# Глава 3. **Функции. 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

# 3.1 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**

# Параметры функции

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

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

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);`

**3. Если надо запретить изменение параметра, переданного адресом, то его описывают const**

```
int prog2(const int *a) { …}
```

# 3.2 Классы памяти

1. Автоматические (локальные) переменные

```
main()
{ int a;…}
abc()
{ int a;…}
```

Автоматические (локальные) переменные

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

# 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

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

# 3.3 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 (Ex3_05):**

```
void summa(const float x[ ][3], float *y)
{  int i,j;
   for(i=0;i<5;i++)
     for(y[i]=0,j=0;j<3;j++) y[i]+=x[i][j];
}
```

Вызов: `summa(a,b);`

# 3.4 Параметры-строки

**Функции типа «строка» целесообразно писать как процедуры-функции.**

**Пример.** Функция удаления «лишних» пробелов между словами.

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

Вызовы: `puts(strdel(str,strres));` или

```
        strdel(str,strres);
```

# 3.5 Параметры-структуры

**Имя структуры не является указателем на нее.**

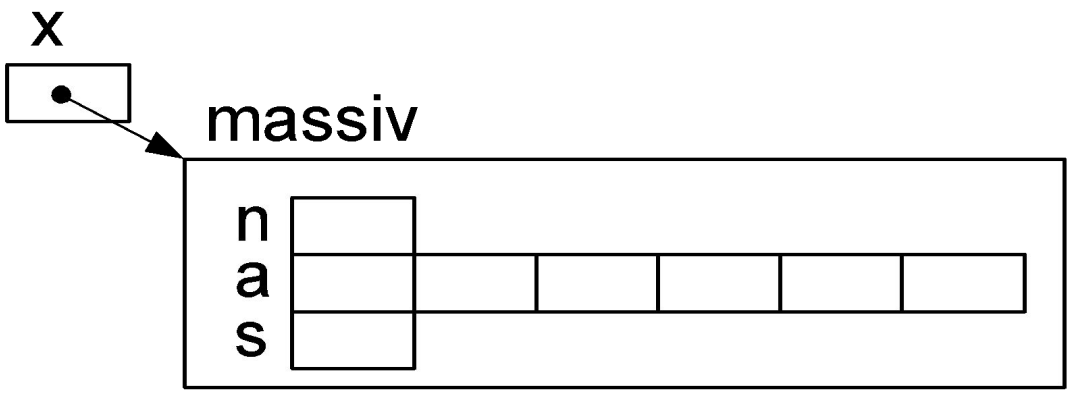
**Пример 1.** Сумма элементов массива (указатель).

```
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).

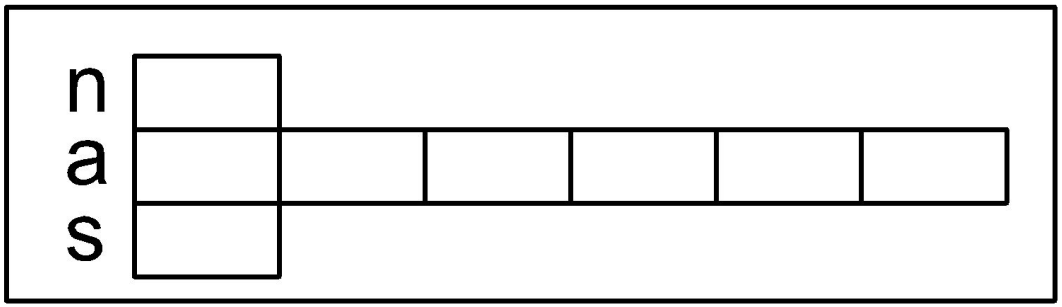
```
struct mas{int n; int a[10]; int sum;} massiv;

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;
}
```

Вызов:

```
summa(massiv);
```

x=massiv



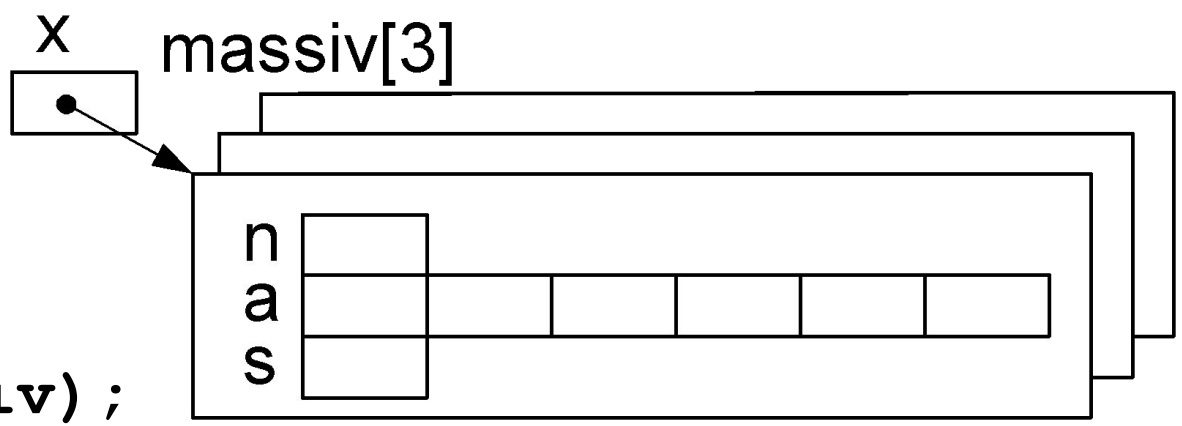
# Параметры-структуры (3)

**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);**



# 3.6 Параметры-функции

**Пример (Ex3_01).**

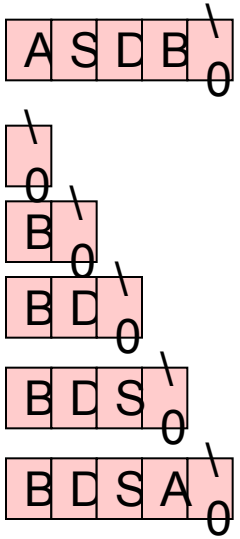
```
#include <stdio.h>
int add(int n,int m) {return n+m;}
int sub(int n,int m) {return n-m;}
int mul(int n,int m) {return n*m;}
int div(int n,int m) {return n/m;}
int main()
{ int (*ptr)(int,int);
  int a=6, b=2; char c='+';
  while (c!=' ')
   {  printf("%d%c%d=",a,c,b);
      switch (c)  { case '+': ptr=add; c='-';break;
                    case '-': ptr=sub; c='*';break;
                    case '*': ptr=mul; c='/';break;
                    case '/': ptr=div; c=' ';  }
      printf("%d\n",a=ptr(a,b));   }
  return 0;
}
```

6+2=8
8-2=6
6*2=12
12/2=6

**Указатель на функцию**

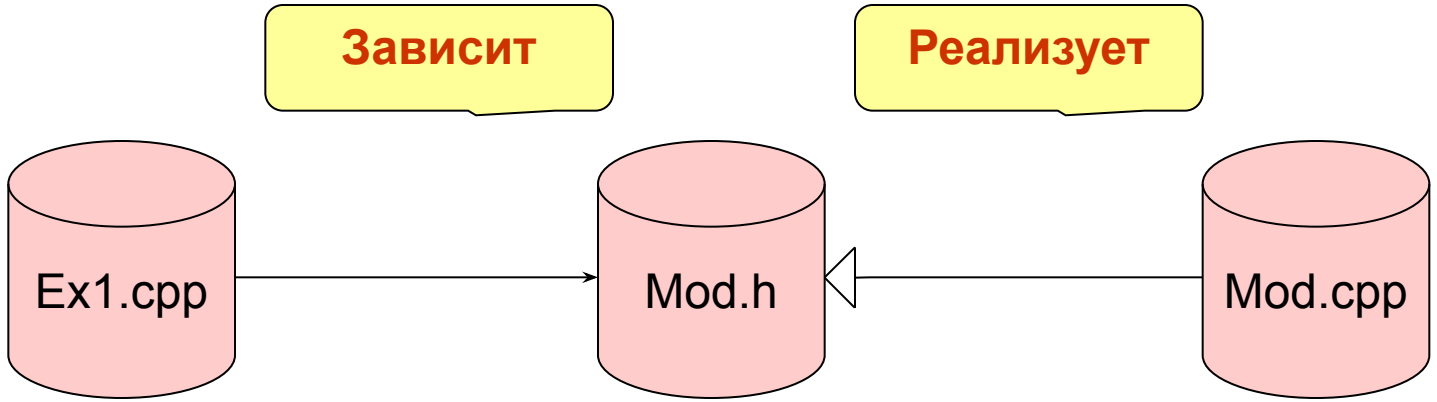
# 3.7 Рекурсия

**Пример**. Переворот строки (**Ex3_02**).




```
#include <stdio.h>
#include <string.h>
void reverser(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(int argc, char* argv[]) {
char s[20],sr[20];
    printf("Input string:");
    scanf("%s",s);
    reverser(s,sr);
    printf("Output string: %s\n",sr);
    return 0;
}
```

# 3.8 Модули C++ (Ex3_03)




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

```
#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;
}
```

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

# 3.9 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 C++ был введен механизм логического разделения области глобальных имен программы, который получил название *пространства имен*.

Имена, определенные в пространстве имен, становятся локальными внутри него и могут использоваться независимо от имен, определенных в других пространствах. Таким образом, снимается требование уникальности имен программы.

**namespace** [<имя>] { <Объявления и определения> }

Например:

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

Имя пространства имен должно быть уникальным, но может быть и опущено. Если имя пространства опущено, то считается, что определено неименованное пространство имен (см. далее).

# Доступ к элементам пространства имен

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.).

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*/
  }
```

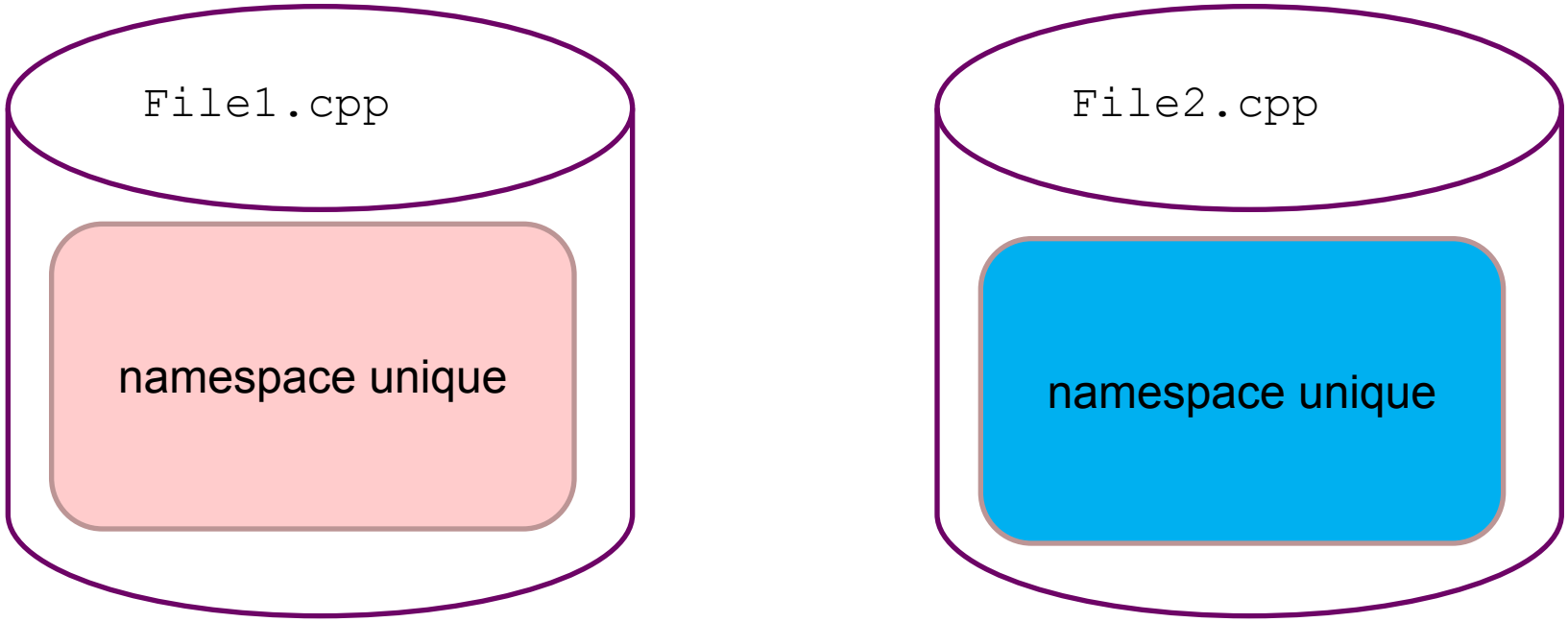
# Непоименованное пространство имен

Непоименованное пространство имен невидимо в других файлах:

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

При трансляции оно именуется как “unique”, доступное в самом файле:

```
namespace unique { namespace-body }
using namespace unique;
```

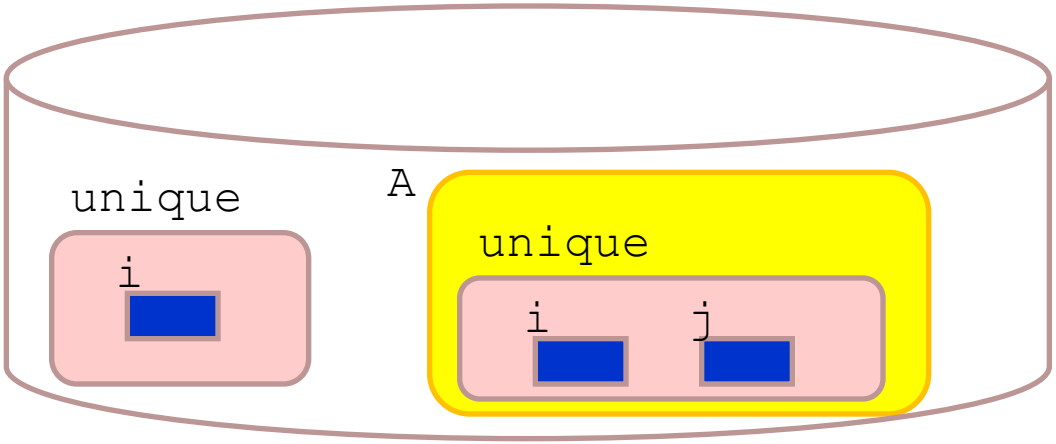


# Пример определения пространства имен

```
namespace { int i; }          // unique::i
void f() { i++; }             // 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
}
```

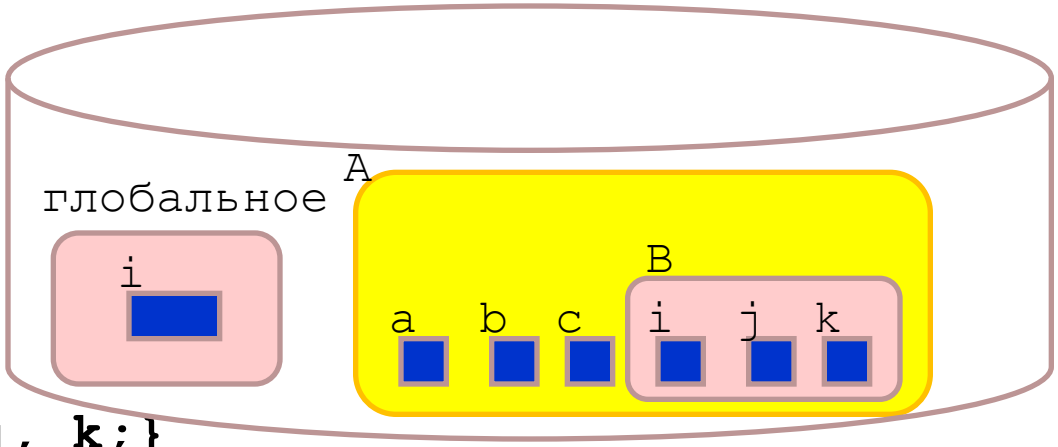
using namespace unique;
подразумевается по умолчанию



# Глобальное пространство имен

Приложение включает одно глобальное пространство имен. Имена, входящие в это пространство, объявляются без указания имени пространства имен.

**Пример:**




```
int i;

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.

# 3.10 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) C или C++:**

```
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]** - третийпараметр из командной строки – `"vvv.txt";`

**argv[4] -** содержит **NULL**.

# 3.11 Дополнительные возможности функций C++

## 1. 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.

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

Достоинство: уменьшается время вызова подпрограммы.
Недостаток: увеличивается объем программы;

# Дополнительные возможности функций C++

2. Переопределяемые функции или 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
```

# Дополнительные возможности функций C++

3. **Параметры функции, принимаемые по умолчанию** – механизм, позволяющий описать параметры функции с наиболее часто встречающимися их 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.