```
int &SL=L; //Значением ссылки SL является адрес переменной L
```



# Формальные и фактические параметры(6)

Если используется подпрограмма функция, которая возвращает в вызывающую подпрограмму формируемое значение, то в теле функции обязательно наличие оператора возврата, передающего это значение .

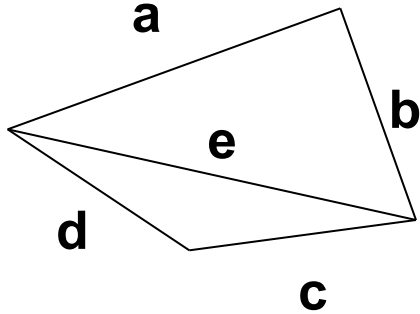
```
int max(int a,int b)
{ if (a>b) return a
  else    return b
}
        ВЫЗОВ    k=max(i,j);
```

Если используется подпрограмма процедура, то она должна возвращать результаты через параметры. В этом случае необходимо использовать ссылки или указатели.

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

ВЫЗОВ swap(i,j);

# Определение площади четырехугольника



Площадь четырехугольника определяем как сумму площадей треугольников.

Площадь треугольника определяем по формуле Герона.

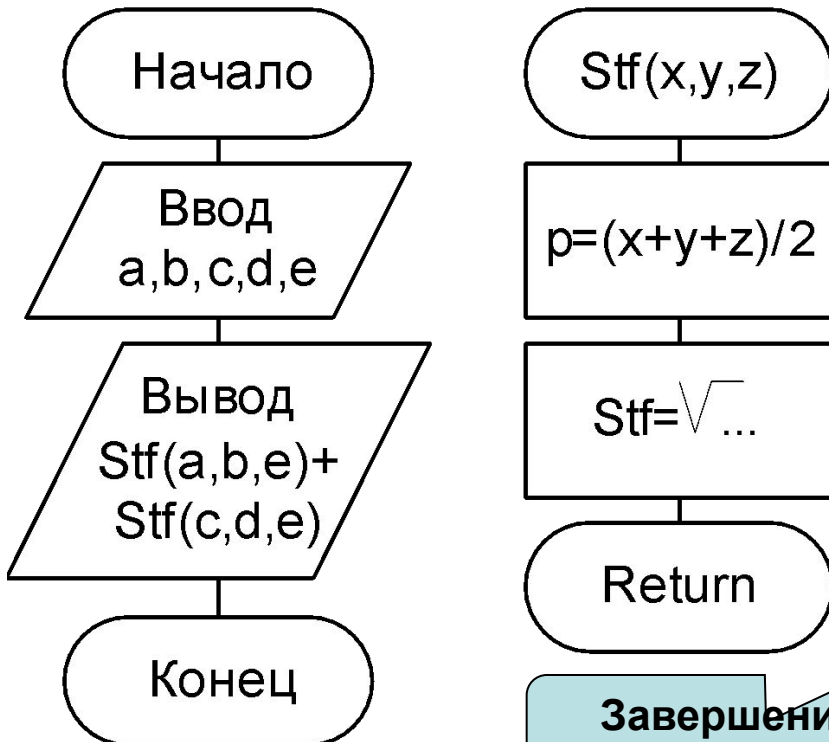
В качестве подпрограммы реализуем вычисление площади треугольника, поскольку эта операция выполняется два раза с разными параметрами.

# Схемы алгоритмов подпрограмм

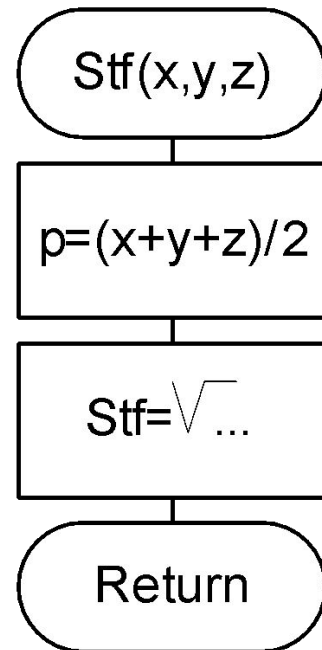
## Подпрограмма-функция

Начало алгоритма подпрограммы

Формальные параметры



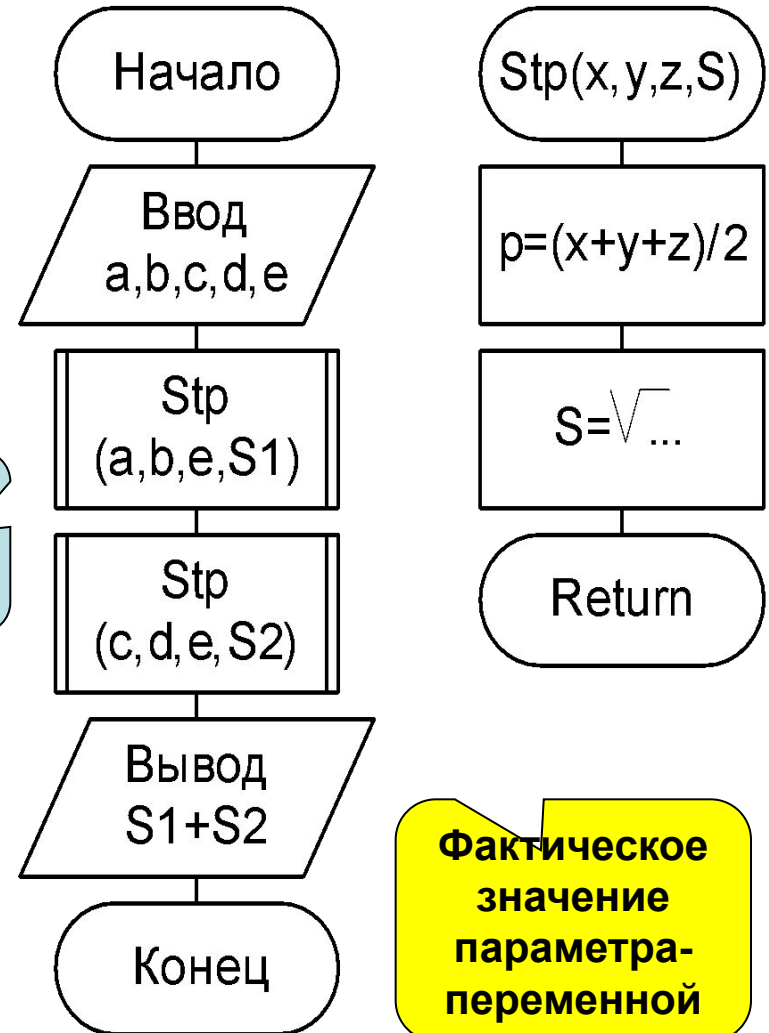
Фактические параметры



Завершение подпрограммы

## Подпрограмма-процедура

Формальный параметр-переменная в заголовке на схеме не выделяется



Фактическое значение параметра-переменной

Вызов процедуры

# Функция

Глобальные  
переменные

Тип  
возвращаемого  
значения

```
// Ex4_1.cpp :  
#include "stdafx.h"  
#include <stdio.h>  
#include <math.h>  
float a,b,c,d,e;  
float stf(double x,double y,double z)  
{double p;  
    p=(x+y+z)/2;  
    return sqrt(p*(p-x)*(p-y)*(p-z));  
}
```

Локальная  
переменная

```
int main(int argc, char* argv[])  
{ puts("Input side a,b,c,d");  
  scanf("%f %f %f %f",&a,&b,&c,&d);  
  puts("Input diagonal e");  
  scanf("%f",&e);  
  printf("A= %5.2f , A=%5.2f , C=%5.2f , D=%5.2f ,  
    E=%5.2f  \n",a,b,c,d,e);  
  printf("PLOSHAD= %8.4f\n",stf(a,b,e)+stf(c,d,e));  
  return 0;  
}
```

Вычисление  
возвращаемого  
значения

Вызов функции из  
выражения

# Функция не возвращающая результата(процедура)

```
// Ex4_2.cpp
```

```
#include "stdafx.h"
```

```
#include <stdio.h>
```

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

```
float a,b,c,d,e,S1,S2;
```

```
void stp(float x,float y,float z,float &S)
```

```
{float p;
```

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

```
    S=sqrt(p*(p-x)*(p-y)*(p-z));}
```

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

```
{ puts("Input side a,b,c,d");
```

```
    scanf("%f %f %f %f",&a,&b,&c,&d);
```

```
    puts("Input diagonal e");
```

```
    scanf("%f",&e);
```

```
stp(a,b,e,S1); stp(c,d,e,S2);
```

```
printf("PLOSHAD= %8.4f\n",S1+S2);
```

```
return 0;}
```

Глобальные  
переменные

Возвращаемое  
Значение-ссылка

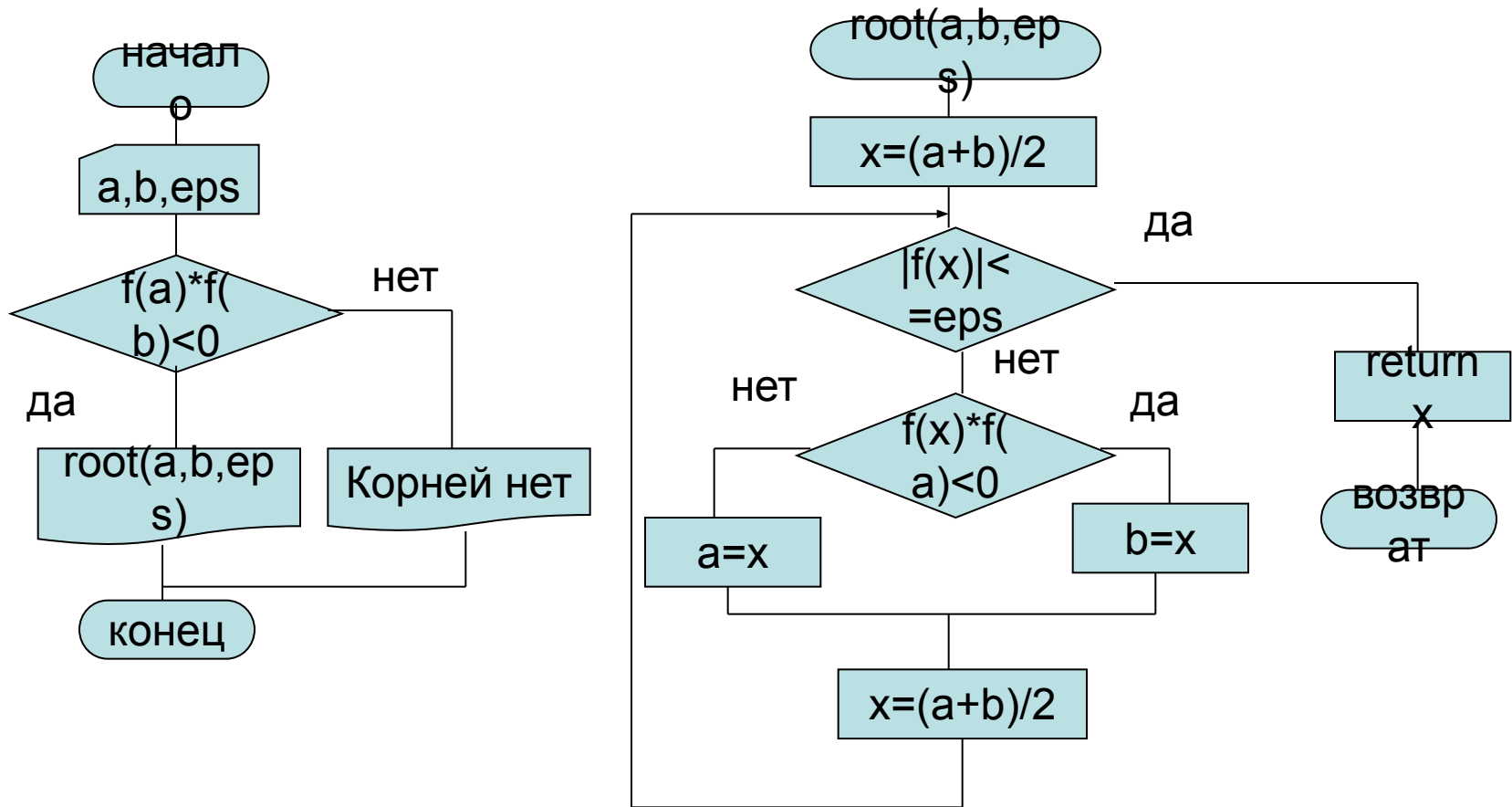
Локальная  
переменная

Вызов  
процедуры

# Пример использования функций

## Пример.

Написать программу вычисления корня функции  $y=x^2*\cos(x)-x+1$  на отрезке  $a,b$  с точностью  $\epsilon$  методом половинного деления.





# Программа вычисления корня функции $y=x^2*\cos(x)-x+1$

```
// Ex4_3.cpp
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
float F1(float x)
{ return x*x*cos(x)-x+1; }
float root(float a, float b, float eps)
{ float fx, fa, fb, x;
  x=(a+b)/2;
  fx=F1(x);
  while(fabs(fx)>=eps)
  { fa=F1(a); fb=F1(b);
    if (fx*fa<0)
    { fb=fx; b=x; }
    else
    { fa=fx; a=x; }
    x=(a+b)/2;
    fx=F1(x); }
  return x;
}
```

Функция для вычисления  
 $F1=x^2*\cos(x)-x+1$

Список формальных  
параметров

Локальные  
переменные

Тело функции  
вычисления корня  
функции F1 на отрезке

Возвращаемое значение

# Программа вычисления корня функции $y=x^2*\cos(x)-x+1(2)$

```
int main(int argc, char* argv[])
{float xn,xk,eps;
 puts("Input Xn,Xk,eps");
 scanf("%f %f %f",&xn,&xk,&eps);
 if (F1(xn)*F1(xk)<0)
 printf("Root F1 on %7.3f - %7.3f raven ",xn,xk);
 printf("%8.6f\n",root(xn,xk,eps));
 else printf("Root F1 on %7.3f - %7.3f",xn,xk);
     printf("is epsent\n");
 return 0;
}
```

Проверка  
существования  
корня на отрезке

Вызов функции root

Список фактических  
параметров функции root

# Примеры использования подпрограмм

Пример. Написать программу вычисления суммы ряда с заданной точностью.

$$S = \sum_{i=1}^{k=\infty} (-1)^i / x^i;$$



$$-1/x + 1/x^2 - 1/x^3 + 1/x^4 - \dots$$

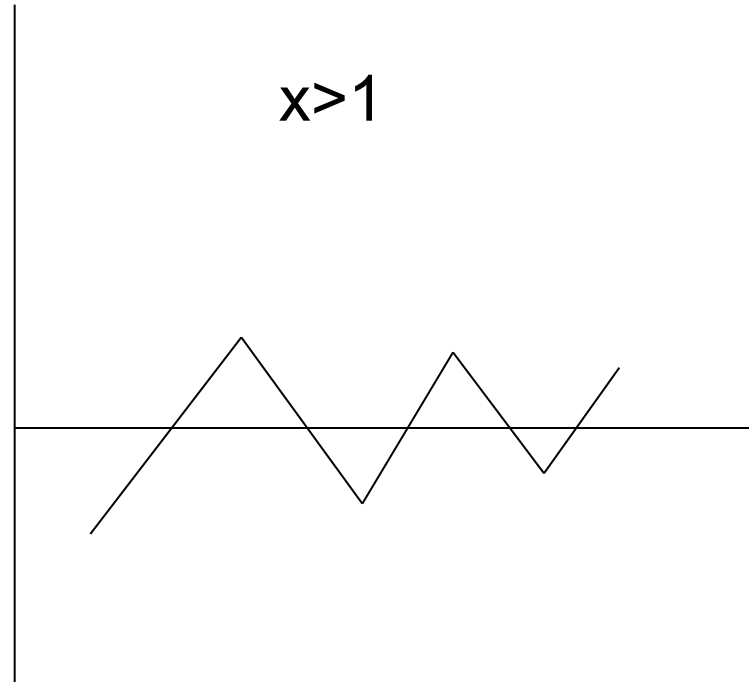
$$R_1 = -1/x;$$

$$R_2 = -R_1/(x);$$

$$R_3 = -R_2/(x);$$

.....

$$R_i = -R_{i-1}/(x);$$



## Примеры использования подпрограмм (2)

```
// Ex4_4.cpp
```

```
#include "stdafx.h"
```

```
#include <stdio.h>
```

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

```
float sumr(float x, float eps)
```

```
{ int i; float s, r;
```

```
  s=0;
```

```
  r=-1/x;
```

```
  i=1;
```

```
  while (fabs(r) > eps)
```

```
  { s+=r;
```

```
    i=i+1;
```

```
    r=-r*i/((i-1)*x);
```

```
  }
```

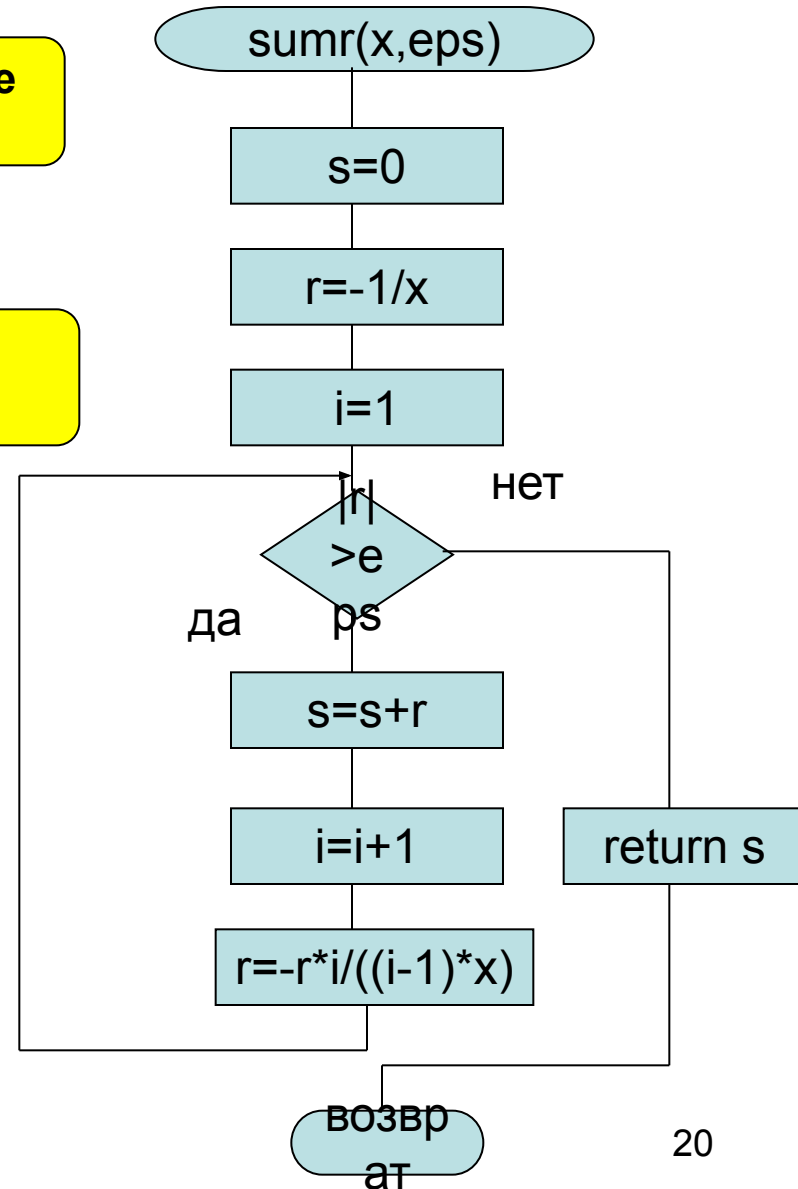
```
  return s;
```

```
}
```

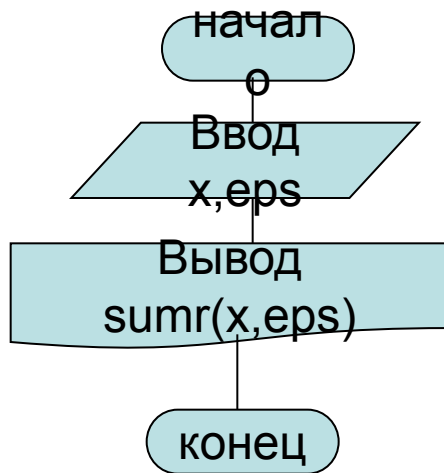
Формальные  
параметры

локальные  
данные

Возвращаемое  
значение



## Примеры использования подпрограмм (3)



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

```
{ float x,eps;
```

```
puts("Input x,eps");
```

```
scanf("%f %f", &x, &eps);
```

Вызов функции

Фактические  
параметры

```
puts("Result");
```

```
printf("SUMMA Ryada. = %8.7f\n", sumr(x,eps));
```

```
return 0;
```

```
}
```

## Примеры использования подпрограмм (4)

//Ex4\_4a.cpp **Подпрограмма - процедура**

```
#include "stdafx.h"
```

```
#include <stdio.h>
```

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

```
void sumrp(float x, float eps,  
           float& s, int & k)
```

```
{ float r;
```

```
  s=0;
```

```
  r=-1/x;
```

```
  k=0;
```

```
  while (fabs(r)>eps)
```

```
  { s+=r;
```

```
    k=k+1;
```

```
    r=-r*k/((k-1)*x);
```

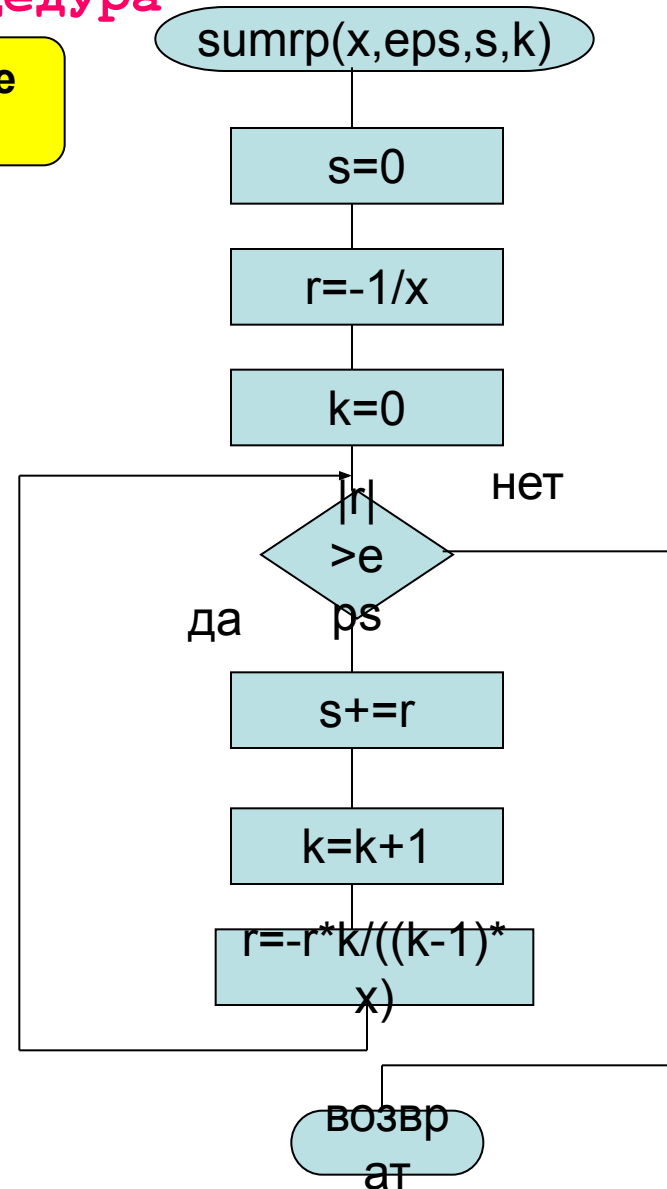
```
  }
```

```
}
```

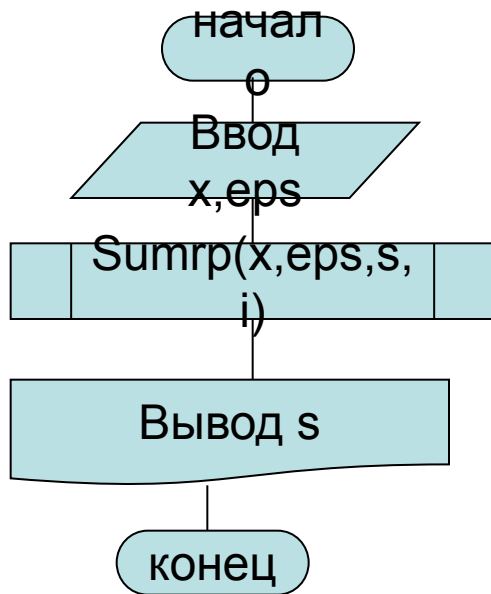
Формальные  
параметры

локальные  
данные

Возвращаемые  
значение



## Примеры использования подпрограмм (3)



**Вызов функции**

```
int main(int argc, char* argv[])
{ float x,eps,sm;int n;
  puts("Input x,eps");
  scanf("%f %f",&x,&eps);
  smrp(x,eps,sm,n);
  printf("SUMMA Ryada. = %8.7f\n",sm);
  printf("Kol. Iteracy = %8d\n",n);
  return 0;
}
```

**Фактические  
параметры**

## 4.2 Передача массивов в подпрограммы

При решении многих задач для хранения и обработки данных используются массивы.

Как уже отмечалось, существуют приемы, позволяющие осуществлять различную обработку массивов.

Такие приемы реализуют универсальные алгоритмы, которые подходят для широкого круга задач, отличающихся только типами и размерами обрабатываемых массивов.

Вполне естественно, что многие из алгоритмов целесообразно оформить в виде функции.

Массивы можно использовать в функции двояко:

- их можно описать в теле функции;
- массивы могут быть аргументами (параметрами функции) .

В силу специфики организации массивов в С++, **массивы передаются в подпрограмму как параметры переменные, однако, без использования ссылок** (особенности организации массивов будет рассмотрены далее).



## Параметры-массивы (2)

В C++ отсутствует контроль размерности массива по первому индексу при передаче этих массивов в качестве параметров!

а) `int x[5] ⇔ int x[]` – размерность  
проверяться не будет

б) `int y[4][8] ⇔ int y[][8]` – будет  
проверяться размерность  
массива по второму индексу

Пример:

```
void summa(const float x[][3], float y[], int n)
{
    int i, j;
    for(i=0; i<n; i++)
        for(j=0; j<3; j++) y[i] += x[i][j];
}
```

```
float a[5][3], b[5];
```

.....

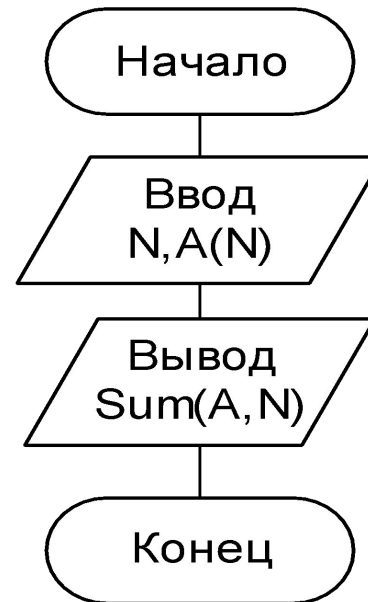
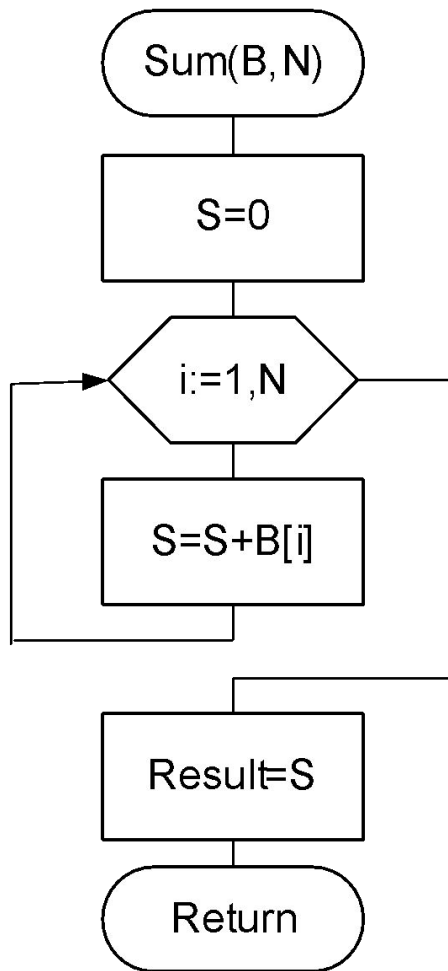
Вызов: `summa(a, b, 5);`

Формальные параметры  
массивы

Фактические  
параметры - массивы

# Примеры использования параметров массивов

**Пример.** Функция вычисления суммы элементов массива.



# Программа

```
// Ex4_5.cpp
#include "stdafx.h"
#include <stdio.h>
int sum(int a[],int n)
{int i,s;
 s=0;
 for(i=0;i<n;i++)
 s=s+a[i];
 return s;
}
int main(int argc, char*
 argv[])
{int x[10],n,i;
 puts("Input n<=10");
 scanf("%d",&n);
 printf("Input %4delemen.\n",n);
 for(i=0;i<n;i++)
 scanf("%d",&x[i]);
 printf("\n");
```

Объявление  
параметра  
массива

```
puts("INPUTED MASSIV");
for(i=0;i<n;i++)
printf("%4d",x[i]);
printf("\n");
printf("SUMMA Elem.=");
printf("%5d\n",sum(x,n));
return 0;
}
```

Фактический  
параметр массив

# Примеры использования параметров массивов

**Пример.** Написать программу удаления из матрицы  $l$  строки и  $k$  столбца с использованием подпрограмм.

```
#include "stdafx.h"  
#include <stdio.h>  
#include <math.h>  
#include <time.h>  
#include <conio.h>  
#include <stdlib.h>
```

Формальные  
параметры

```
void delsts(int a[][10],int & n,int & m,int l,int k)
```

```
{ int i,j;  
  for(i=l;i<n-1;i++)  
    for(j=0;j<m;j++)  
      a[i][j]=a[i+1][j];  
  for(j=0;j<m;j++)  
    a[n-1][j]=0;  
  n=n-1;  
  for(j=k;j<m-1;j++)  
    for(i=0;i<n;i++)  
      a[i][j]=a[i][j+1];  
  for(i=0;i<n;i++)  
    a[i][m-1]=0;  
  m=m-1;}
```

Вычеркивание  
строки  $l$

Вычеркивание  $k$   
столбца

# Пример использования параметров массивов

```
int main(int argc, char* argv[])
{ int matr[10][10],n,m,l,k,i,j;
  puts("Input n,m<=10");
  scanf("%d %d",&n,&m);
  puts("Isxodnaya Matrica");
  srand( (unsigned)time( NULL ));
  for(i=0;i<n;i++)
  {for(j=0;j<m;j++)
  {matr[i][j]=rand()/1000;
   printf("%4d",matr[i][j]);}
   printf("\n");}
  printf("Input l< %5d    k<%5d    for delete\n",n,m);
  scanf("%d %d",&l,&k);
  delsts(matr,n,m,l,k);
  puts("Isxodnaya Matrica");
  for(i=0;i<n;i++)
  {for(j=0;j<m;j++)
   printf("%4d",matr[i][j]);
   printf("\n");
  }
  getch();
  return 0;}
```

Формирование  
матрицы

Вызов функции  
преобразования  
матрицы

Печать матрицы

## 4.3 Классы памяти

В C++ переменные могут быть описаны как вне, так и внутри функций.

При этом каждой переменной присваивается класс памяти.

Класс памяти определяет

- размещение объекта в памяти (место описания);
- область действия (доступность переменной из функций);
- время жизни переменной (как долго она находится в памяти).

Есть 4 ключевых слова, используемые для описания классов памяти:

**extern** (внешние), **auto** (автоматические), **static** (статические), **register** (регистровые).

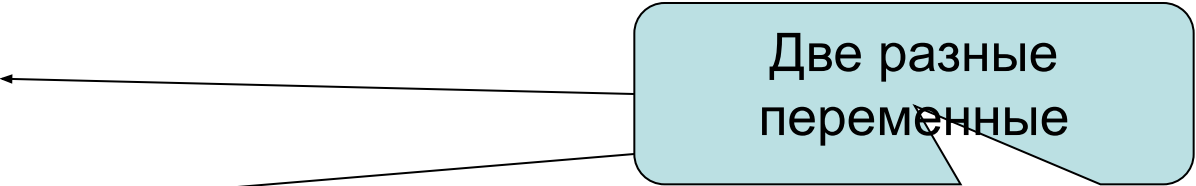
### 1. Автоматические переменные (*auto*)

main()

{**auto** int a;...}

abc()

{**auto** int a;...}



Две разные  
переменные

Место описания – локальная память, область действия – внутри функции или блока, где она определена, время действия – существует с момента вызова функции и до возврата управления.

По умолчанию все переменные описанные внутри функции - автоматические

# Классы памяти(2)

## 2. Внешние переменные (*extern*)

**extern** int a;

main()

{extern int a;...}

abc()

{extern int a;...}

bcd()

{int a;...}

Одна и та же  
переменная

Автоматическая переменная,  
которая  
внутри функции перекрывает  
внешнюю

Место описания – глобальная память, область действия – все файлы программы, где она определена, время действия – существует с момента вызова программы и до возврата управления операционной системы.

По умолчанию, если переменная описана вне функции, то она – внешняя.

# Классы памяти(3)

## 3. Статические переменные (*static*)

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

В отличие от автоматической  
статическая переменная  
увеличивается с каждым вызовом

Локальная переменная.  
При каждом вызове  
начинается с 1

Принимает значение 1 только первый раз. При  
каждом следующем вызове начинается с  
последнего значения

Место определения – внутри функции (локальная область), область действия – внутри функции, в которой она определена,

Время жизни – все время работы программы (в отличие от автоматической не исчезает, когда функция завершает работу).

Статическую переменную можно инициализировать, однако инициализация осуществляется только при первом обращении к функции.



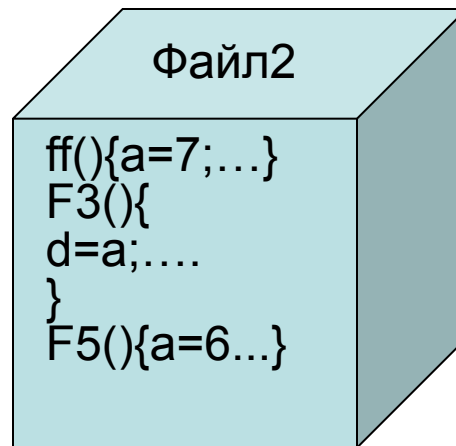
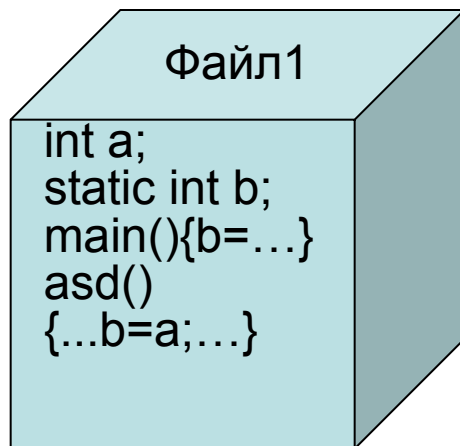
# Классы памяти (4)

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

**int a;**

**extern static int b;**

Внешняя переменная **a** доступна во всех файлах программы, а внешняя статическая **b** только в том файле, где она описана



Переменная **a** доступна обоим файлам, переменная **b** – только первому

Место описания – глобальная память,  
область действия – внутри всех функций того файла программы, где она определена,  
время действия – существует с момента вызова программы и до возврата управления операционной системы.

# Классы памяти (5)

## 5. Регистровые переменные (register)

```
register int a;
```

По доступу аналогична автоматической, но по возможности размещается в регистрах

Регистровые переменные аналогичны автоматическим, но по возможности их нужно размещать в регистровой памяти.

Если регистры заняты, то переменная размещается аналогично переменной **auto**.

### Общие рекомендации:

- По возможности следует использовать автоматические переменные.
- Внешние и статические переменные сложных структурных типов можно инициализировать.

## 4.4 Дополнительные возможности C++

### 1. Подставляемые функции

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

При таком описании функции код подставляемой функции вставляется в то место программы, откуда она вызывается.

Если вставка не возможна, то вызов идет по стандартному механизму.

Однако, на использование inline функции есть ограничения:

- функция не должна быть большой;
- не должна содержать циклов ;
- не должна содержать операторов переходов или переключателей ;
- не может быть рекурсивной ;
- не должна вызываться более одного раза в выражении ;
- не должна вызываться до определения.

# Дополнительные возможности C++(2)

## 2. Переопределяемые функции

В C++ функции могут различаться по сигнатуре (списку, количеству и типам параметров) и типу возвращаемого параметра.

Поэтому можно определить несколько вариантов одной и той же функции с одинаковыми именами, но с разными списками параметров (сигнатурами).

При вызове, компилятор по сигнатуре определяет нужный аспект функции и вызывает нужную реализацию функции.

```
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) ;}
```

# Пример переопределения функции

Пример. Написать программу для определения максимального элемента массива произвольного размера и типа.

```
// Ex4_15.cpp
#include "stdafx.h"
#include <stdio.h>
int max_elem(int n,int array[])
{int max;
    max=array[0];
    for(int i=1;i<n;i++)
        if (array[i]>max)
            max=array[i];
    return max;
}
long max_elem(int n,long array[])
{long max;
    max=array[0];
    for(int i=1;i<n;i++)
        if (array[i]>max)
            max=array[i];
    return max;
}
```

```
float max_elem(int n,float
array[])
```

```
{float max;
    max=array[0];
    for(int i=1;i<n;i++)
        if (array[i]>max)
            max=array[i];
    return max;
}
```

```
double max_elem(int n,double
array[])
```

```
{double max;
    max=array[0];
    for(int i=1;i<n;i++)
        if (array[i]>max)
            max=array[i];
    return max;
}
```

## Пример переопределения функции(2)

```
int main(int argc, char* argv[])
{
    int x[]={10,20,30,40,50,25};
    long f[]={12L,34L,10L,44L,8L};
    float y[]={0.1,0.003,0.5,0.7,0.009};
    double z[]={0.0007,0.00008,0.0002,0.00004};

    printf(" max_elem(6,x)=%4d\n",max_elem(6,x));

    printf(" max_elem(5,f)=%6d \n",max_elem(5,f));

    printf(" max_elem(5,y)=%5.3f\n",max_elem(5,y));

    printf(" max_elem(4,z)=%7e \n",max_elem(4,z));
    return 0;
}
```

Вызов  
функции

# Дополнительные возможности C++(3)

## 3. Параметры функции, принимаемые по умолчанию

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

Список параметров по умолчанию

Примеры вызова:

```
InitWindow();
```

Все параметры берутся по умолчанию

```
InitWindow(20, 10);
```

Меняются размеры окна, остальные - по умолчанию

Если нужно изменить например цвет, то все предыдущие надо повторить.

```
InitWindow(80, 25, GREEN);
```

Меняем цвет рамки окна, остальное - по умолчанию

```
InitWindow(80, 25, BLUE, GREEN);
```

Меняем цвет фона окна, остальное - по умолчанию

## 4.4 Аргументы командной строки

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

где **argc** - количество параметров командной строки +1;

**argv[0]** - может содержать полное имя файла программы, например "**A:\ddd.exe**".

**argv[1]** - содержит первый параметр из командной строки;

**argv[2]** - содержит второй параметр из командной строки и т.д. Номер предпоследнего элемента массива **argv[ ]** равен **argc**. Он содержит последний параметр. Последний элемент массива **argv** содержит **NULL**.

*Примечание.* Пример использования параметров командной строки будет рассмотрен позднее.



# Модули C++. Файлы заголовков.

Среда Visual C++ позволяет создавать и отлаживать программы, использующие не только стандартные, но и пользовательские библиотеки (модули).

Модуль C++ обычно включает два файла:

- заголовочный файл с расширением .h
- файл реализации с расширением .cpp.

Заголовочный файл играет роль интерфейсной секции модуля.

В него помещают объявление экспортируемых ресурсов модуля:

- прототипы (заголовки) процедур и функций,
- объявление переменных, типов и констант.

Заголовочный файл подключают командой **#include** “<имя модуля>.h” в файле реализации программы или другого модуля, если они используют ресурсы описываемого модуля.

# Модули C++(2)

Файл реализации представляет собой секцию реализации модуля. Он должен содержать команды подключения используемых модулей, описания экспортируемых процедур и функций, а также объявления внутренних ресурсов модуля. В начало каждого файла реализации необходимо поместить оператор подключения заголовочного файла **stdafx.h**:

**#include "stdafx.h".**

Этот файл осуществляет подсоединение специальных библиотек среды, и при его отсутствии компилятор выдает ошибку «не найден конец файла».

При создании файл проекта уже содержит заготовку главной функции программы – функции **main()**.

Для создания файлов модуля и добавления их к проекту необходимо вновь вызвать многошаговый Мастер заготовок.

Это делается с использованием команды меню **File/New**.

Выполнение этой команды при открытом проекте вызовет открытие окна Мастера заготовок на вкладке **Files**

# Модули C++ (Ex3\_03)

Файл Mod.h:

```
int nod(int a,int b);
```

Файл Mod.cpp:

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

Файл Ex3\_03.cpp:

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

