

Лекция 6. Функции

Объявление и определение функций, параметры, возвращаемое значение, область видимости, рекурсия

1. Введение

Функции являются одним из ключевых элементов языка C++.

Они позволяют **структурировать программу**, делить код на логические блоки, повышая читаемость, повторное использование и удобство сопровождения.

Функция — это именованный блок кода, выполняющий определённую задачу. После выполнения функция может **возвращать значение** и передавать его в место вызова.

2. Объявление и определение функции

В C++ функция должна быть **объявлена** до её использования и **определена** (описана) — где указано, что именно она делает.

Объявление функции (прототип):

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

Определение функции:

```
int sum(int a, int b) {  
    return a + b;  
}
```

Использование (вызов):

```
int result = sum(5, 7);  
cout << result; // 12
```

Разделение объявления и определения удобно при работе с заголовочными файлами (.h), где пишут прототипы, а в .cpp — реализацию.

3. Параметры функции

Функции могут принимать данные — **параметры**, которые задаются в круглых скобках.

Типы параметров:

1. **По значению (by value)** — передаётся копия аргумента:
 2. `void show(int x) {`
 3. `x = 10; // изменяется только копия`
 4. `}`
 5. **По ссылке (by reference)** — передаётся ссылка на переменную:
 6. `void modify(int &x) {`
 7. `x = 10; // изменяет исходное значение`
 8. `}`
 9. **По указателю:**
 10. `void setZero(int *p) {`
 11. `*p = 0;`
 12. `}`
 13. **Параметры по умолчанию:**
 14. `void greet(string name = "Гость") {`
 15. `cout << "Привет, " << name << "!\n";`
 16. `}`
-

4. Возвращаемое значение

Функция может возвращать результат с помощью оператора `return`.

Пример:

```
double square(double x) {  
    return x * x;  
}
```

Если функция не возвращает значение, указывается тип `void`:

```
void hello() {  
    cout << "Привет, мир!" << endl;  
}
```

Функция может возвращать:

- простые типы (`int`, `double`),
 - указатели и ссылки,
 - структуры и объекты классов.
-

5. Область видимости и время жизни переменных

Область видимости (scope) определяет, где доступна переменная.

- **Локальные переменные** — создаются внутри функции, видимы только в ней.
- **Глобальные переменные** — объявлены вне функций и доступны везде.

Пример:

```
int globalVar = 5;

void func() {
    int localVar = 10;
    cout << globalVar + localVar;
}
```

Статические переменные сохраняют значение между вызовами:

```
void counter() {
    static int count = 0;
    count++;
    cout << count << endl;
}
```

6. Рекурсия

Рекурсия — это вызов функции самой себя.

Используется для решения задач, которые можно разбить на подзадачи того же типа: вычисление факториала, числа Фибоначчи, обход дерева и др.

Пример: факториал

```
int factorial(int n) {
    if (n <= 1)
        return 1;
    return n * factorial(n - 1);
}
```

Вызов:

```
cout << factorial(5); // 120
```

Важно: рекурсия должна иметь **условие выхода**, иначе произойдёт переполнение стека (stack overflow).

7. Перегрузка функций

C++ поддерживает **перегрузку функций** — возможность объявлять несколько функций с одинаковым именем, но разными параметрами.

Пример:

```
int add(int a, int b) { return a + b; }  
double add(double a, double b) { return a + b; }
```

Компилятор определяет, какую версию вызвать, исходя из типов аргументов.

8. Inline-функции

Для ускорения работы небольших функций используется ключевое слово `inline`.

При этом компилятор вставляет код функции непосредственно в место вызова (без накладных расходов на стек).

Пример:

```
inline int square(int x) { return x * x; }
```

9. Лямбда-функции (C++11 и новее)

C++ поддерживает **анонимные функции** — *лямбда-выражения*, которые можно определять прямо в коде.

Пример:

```
auto sum = [](int a, int b) { return a + b; };  
cout << sum(2, 3); // 5
```

Можно захватывать переменные из внешней области видимости:

```
int factor = 10;  
auto multiply = [factor](int x) { return x * factor; };  
cout << multiply(5); // 50
```

10. Передача функций как аргументов

Функции можно передавать другим функциям через указатели или `std::function`:

```
void apply(int x, int (*func)(int)) {  
    cout << func(x) << endl;  
}
```

```
int square(int n) { return n * n; }
```

```
apply(5, square); // 25
```

11. Организация функций в проектах

Обычно функции группируют по смыслу и помещают в отдельные файлы:

- `.h` — заголовочный файл с объявлениями;
- `.cpp` — исходный файл с реализациями.

Пример структуры:

```
math.h  
math.cpp  
main.cpp
```

12. Заключение

Функции — основа структурного программирования.

Они позволяют писать понятный, переиспользуемый и легко расширяемый код.

Понимание параметров, области видимости и рекурсии — ключ к написанию надёжных программ на C++.

13. Вопросы для самопроверки

1. Чем отличается объявление функции от определения?
2. Что такое параметры по значению и по ссылке?
3. Для чего нужны функции без возвращаемого значения (`void`)?
4. Что такое рекурсия и когда она применяется?
5. Что делает ключевое слово `inline`?

14. Рекомендуемая литература

1. Бьерн Страуструп — *Язык программирования C++ (4-е издание)*.
2. Стивен Прата — *Язык программирования C++. Лекции и упражнения*.
3. Херб Саттер — *Exceptional C++*.
4. Николай М. Джосаттис — *The C++ Standard Library: A Tutorial and Reference*.
5. ISO/IEC 14882:2020 — *The C++20 Standard*.