# Лекция 3

Функции в C++.
Перегрузка функций.

# Структурирование кода

Одним из способов структурирования программного кода является использование процедур и функций (процедурное программирование).

**Функция** представляет собой именованную группу операторов, которые выполняют определенную задачу. Эта группа операторов задействуется путем **вызова функции**.

Основные причины использования функций:

1) стремление сократить размер кода,

2) стремление упростить кодирование часто повторяющихся задач.

# Методы использования функций

Часто выделяют следующие этапы создания функций и работы с ними:

1. объявление функции (прототип)
2. определение функции
3. вызов (выполнение функции)

# 1) Объявление (прототип)

- не содержит тела функции, но указывает ее имя, арность, типы аргументов и возвращаемый тип данных,
- представляет собой описание интерфейса функции (ее сигнатуру).

*Пример*

```
int func1(int k);
```

# 2) Определение

- содержит сигнатуру функции,
- содержит тело функции – то есть код, выполняемый при вызове.

```
int func1(int k)
{
    int n = 1;
    for(int i = 1; i <= k; i++)
      n *= i;
    return n;
}
```

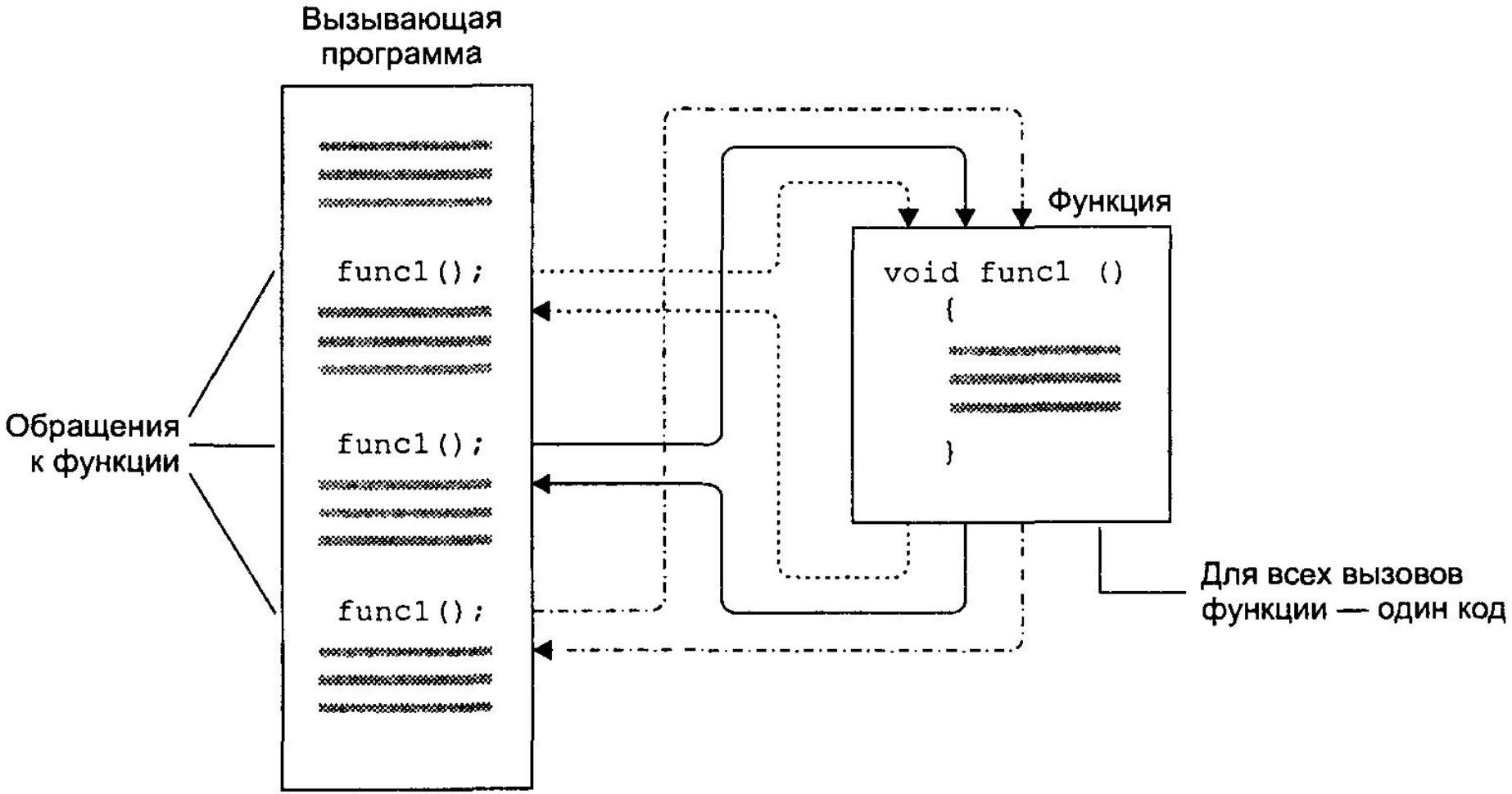
# 3) Вызов функции

- внешне похож на прототип,
- инициирует выполнение функции.

```
int F;
...
F = func1(5);
```

Параметры, передаваемые в функцию при вызове, называются **фактическими** параметрами, в отличие от **формальных** параметров, используемых в определении функции. В момент вызова фактические параметры должны иметь **определенное значение**.

# Передача программного управления при вызове функции



# Использование стека вызовов

**Стек вызовов** (*call stack*) — область памяти, используемая при вызовах функций.

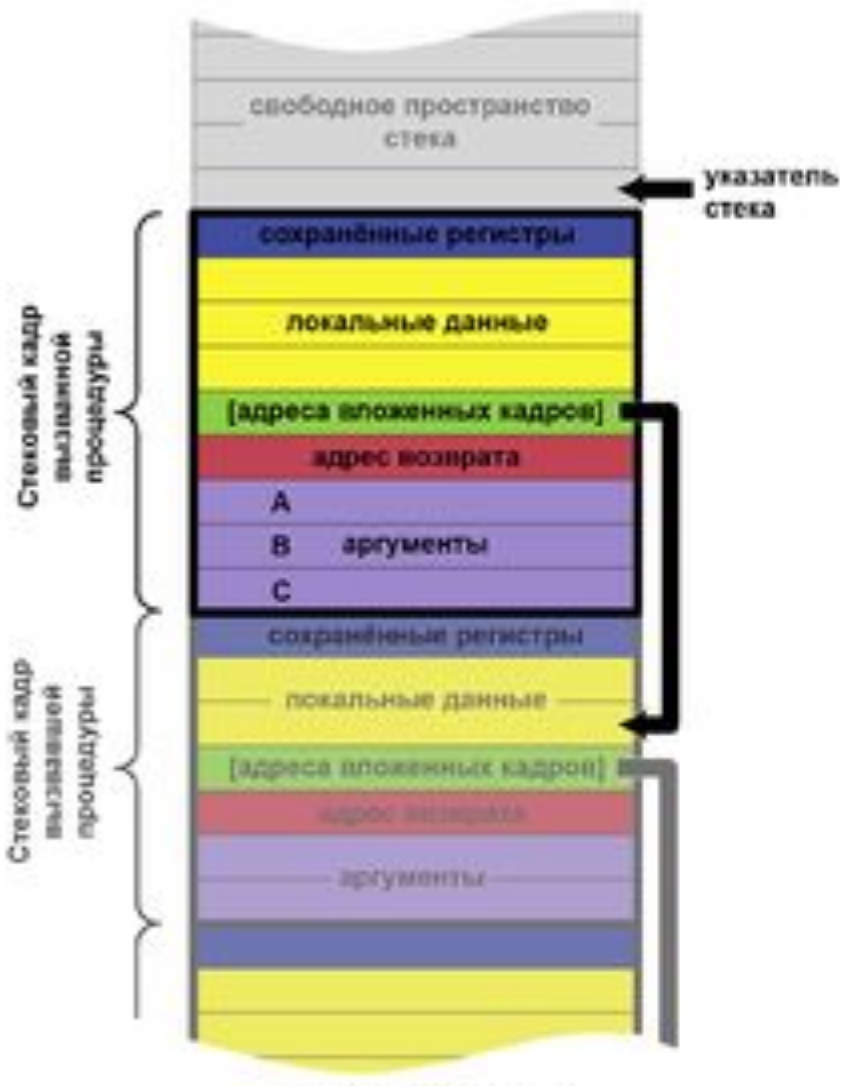
В стек помещаются:

1) фактические параметры,

2) адрес возврата из функции,

3) локальные переменные,

4) значения регистров.



# Ссылки

Ссылка – особый тип данных, являющийся скрытой формой указателя, который при использовании автоматически разыменовывается. Ссылка может использоваться как псевдоним переменной, на которую ссылается.

```
тип &имя_ссылки = имя переменной;
```

# Пример использования ссылки

```
int main(int argc, char* argv[])
{
   int value = 15;
   int &reference = value;      // ссылка
   cout << "value     = " << value     << endl;
   cout << "reference = " << reference << endl;
   reference+=15;               // изменяем
   cout << "value     = " << value     << endl;
   cout << "reference = " << reference << endl;
   system("pause");
   return 0;
}
```

# Передача аргументов в функцию

В языке C++ существует несколько способов передачи аргументов внутрь вызываемой функции

1. **Передача по значению.** В стеке создается копия передаваемого значения, и функция получает доступ к этой копии.
2. **Передача по адресу.** Функции передается указатель на область памяти, где располагается аргумент. Используя операцию разыменования, функция получает прямой доступ к значению.
3. **Передача по ссылке.** В функцию передается ссылка на аргумент (фактически, его адрес). Для доступа к значению аргумента разыменование не требуется.

```
void f(int x) // по значению
{
    cout << x;
    x = 1;
    cout << x;
}

void g(int* x) // по адресу
{
    cout << *x;
    *x = 2;
    cout << *x;
}

void h(int& x) // по ссылке
{
    cout << x;
    x = 3;
    cout << x;
}
```

```
int main()
{
    int x = 0;
    f(x);
    g(&x);
    h(x);
    return 0;
}
```

```
01 02 23
```

В функцию могут передаваться не только переменные стандартных типов (int, float, char, ...), но и структурные переменные. В C++ структура может быть передана в функцию любым из трех способов - по значению, по адресу, по ссылке.

```
#include <iostream>
using namespace std;

struct Distance
{
  int feet;
  float inches;
};

void display(Distance dd)
{
  cout << dd.feet << "\'-" << dd.inches << "\"";
}
```

```
void scale(Distance& dd, float f)
{
   float inches = (dd.feet*12 + dd.inches) * f;
   dd.feet = static_cast<int>(inches / 12);
   dd.inches = inches - dd.feet * 12;
}

int main()
{
   ...
   Distance d1 = {12, 6.5};
   Distance d2 = {10, 5.5};
   scale(d1, 0.5);
   scale(d2, 0.25);
   display(d1);
   display(d2);
   ...
   return 0;
}
```

**Пример приложения**. Прорисовка окружностей в текстовом режиме

```
#include "mscon.h"

struct circle
{
   int x, y;                  // центр окружности
   int radius;                // радиус
   color fillcolor;           // цвет
   fstyle fillstyle;          // стиль заполнения
};

void circ_draw(circle c)
{
   set_color(c.fillcolor);
   set_fill_style(c.fillstyle);
   draw_circle(c.x, c.y, c.radius);
}
//... (окончание)
```

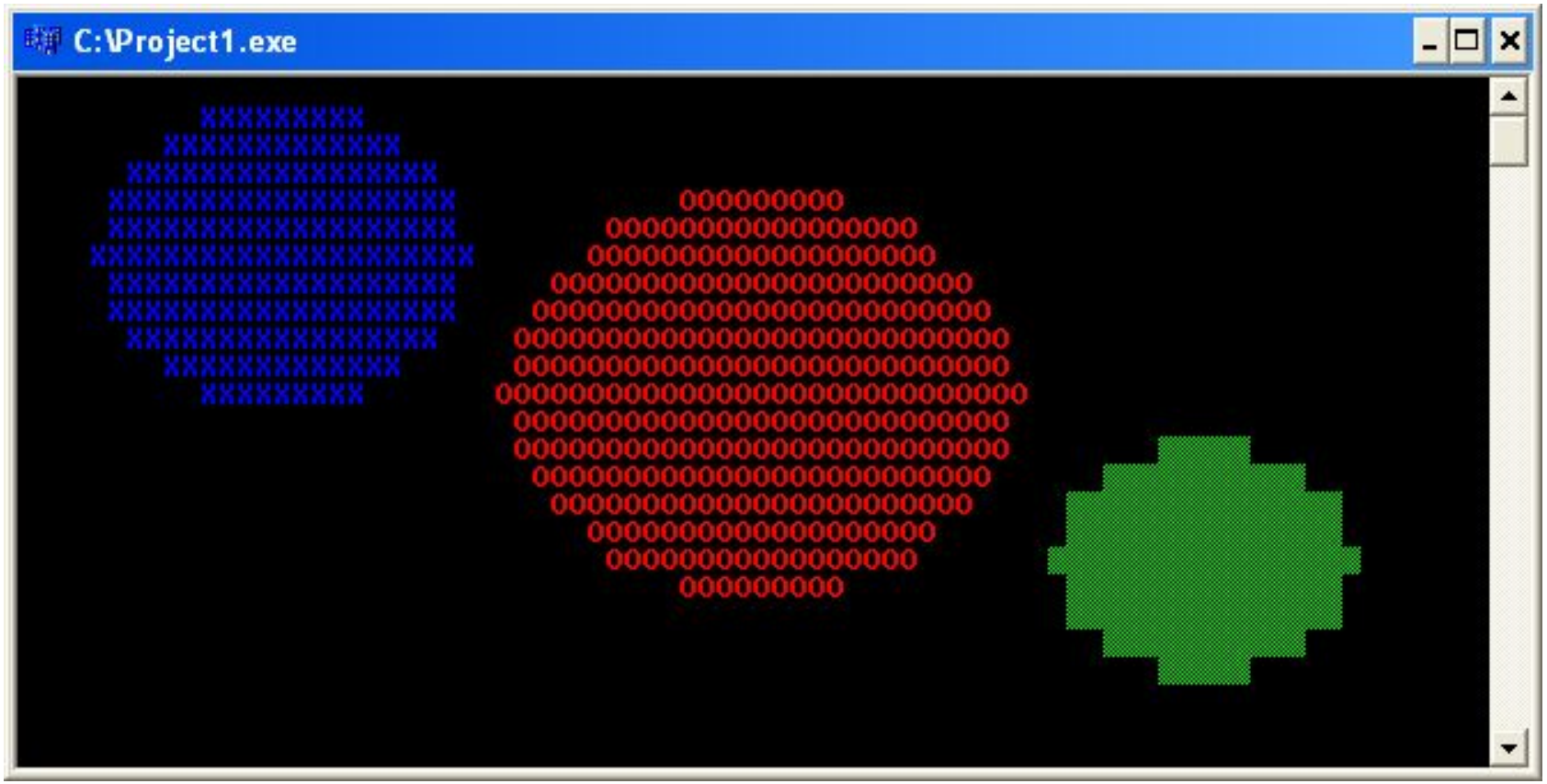
```
// ... (начало)

int main()
{
    init_graphics();

    circle c1 = {15, 7, 5, cBLUE, X_FILL};
    circle c2 = {41, 12, 7, cRED, O_FILL};
    circle c3 = {65, 18, 4, cGREEN, MEDIUM_FILL};

    circ_draw(c1);
    circ_draw(c2);
    circ_draw(c3);
    set_cursor_pos(1, 25);
    return 0;
}
```

# Результата работы приложения



# Значение, возвращаемое функцией

Функция **возвращает** результат своей работы в вызывающую программу с помощью оператора return.

Кроме переменных стандартных типов (int, float, char, ...) в языке C++ функция может возвращать значения структурного типа.

# *Пример*. Структура как возвращаемое значение

```
Distance add(Distance dd1, Distance dd2)
{
   Distance dd3;

   dd3.inches = dd1.inches + dd2.inches;
   dd3.feet = 0;
   if(dd3.inches >= 12.0)
   {
      dd3.inches -= 12.0;
      dd3.feet++;
   }
   dd3.feet += dd1.feet + dd2.feet;
   return dd3;
}
```

# Перегрузка функций

**Перегрузка** функций в языке C++ означает использование нескольких вариантов функции с одним и тем же именем, но с разным числом аргументов или разными типами аргументов.

При вызове перегруженной функции, выбор ее конкретного варианта происходит на основе анализа количества и типа аргументов. Такой подход получил называние **статического полиморфизма**.

На этапе трансляции программы происходит контроль одноимённых функций, чтобы они различались по сигнатуре.

*Пример*. Перегрузка функц

```
// объем куба
int volume(int s)
{
   return(s*s*s);
}

// объем цилиндра
double volume(double r, int h)
{
   return(3.14*r*r*h);
}

// объем параллелепипеда
long volume(long l, int b, int h)
{
   return(l*b*h);
}
```

```
int main()
{
  cout << volume(10);
  cout << volume(2.5,8);
  cout << volume(9,5,4);
  return 0;
}
```

# Константные аргументы функции

Ссылки на аргументы функции используются не только в случаях, когда необходимо менять значения аргументов. Часто ссылочный механизм передачи применяется для **повышения скорости работы** программы (пример - передача через стек крупных структурных объектов с большим количеством полей).

В случае, если ссылка используется только для повышения эффективности, свободный доступ к значению аргумента может быть ограничен с помощью **модификатора const**.

*Пример*. Использование константной ссылки в качестве аргумента функции

```
void func(int& a, const int& b)
{
  a = 107;    // нет ошибки
  b = 111;    // ошибка: попытка изменения
}             //    константного аргумента

int main()
{
  int alpha = 7;
  int beta = 11;
  func(alpha, beta);
  return 0;
}
```

# Создание библиотечного модуля

*Пример*: модуль арифметических функций.

Заголовочный файл библиотеки «mathfuncs.h»

```
#ifdef MATHFUNCS_H
#define MATHFUNCS_H

 namespace mathspace
 {
        double add(double a, double b);
        double subtract(double a, double b);
        double multiply(double a, double b);
        double divide(double a, double b);
 }

#endif
```

# Файл реализации библиотечных функций «mathfuncs.cpp»

```
#include "mathfuncs.h"
#include <stdexcept>

 namespace mathspace
 {
    double add(double a, double b)
    {
       return a + b;
    }

    double subtract(double a, double b)
    {
       return a - b;
    }
    // ... (продолжение)
```

```
    //... (начало)
    double multiply(double a, double b)
    {
       return a * b;
    }
    double divide(double a, double b)
    {
       if (b == 0)
       {
         throw std::invalid_argument("b = 0!");
       }
       return a / b;
    }
}
```

# Использование модуля

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

int main()
{
    double a = 5.0;
    int b = 10;
    cout << "a + b = " <<
        mathspace::add(a, b) << endl;
    cout << "a - b = " <<
        mathspace::subtract(a, b) << endl;
    cout << "a * b = " <<
        mathspace::multiply(a, b) << endl;
    cout << "a / b = " <<
        mathspace::divide(a, b) << endl;
    return 0;
}
```