

# Программирование

## Лекция 6

# Функции

- Функции в C++ можно разбить на две категории: функции, которые возвращают значения, и функции, значения не возвращающие.

`x = sqrt(6.25); // возвращает значение 2.5 и присваивает его переменной x`

Вызывающая функция

Вызываемая функция

```
int main()
```

```
{
```

```
① ↓ ...  
...  
↓
```

```
X = sqrt(6+25);
```

```
⑤ ↓ ...  
...  
↓
```

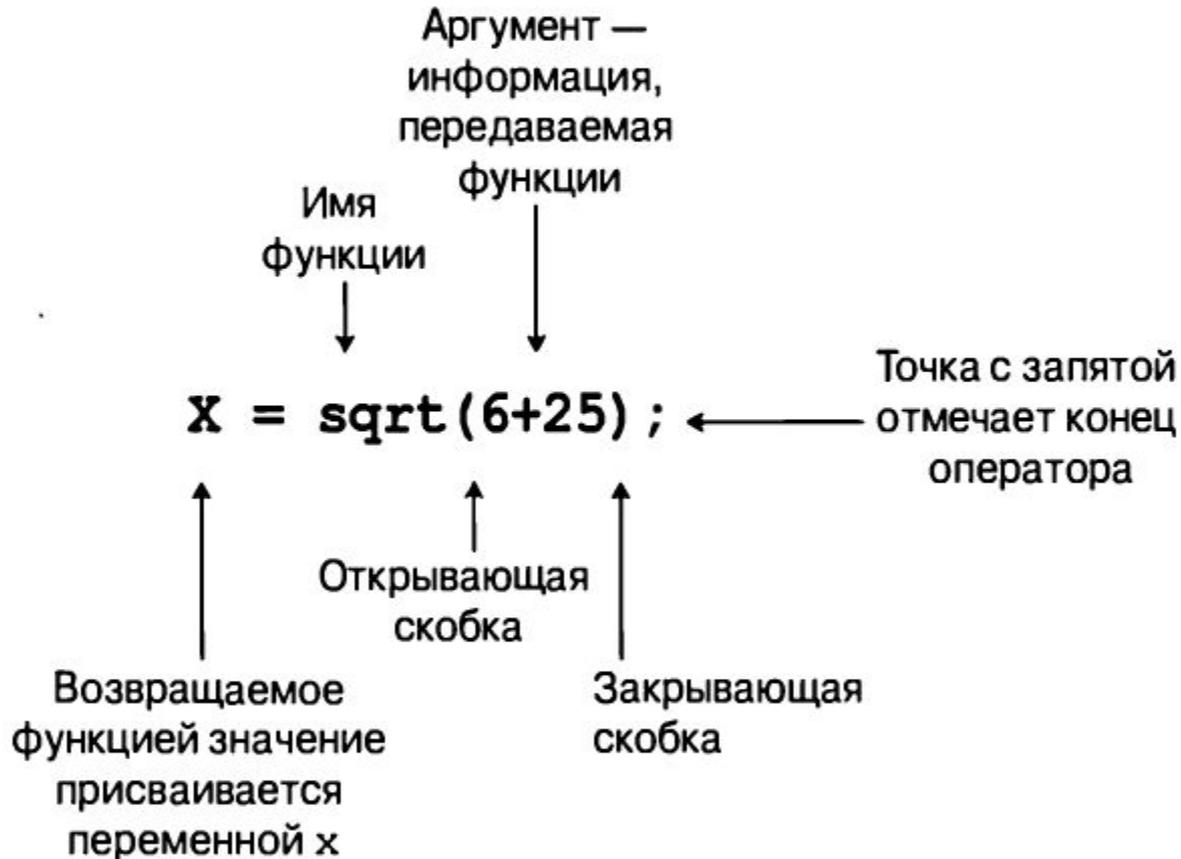
Код функции `sqrt()`

```
...  
...  
...  
...  
...  
...
```

Вызов функции

Возврат в вызывающую функцию

# Синтаксис вызова функции



# Разновидности функций

Для некоторых функций требуется более одного элемента

```
double pow(double, double); // прототип функции с двумя аргументами
answer = pow(5.0, 8.0); // вызов функции со списком аргументов
```

Есть функции, которые не принимают аргументов:

```
int rand(void); // прототип функции, не принимающей аргументов
myGuess = rand(); // вызов функции без аргументов
```

Существуют также функции, которые не имеют возвращаемого

```
void bucks(double); // прототип для функции, не имеющей возвращаемого значения
bucks(1234.56); // вызов функции; возвращаемого значения нет
```

# Функции, определяемые пользователем

```
// ourfunc.cpp -- определение собственной функции
#include <iostream>
void simon(int);          // прототип функции simon()

int main()
{
    using namespace std;
    simon(3);             // вызов функции simon()
    cout << "Pick an integer: ";
    int count;
    cin >> count;
    simon(count);        // еще один вызов simon()
    cout << "Done!" << endl;
    return 0;
}

void simon(int n)        // определение функции simon()
{
    using namespace std;
    cout << "Simon says touch your toes " << n << " times." << endl;
} // функции void не требуют операторов return
```

```
Simon says touch your toes 3 times.
Pick an integer: 512
Simon says touch your toes 512 times.
Done!
```

# Функции как программные модули C++

Для того чтобы использовать функцию в C++, вы должны выполнить следующие шаги:

- предоставить определение функции;
- представить прототип функции;
- вызвать функцию.

Например, **strlen** – функция нахождения длины строки (библиотека **cstring** )

# Определение, прототипирование и вызов функции

```
#include <iostream>

void simple();           // прототип функции
int main()
{
    using namespace std;
    cout << "main() will call the simple() function:\n";
    simple();           // вызов функции
    cout << "main() is finished with the simple() function.\n";
    // cin.get();
    return 0;
}

// Определение функции
void simple()
{
    using namespace std;
    cout << "I'm but a simple function.\n";
}
```

```
main() will call the simple() function:
I'm but a simple function.
main() is finished with the simple() function.
```

# Определение функции

Все функции можно разбить на две категории:

- те, которые не возвращают значений,
- и те, которые их возвращают.

Функции, не возвращающие значений, называются функциями типа **void** и имеют следующую общую форму:

```
void имяФункции(списокПараметров)
{
    оператор (ы)
    return;           // не обязательно
}
```

```
void cheers(int n) // возвращаемое значение отсутствует
{
    for (int i = 0; i < n; i++)
        std::cout << "Cheers! ";
    std::cout << std::endl;
}
```

# Функция с возвращаемым значением

```
имяТипа имяФункции(списокПараметров)
{
    оператор(ы)
    return значение;    // значение приводится к типу имяТипа
}
```

- выражение должно сводиться по типу к **имяТипа** либо может быть преобразовано в **имяТипа**
- Язык C++ накладывает ограничения на типы возвращаемых значений: возвращаемое значение не может быть массивом. Все остальное допускается — целые числа, числа с плавающей точкой, указатели и даже структуры и объекты.
- Хотя функция C++ не может вернуть массив непосредственно, она все же может вернуть его в составе структуры или объекта.

# ФУНКЦИИ

- Функция завершается после выполнения оператора **return**.

```
int bigger(int a, int b)
{
    if (a > b )
        return a;           // если a > b, функция завершается здесь
    else
        return b;          // в противном случае функция завершается здесь
}
```

```
double cube(double x)      // x умножить на x и еще раз умножить на x
{
    return x * x * x;      // значение типа double
}
```

# Прототипирование и вызов функции

```
#include <iostream>
void cheers(int);          // прототип: нет значения возврата
double cube(double x);    // прототип: возвращает double
int main()
{
    using namespace std;
    cheers(5);              // вызов функции
    cout << "Give me a number: ";
    double side;
    cin >> side;
    double volume = cube(side);    // вызов функции
    cout << "A " << side << "-foot cube has a volume of ";
    cout << volume << " cubic feet.\n";
    cheers(cube(2));       // защита прототипа в действии
    return 0;
}
```

```
void cheers(int n)
{
    using namespace std;
    for (int i = 0; i < n; i++)
        cout << "Cheers! ";
    cout << endl;
}
double cube(double x)
{
    return x * x * x;
}
```

```
Cheers! Cheers! Cheers! Cheers! Cheers!
Give me a number: 5
A 5-foot cube has a volume of 125 cubic feet.
Cheers! Cheers! Cheers! Cheers! Cheers! Cheers! Cheers! Cheers!
```

# Зачем нужны прототипы?

- Прототип описывает интерфейс функции для компилятора. Это значит, что он сообщает компилятору, каков тип возвращаемого значения, если оно есть у функции, а также количество и типы аргументов данной функции.

```
double volume = cube(side);
```

## Синтаксис прототипа

- Прототип функции является оператором, поэтому он должен завершаться точкой с запятой.

```
double cube(double x); // добавление ; к заголовку для получения прототипа
```

- Прототип функции не требует предоставления имен переменных-параметров; достаточно списка типов

```
void cheers(int); // в прототипе можно опустить имена параметров
```

# Что обеспечивают прототипы

Прототипы значительно снижают вероятность допущения ошибок в программе. В частности, они обеспечивают следующие моменты:

- Компилятор корректно обрабатывает возвращаемое значение.
- Компилятор проверяет, указано ли правильное количество аргументов.
- Компилятор проверяет правильность типов аргументов. Если тип не подходит, компилятор преобразует его в правильный, когда это возможно.

```
double z = cube(); // неверное количество переменных
```

```
cheers(cube(2)); // два несоответствия типа в одном  
операторе
```

Прототипирование происходит во время компиляции и называется **статическим контролем типов**.

# Аргументы функций и передача по значению

- В C++ аргументы обычно передаются **по значению**. Это означает, что числовое значение аргумента передается в функцию, где присваивается новой переменной.

```
double volume = cube(side);
```

```
double cube(double x)
```

Переменная, которая используется для приема переданного значения, называется *формальным аргументом* или

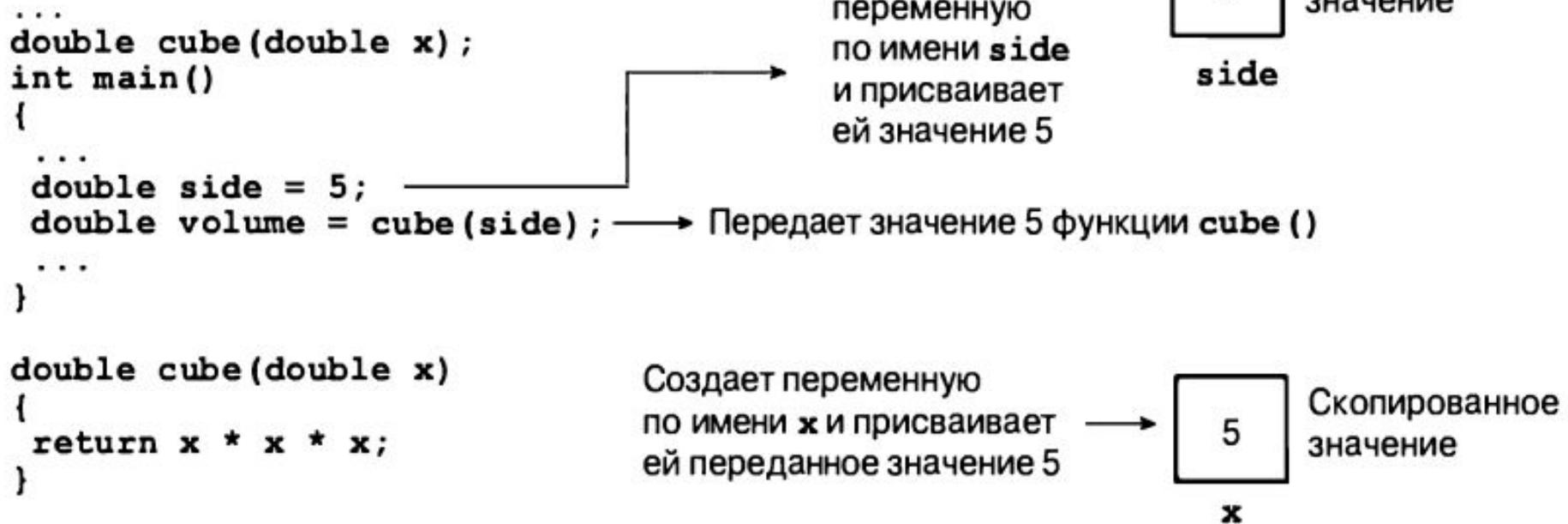
*формальным*

*параметром*. Значение, переданное функции, называется

*фактическим аргументом* или *фактическим параметром*.

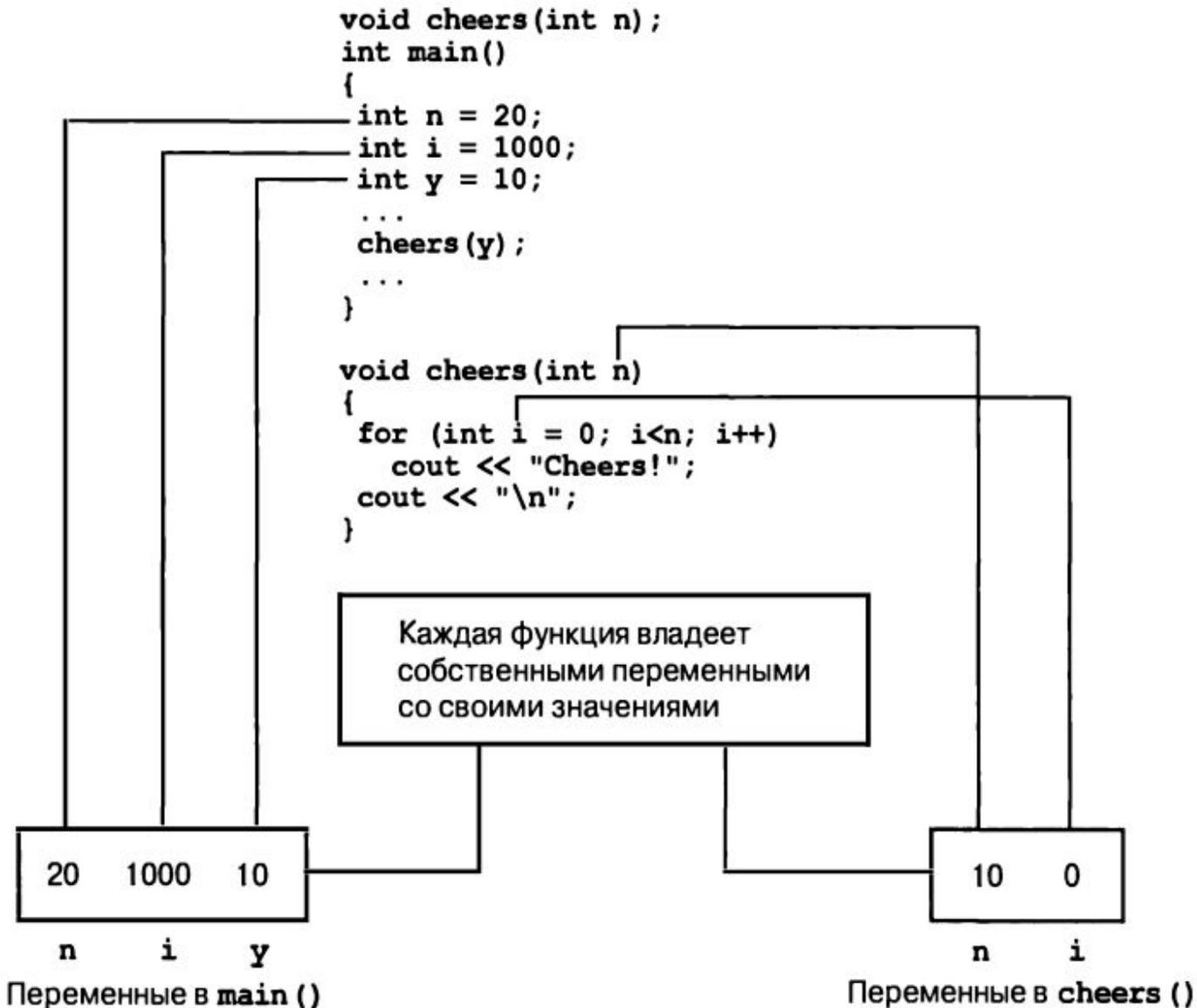
Иногда в стандарте C++ слово **аргумент** используется для обозначения фактического аргумента или параметра, а слово **параметр** — для обозначения формального аргумента или параметра.

# Передача по значению



Когда функция вызывается, компьютер выделяет память, необходимую для этих переменных. Когда функция завершается, компьютер освобождает память, которая была использована этими переменными.

# Локальные переменные



# Множественные аргументы

Функция может принимать более одного аргумента. При вызове функции такие аргументы просто отделяются друг от друга запятыми:

```
n_chars('R', 25);
```

Определение функции:

```
void n_chars(char c, int n) // два параметра
```

Если функция имеет два параметра одного и того же типа, то типы каждого параметра должны указываться по отдельности:

```
void fifi(float a, float b) // объявляет каждую переменную отдельно  
void fufu(float a, b) // не допускается
```

Прототипы:

```
void n_chars(char c, int n); // прототип, стиль 1
```

```
void n_chars(char, int); // прототип, стиль 2
```

```
double melon_density(double weight, double volume);
```



# Еще одна функция с двумя аргументами

Если вы должны угадать 6 значений из 51, математики говорят, что у вас имеется один шанс выигрыша из  $R$ , где  $R$  вычисляется по следующей формуле:

$$R = \frac{51 \times 50 \times 49 \times 48 \times 47 \times 46}{6 \times 5 \times 4 \times 3 \times 2 \times 1}$$

```
#include <iostream>
// Примечание: некоторые реализации требуют применения double вместо long double
long double probability(unsigned numbers, unsigned picks);
int main()
{
    using namespace std;
    double total, choices;
    // Ввод общего количества номеров и количества номеров, которые нужно угадать
    cout << "Enter the total number of choices on the game card and\n"
         << "the number of picks allowed:\n";
    while ((cin >> total >> choices) && choices <= total)
    {
        cout << "You have one chance in ";
        cout << probability(total, choices);      // вычисление и вывод шансов
        cout << " of winning.\n";
        cout << "Next two numbers (q to quit): ";
        // Ввод следующих двух чисел (q для завершения)
    }
    cout << "bye\n";
    return 0;
}
```

# Еще одна функция с двумя аргументами

```
long double probability(unsigned numbers, unsigned picks)
{
    long double result = 1.0; // несколько локальных переменных
    long double n;
    unsigned p;
    for (n = numbers, p = picks; p > 0; n--, p--)
        result = result * n / p;
    return result;
}
```

Enter the total number of choices on the game card and  
the number of picks allowed:

**49 6**

You have one chance in 1.39838e+007 of winning.

Next two numbers (q to quit): **51 6**

You have one chance in 1.80095e+007 of winning.

Next two numbers (q to quit): **38 6**

You have one chance in 2.76068e+006 of winning.

Next two numbers (q to quit): **q**

bye

# Функции и массивы

```
#include <iostream>
const int ArSize = 8;
int sum_arr(int arr[], int n); // прототип
int main()
{
    using namespace std;
    int cookies[ArSize] = {1,2,4,8,16,32,64,128};
    // Некоторые системы требуют предварить int словом static,
    // чтобы разрешить инициализацию массива
    int sum = sum_arr(cookies, ArSize);
    cout << "Total cookies eaten: " << sum << "\n"; // вывод количества съеденного печенья
    return 0;
}

// Возвращает сумму элементов массива целых чисел
int sum_arr(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total = total + arr[i];
    return total;
}
```

```
Total cookies eaten: 255
```

# Функции с аргументами-строками

```
#include <iostream>
unsigned int c_in_str(const char * str, char ch);
int main()
{
    using namespace std;

    char mmm[15] = "minimum"; // строка в массиве
    // Некоторые системы требуют предварить char словом static,
    // чтобы разрешить инициализацию массива

    char *wail = "ululate"; // wail указывает на строку
    unsigned int ms = c_in_str(mmm, 'm');
    unsigned int us = c_in_str(wail, 'u');
    cout << ms << " m characters in " << mmm << endl; // вывод количества символов m
    cout << us << " u characters in " << wail << endl; // вывод количества символов u

    // Эта функция подсчитывает количество символов ch в строке str
    unsigned int c_in_str(const char * str, char ch)
    {
        unsigned int count = 0;
        while (*str) // завершение, когда *str равно '\0'
        {
            if (*str == ch)
                count++;
            str++; // перемещение указателя на следующий символ
        }
        return count;
    }
}
```

```
3 m characters in minimum
2 u characters in ululate
```



# Функции и структуры

```
#include <iostream>
struct travel_time
{
    int hours;
    int mins;
};
const int Mins_per_hr = 60;
travel_time sum(travel_time t1, travel_time t2);
void show_time(travel_time t);

int main()
{
    using namespace std;
    travel_time day1 = {5, 45};           // 5 часов 45 минут
    travel_time day2 = {4, 55};           // 4 часов 55 минут
    travel_time trip = sum(day1, day2);
    cout << "Two-day total: ";           // итог за два дня
    show_time(trip);
    travel_time day3 = {4, 32};
    cout << "Three-day total: ";         // итог за три дня
    show_time(sum(trip, day3));
    return 0;
}
```

# ФУНКЦИИ И СТРУКТУРЫ

```
travel_time sum(travel_time t1, travel_time t2)
{
    travel_time total;
    total.mins = (t1.mins + t2.mins) % Mins_per_hr;
    total.hours = t1.hours + t2.hours +
        (t1.mins + t2.mins) / Mins_per_hr;
    return total;
}

void show_time(travel_time t)
{
    using namespace std;
    cout << t.hours << " hours, "
         << t.mins << " minutes\n";           // часов, минут
}
```

```
Two-day total: 10 hours, 40 minutes
Three-day total: 15 hours, 12 minutes
```

# Рекурсия

- Функция C++ обладает интересной характеристикой — она может вызывать сама себя. Эта возможность называется рекурсией.

```
void recurs(списокАргументов)
{
    операторы1
    if (проверка)
        recurs(аргументы)
    операторы2
}
```

# Использование рекурсии

```
#include <iostream>
void countdown(int n);
int main()
{
    countdown(4);          // вызов рекурсивной функции
    return 0;
}
void countdown(int n)
{
    using namespace std;
    cout << "Counting down ... " << n << endl;
    if (n > 0)
        countdown(n-1);   // функция вызывает сама себя
    cout << n << ": Kaboom!\n";
}
```

Counting down ... 4	← уровень 1; добавление уровней рекурсии
Counting down ... 3	← уровень 2
Counting down ... 2	← уровень 3
Counting down ... 1	← уровень 4
Counting down ... 0	← уровень 5; финальный рекурсивный вызов
0: Kaboom!	← уровень 5; начало обратного прохода
1: Kaboom!	← уровень 4
2: Kaboom!	← уровень 3
3: Kaboom!	← уровень 2
4: Kaboom!	← уровень 1

# Вопросы

- 1. Назовите три шага по созданию функции.
- 2. Постройте прототипы, которые соответствовали бы следующим описаниям.
  - а. `igor ()` не принимает аргументов и не возвращает значения.
  - б. `tofu ()` принимает аргумент `int` и возвращает `float`.
  - в. `mpg ()` принимает два аргумента типа `double` и возвращает `double`.
  - г. `summation ()` принимает имя массива `long` и его размер и возвращает значение `long`.
  - д. `doctor ()` принимает строковый аргумент (строка не должна изменяться) и возвращает `double`.
  - е. `of course ()` принимает структуру `boss` в качестве аргумента и не возвращает ничего.
- 3. Напишите функцию, принимающую три-аргумента: имя массива `int`, его размер и значение `int`. Функция должна присвоить каждому элементу массива это значение `int`.

# Вопросы

- 4. Напишите функцию, принимающую имя массива `double` и его размер в качестве аргументов и возвращающую наибольшее значение, которое содержится в этом массиве. Обратите внимание, что функция не должна модифицировать содержимое массива.