Функции

В рамках модуля «Язык программирования C/C++»

Мухаметов Данил Илгизович

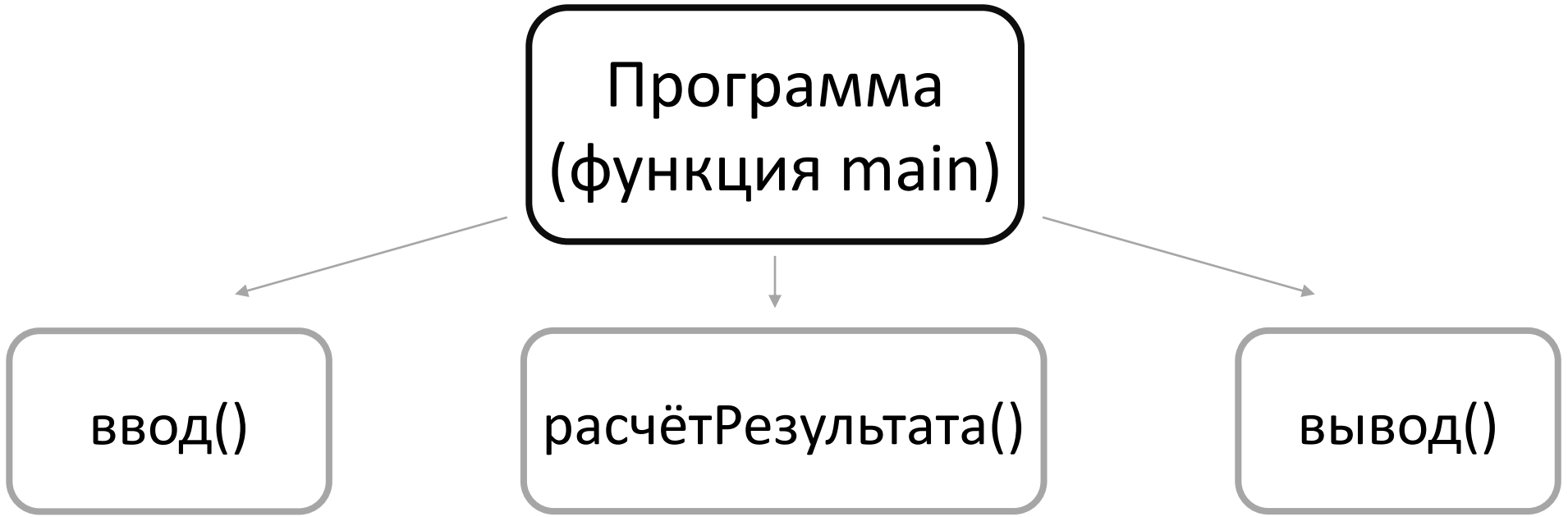
seemsclever@mail.ru

2025

Зачем нужны функции?

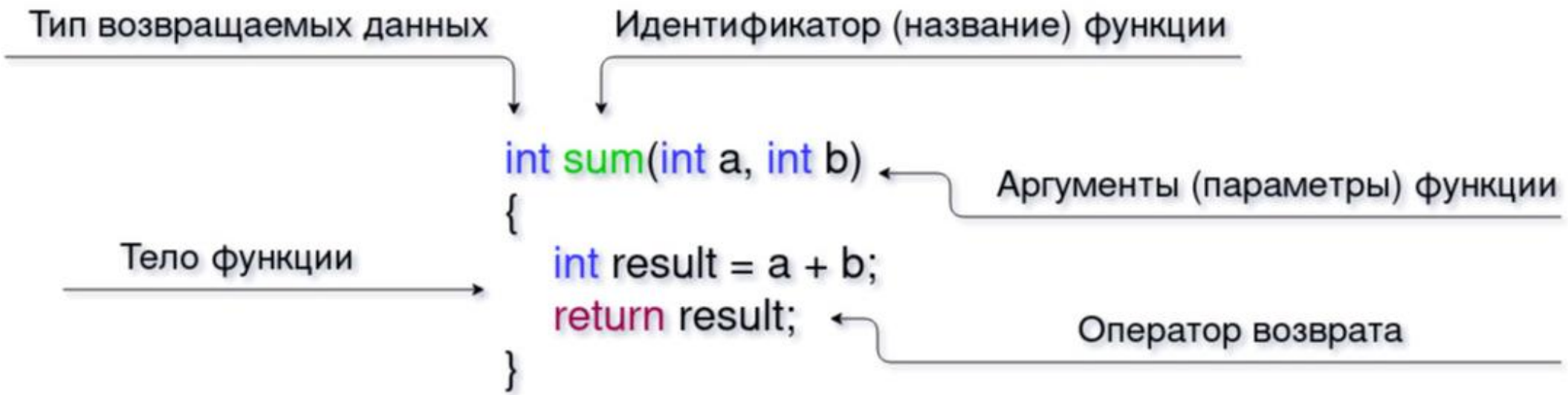
Функции позволяют структурировать программу, разделяя её на логически обособленные блоки, каждый из которых выполняет конкретную задачу.

Они упрощают чтение кода, уменьшают повторение и позволяют многократно использовать одни и те же действия.



Что такое функции?

Функция – это именованный блок кода, выполняющий определённую задачу. Она может принимать данные (аргументы), выполнять действия и возвращать результат.



Что такое сигнатура функции?

Сигнатура функции – это её "уникальный идентификатор". Функции с одинаковой сигнатурой нельзя определять повторно.

Включает в себя:

– имя функции;

– количество и типы параметров (в указанном порядке).

Не включает:

– имена параметров;

– возвращаемый тип.

```
int sum(int a, int b);      // сигнатура: sum(int, int)
double sum(double a, double b);  // сигнатура: sum(double, double)
```

Определение и вызов функции

Функция определяется один раз, а вызываться может многократно. Имя функции используется вместе с аргументами в скобках.

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

int main() {
    int value = 5;
    int result = square(value);
    cout << "Квадрат числа: " << result;
    return 0;
}
```

Возвращаемое значение

Функция может возвращать результат работы с помощью оператора return. Тип возвращаемого значения указывается перед именем функции.

```
void greet(string name) {
    cout << "Привет, " << name << "!";
}

int main() {
    greet("Андрей");
    return 0;
}
// Привет, Андрей!
```

```
double average(int a, int b) {
    return (a + b) / 2.0;
}

int main() {
    double result = average(8, 10);
    cout << "Среднее: " << result;
    return 0;
}
// Среднее: 9
```

Аргументы функции

Функция может принимать входные данные – аргументы, которые передаются при вызове и используются внутри тела функции.

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

int main() {
    int result = sumOfTwo(5, 25);
    cout << result;
    return 0;
}
// 30
```

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

int main() {
    int result = sumOfThree(5, 25, 50);
    cout << result;
    return 0;
}
// 80
```

Функция main()

Каждая программа на C++ обязательно содержит функцию main.

Основные особенности:

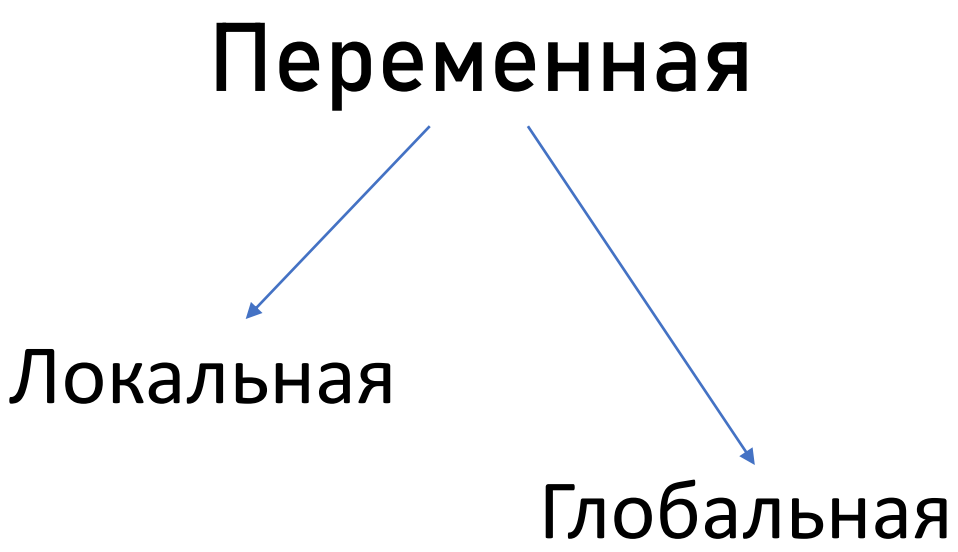
- Имеет тип int, возвращает код завершения;
- Может вызываться только системой (точка входа);
- Должна быть определена один раз в программе;
- Возврат 0 обычно означает успешное завершение.

```
int main() {
    // Код программы
    return 0;
}
```

```
int main(int argc, char* argv[])) {
    // Код программы
    return 0;
}
```

Область видимости переменных

Область видимости определяет, где в программе доступна переменная.




```
int globalVar = 10;

void printValue() {
    int localVar = 5;
    cout << globalVar << " " << localVar;
}

int main() {
    printValue();
    cout << globalVar;
    // cout << localVar; // ошибка
}
```

Жизненный цикл переменной

Жизненный цикл – это время существования переменной в памяти.

Локальные переменные – создаются при входе в блок, удаляются при выходе из блока

Глобальные переменные – создаются до начала выполнения main(), удаляются после завершения всей программы

```
int globalVar = 10;

void printValue() {
    int localVar = 5;
    cout << globalVar << " " << localVar;
}

int main() {
    printValue();
    cout << globalVar;
    return 0;
}
```

Создаются локальные переменные

Удаляются локальные переменные

Создаются глобальные переменные

Удаляются глобальные переменные

Передача переменных по значению

По умолчанию переменные передаются в функцию по значению.

Это значит, что создаётся копия, и оригинал не изменяется.

Функция работает с копией, а не с оригинальной переменной.

```
void update(int x) {
    x = x + 10;
}

int main() {
    int value = 5;
    update(value);
    cout << value;  // 5
    return 0;
}
```

Передача переменных по ссылке и через указатель

Чтобы изменить переменную внутри функции, её можно передать по ссылке или указателю.

По ссылке

```
void update(int& x) {
    x = x + 10;
}
```

По указателю

```
void update(int* x) {
    x* = x* + 10;
}
```

Вызов

```
int a = 5;
update(a);     // по ссылке
update(&a);    // через указатель
```

Передача массивов в функцию

Массивы в C++ передаются в функции по указателю.

Это значит, что функция работает с оригинальным массивом, а не с копией.

Размер массива всегда должен передаваться отдельно.

```
void print(int arr[], int size) {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}

int main() {
    int nums[5] = {1, 2, 3, 4, 5};
    print(nums, 5);  // 1 2 3 4 5
    return 0;
}
```

Передача строк в функцию

В C++ в функцию можно передавать строки разных типов. Независимо от реализации, передача в строки в функцию при вызове останется прежней.

Строка в стиле C

```
void print(const char text[]) {
    cout << text;
}
```

Строка string

```
void print(string text) {
    cout << text;
}
```

Вызов

```
print("Hello");

string msg = "Hello";
print(msg);
```

Способы передачи string в функцию

По значению

```
void print(string text){
   // код функции
}
```

- Создаётся копия строки.
- Оригинал не изменяется.

По ссылке

```
void print(string& text){
   // код функции
}
```

- Не создаётся копия строки.
- Оригинал можно изменять.

По ссылке (без изменений)

```
void print(const string& text){
   // код функции
}
```

- Не создаётся копия строки.
- Оригинал нельзя изменить.

Прототип функции

Прототип – это объявление функции до её определения.

Он сообщает компилятору, какая функция будет использоваться, даже если её реализация написана позже.

```
int sum(int, int);    // прототип

int main() {
    int result = sum(5, 3);
    cout << result;      // 8
    return 0;
}

int sum(int x, int y) {
    return x + y;
}
```

Рекурсия

Рекурсия – это вызов функцией самой себя.

Каждый вызов работает с новым значением, приближая решение.

```
void countdown(int n) {
    if (n == 0)
        return;
    cout << n << " ";
    countdown(n - 1);
}

int main() {
    countdown(5);  // 5 4 3 2 1
    return 0;
}
```

Условие выхода

Пример – нахождение факториала с помощью рекурсии

Определение факториала числа n:

$$n! = n \times (n - 1) \times (n - 2) \times \ldots \times 1$$

Рекурсивное определение факториала числа n:

$$n! = 1, \text{если } n = 0$$

$$n! = n \times (n - 1)!, \text{если } n > 0$$

```
int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}

int main() {
    cout << factorial(5);  // 120
}
```

5 × factorial(4)

→ 5 × 4 × factorial(3)

→ 5 × 4 × 3 × factorial(2)

→ 5 × 4 × 3 × 2 × factorial(1)

→ 5 × 4 × 3 × 2 × 1 × 1= 120

Рекурсия и цикл – в чём разница

	Рекурсия	Цикл
Стиль записи	Через вызов самой себя	Через ключевые слова for, while
Память	Каждый вызов сохраняется в стеке	Одна и та же область памяти
Условие завершения	Базовый случай (if)	Логическое условие (i < n)
Прозрачность идеи	Лучше выражает рекурсивные задачи	Лучше подходит для последовательностей
Производительность	Медленнее из-за затрат на стек	Быстрее и экономичнее

Перегрузка функций

Перегрузка – это возможность создавать несколько функций с одинаковым именем, но с разным числом или типами параметров.

Компилятор сам выбирает нужную версию, исходя из аргументов при вызове

```
void greet() {
    cout << "Привет!";
}

void greet(string name) {
    cout << "Привет, " << name << "!";
}

int main() {
    greet();          // Привет!
    greet("Андрей");  // Привет, Андрей!
}
```

Пример перегрузки функции

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

double printSum(double a, double b) {
    return a + b;
}

int printSum(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++)
        sum += arr[i];
    return sum;
}
```

```
int main(){
    int nums[] = {1, 2, 3};
    cout << printSum(5, 10) << endl;
    // 15
    cout << printSum(2.5, 3.3) << endl;
    // 5.8
    cout << printSum(nums, 3) << endl;
    // 6
    return 0;
}
```

Ошибки при работе с функциями

Несовпадение прототипа и определения

```
int sum(int a, int b);    // прототип
double sum(int a, int b) { return a + b; }  // ошибка: тип не совпадает
```

Отсутствие return

```
int square(int x) {
    x * x;   // ошибка: нет return
}
```

Повторное определение

```
int max(int a, int b);
int max(int x, int y);  // ошибка: сигнатура та же
```

Спасибо за внимание!