**ЛЕКЦИЯ 4. ФУНКЦИИ**

Содержание

[1 ЗАЧЕМ ИСПОЛЬЗОВАТЬ ФУНКЦИИ? 1](#_Toc146570959)

[2 ОПРЕДЕНИЕ И ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ 3](#_Toc146570960)

[2.1 Определение функций. 3](#_Toc146570961)

[2.2 Прототипы функций. 5](#_Toc146570962)

[2.3 Вызов функции. 5](#_Toc146570963)

[2.4 Глобальные и локальные данные. Параметры. Передача параметров 8](#_Toc146570964)

[2.5 Передача массивов в качестве параметров 12](#_Toc146570965)

[3 СОВЕТЫ ПРОФЕССИОНАЛА 13](#_Toc146570966)

[3.1 Зачем использовать функции? 13](#_Toc146570967)

[3.2 Рекомендации при написании функций: 13](#_Toc146570968)

[3.3 Как правильно использовать параметры? 13](#_Toc146570969)

# ЗАЧЕМ ИСПОЛЬЗОВАТЬ ФУНКЦИИ?

Начинающие программисты часто спрашивают: «А можно ли обходиться без функций и весь код помещать непосредственно в функцию **main()**?». Если вашего кода всего 10-20 строчек, то можно, если же серьёзно, то функции предназначены для упрощения кода, а не для его усложнения.

Функции имеют ряд преимуществ, которые делают их чрезвычайно полезными в нетривиальных программах:

* **структура**. Как только программы увеличиваются в размере/сложности, сохранять весь код внутри **main()** становится трудно. **Функция** — это как мини-программа, которую мы можем записать отдельно от головной программы, не «заморачиваясь» при этом об остальных частях кода. Это позволяет разбивать сложные задачи на более мелкие и простые, что кардинально снижает общую сложность программы;
* **повторное использование**. После объявления функции, её можно вызывать много раз. Это позволяет избежать дублирования кода и сводит к минимуму вероятность возникновения ошибок при копировании/вставке кода. Функции также могут использоваться и в других программах, уменьшая объём кода, который нужно писать с нуля каждый раз;.
* **тестирование**. Поскольку функции убирают лишний код, то и тестировать его становится проще. А так как функция — это самостоятельная единица, то нам достаточно протестировать её один раз, чтобы убедиться в её работоспособности, а затем мы можем её повторно использовать много раз без необходимости проводить тестирование (до тех пор, пока не внесём изменения в эту функцию);
* **модернизация**. Когда нужно внести изменения в программу или расширить её функциональность, то функции являются отличным вариантом. С их помощью можно внести изменения в одном месте, чтобы они работали везде;
* **абстракция**. Для того, чтобы использовать функцию, нам нужно знать её имя, данные ввода, данные вывода и где эта функция находится. Нам не нужно знать, как она работает. Это очень полезно для написания кода, понятного другим (например, стандартная библиотека С++ и всё, что в ней находится, создана по этому принципу).

Каждый раз, при вызове [**std::cin** или **std::cout**](https://ravesli.com/urok-11-sout-cin-i-endl/) для ввода или вывода данных, мы используем функцию из стандартной библиотеки C++, которая соответствует всем указанным выше концепциям.

Для простых программ (менее, чем 30 строчек кода) частично или полностью код получения данных от пользователя, обработки данных и вывода результата можно записать в функции **main()**. Для более сложных программ (или просто для практики) каждый из трёх перечисленных выше процессов является хорошим вариантом для написания отдельной функции.

Новички часто комбинируют обработку ввода и вывод результата в одну функцию, это нарушает **правило «одного задания»**.

# ОПРЕДЕНИЕ И ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

### Определение функций.

Формально определение функции выглядит так:

**тип имя (список\_параметров) //**прототип,заголовок функции

**{ //**реализация функции

**объявление локальных данных; //**могут отсутствовать

**операторы;**

**}**

Локальные данные и операторы инкапсулированы(спрятаны), так что ни локальные данные, ни операторы по отдельности не доступны пользователю функции. Пользователь может только вызвать функцию.

**Вызывающая программа** – это программа (функция), которая обращается к функции (вызываетфункцию).

**Оператор вызова** – оператор вызывающей программы, с помощью которого она обращается к функции.

Оператор вызова содержит:

* имя функции;
* список фактических параметров.

Фактические параметры – это элементы вызывающей программы (константы, переменные, массивы и другие), значения которых передаются в функцию или возвращаются из нее. При вызове функции значения фактических параметров присваиваются соответствующим формальным параметрам (типы фактических и формальных параметров должны совпадать!).

Схема передачи управления между вызывающей программой и функцией приведена на рисунке 2.1.

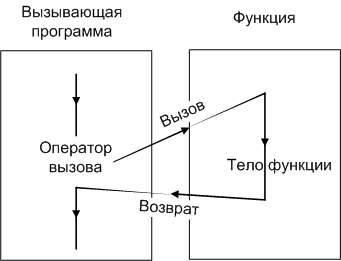


Рисунок 2.1 – Схема передачи управления между вызывающей

программой и функцией

Определим функцию, которая вычисляет сумму двух целых чисел.

**#include <iostream>**

**using namespace std;**

**//функция вычисления суммы двух значений**

**int** **add ( int x, int y)**

**{**

**int** **z;**

**z = x + y;**

**return**  **z;**

**}**

Как видим, определение функции состоит из двух частей: заголовка и тела, которое представляет собой блок инструкций. Заголовок функции включает тип возвращаемого функцией значения, имя функции и список параметров. В нашем случае функция возвращает значение типа **int**, имеет имя **add** и два формальных параметра **x** и **y**. Для вычисления суммы внутри функции объявляется переменная **z**, которая не видна вне функции и поэтому называется локальной переменной. Полученное значение суммы возвращается в вызываемую программу инструкцией **return**.

Эта же функция может быть определена более кратко:

**int** **add ( int x, int y)**

**{**

**return x + y;**

**}**

Если функции не возвращает никакого значения, то в языке С++ считается, что эта функция возвращает значение типа **void**, то есть пустое значение. В этом случае для выхода из функции используется инструкция **return**. Если выход из функции, не возвращающей значения, происходит как выход из блока, то в этом случае инструкцию **return** можно опустить. Например, следующая функция напечатает сумму двух чисел:

**void** **print ( int x, int y)**

**{**

**cout << x + y**;

**}**

Если функция не имеет параметров, то вместо списка параметров пишется ключевое слово **void**, которое можно также опустить. Предпочтительнее использовать первый вариант, так как он явно указывает, что параметров нет.

Например, следующая функция печатает сообщение:

**void**  **hello ( )**

**{**

**cout << "Hello!<<endl;**

**}**

Формальные параметры функции (**передаваемые по значению!**) можно рассматривать как локальные переменные, которые видны только в теле функции. Отсюда следует, что значения формальных параметров можно изменять в теле функции. Например, функция, которая возвращает значение параметра, увеличенное на 1:

**int** **inc ( int n)**

**{**

**return ++n;**

**}**

Заметим, что функция

**int** **inc ( int n)**

**{**

**return n++;**

**}**

вернет значение **n** (почему?)

### Прототипы функций.

Для предварительного описания функции может использоваться её объявление или **прототип**. В отличие от определения функции, прототип содержит только заголовок функции и не включает её тело.

Прототипы функций используются в следующих случаях:

* Функция определена в другом файле,
* Функция определена после обращения к ней,
* Функция определена в библиотеке функций, которая подключается к программе на этапе её компоновки или выполнения.

Например, функция сложения двух чисел имеет следующий прототип:

**int** **add ( int x, int y);**

При объявлении функции в списке параметров можно указывать только типы параметров, опуская их названия. Например, функцию **add** можно объявить следующим образом:

**int** **add ( int, int);**

Заметим, что объявление функции заканчивается символом ‘**;**’

### Вызов функции.

Для вызова функции в программе пишется имя функции, за которым в скобках следует список аргументов, которые являются переменными и константами, определенными в программе. При этом происходит выполнение тела функции, в котором параметры заменяются значениями аргументов. Важно отметить, что тип аргумента должен соответствовать типу соответствующего параметра или допускать приведение к типу параметра.

Функция должна быть объявлена или определена до своего вызова.

**Вариант 1 (в стиле Pascal)**

**#include <iostream>**

**using namespace std;**

**//функция вычисления суммы двух значений**

**int** **add ( int x, int y)**

**{**

**return**  **x + y;**

**}**

**int main()**

**{**

**int** **z;**

**z = add ( 1, 2); // вызов функции add**

**cout << "z= "<< z << endl;**  **// вывод результата вызова функции add**

**}**

**Вариант 2 (в стиле C)**

**#include <iostream>**

**using namespace std;**

**int** **add ( int , int ); //прототип функции вычисления суммы двух значений**

**int main()**

**{**

**int** **z;**

**z = add ( 1, 2); // вызов функции add**

**cout << "z= "<< z<< endl;**  **// вывод результата вызова функции add**

**int** **a=1, b=2;**

**z = add ( a, b);**

**cout << "z= "<<z<<endl;**

**}**

**//функция вычисления суммы двух значений**

**int** **add ( int x, int y)**

**{**

**return**  **x + y;**

**}**

Если функция не имеет параметров, то при её вызове аргументы не указываются, но круглые скобки остаются, например вызов функции **hello**:

**#include <iostream>**

**using namespace std;**

**void hello(); //прототип функции hello**

**int main()**

**{**

**hello();**

**}**

**// функциия hello**

**void**  **hello ( )**

**{**

**cout << "Hello!<<endl;**

**}**

Программа нахождения максимума из трех чисел

**Вариант 1**

**#include <iostream>**

**using namespace std;**

**//функция находит максимум из двух целых чисел**

**int max2(int a, int b) //заголовок функции**

**{ //тело функции**

**if (a>b)**

**return a;//return** – **оператор возврата в вызывающую программу**

**else**

**return b;**

**}**

**int main() // main** –  **это тоже функция**

**{ //использование функции max для нахождения большего из a, b, c**

**int a, b, c, d;**

**cout << "Введите а, b, с:";**

**cin >> a >> b >> c;**

**d = max2(a, b); //вызов функции max с параметрами a, b**

**// d = результат функции max** – **большее из a, b**

**d = max2(d, c); //вызов функции max с параметрами d, c**

**// d = результат функции max** – **большее из d, c**–

**//большее из a, b, c**

**cout << "Максимальное число - " << d<<endl;**

**system("pause");**

**return 0;**

**}**

**Вариант 2**

**#include <iostream>**

**using namespace std;**

**int max2(int, int); //прототип функции: имена параметров можно не указывать**

**int main()**

**{ //использование функции max для нахождения большего из a,b,c**

**int a, b, c, d; //локальные переменные функции main**

**cout << "Введите а, b, с:";**

**cin >> a >> b >> c;**

**d = max2(max2(a,b), c); //вызов функции max как параметр функции**

**//порядок вызовов max как в варианте 1 примера 2\_1**

**cout << "Максимальное число - " << d<<endl;**

**system("pause");**

**return 0;**

**}**

**//функция находит максимум из двух целых чисел**

**int max2(int a, int b) //заголовок функции**

**{ //тело функции**

**int d; //локальная переменная, доступна только в функции max**

**a>b? d=a : d=b;**

**return d;**

**}**

### Глобальные и локальные данные. Параметры. Передача параметров

Обмен информацией между вызывающей программами и функциями может осуществляться с использованием глобальных данных (рисунок 2.2) или передачей параметров.

**Глобальные данные *–*** это данные (константы, переменные, массивы и другие), объявленные в начале программы и доступные в функциях и «главной программе» (функции **main**).

Глобальные данные следует использовать только для передачи информации от одной функции к другой. Например, одна функция формирует значения данных, а другая функция использует эти значения.

**Передача параметров**. Когда вызывающая программа обращается к функции, требуется связать фактические параметры вызывающей программы с формальными параметрами функции. Чаще всего применяются два подхода: либо фактический параметр вычисляется и полученное значение передается формальному параметру (передача значением), либо формальному параметру становится доступен адрес значения фактического параметра (передача по адресу).



Рисунок 2.2 – Глобальные и локальные данные

**Передача значением.** Схема обмена информацией с использованием передачи параметра значением приведена на рисунке 2.3. В этом случае формальный параметр обрабатывается в функции как локальная переменная, инициализируемая в начале выполнения функции значением соответствующего фактического параметра*.*



Рисунок 2.3 – Передача параметра значением

Значение фактического параметра при изменении соответствующего ему формального параметра не изменяется.

В примере на рисунке 2.3 после завершения работы функции **f** значение фактического параметра **x** остается равным**0,** не смотря на оператор **a++** в теле функции **f**.

В операторе вызова фактическим параметром (при передаче параметра значением) может быть константа, переменная, выражение или функция.

**Передача параметров по адресу.** Указатели и ссылки используются для передачи параметров по адресу.Схема обмена информацией вызывающей программы и функции с использованием передачи параметра по адресу приведена на рисунке 2.4. В этом случае формальный параметр **a** обрабатывается в функции как переменная, адрес которой есть адрес соответствующего фактического параметра **x**.

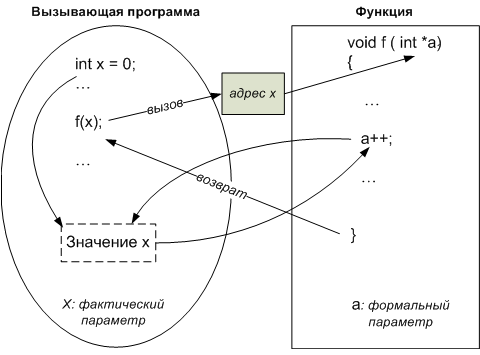


Рисунок 2.4 – Передача параметра по адресу

В операторе вызова фактическим параметром может быть только переменная.

Любое изменение формального параметра **a** есть изменение соответствующего ему фактического параметра **x**.После завершения работы функции f значение **x**равно**1,** так как в функции выполнен оператор **а++.**

Программа перестановки значений

**Вариант 1**

**#include <iostream>**

**using std::cout;**

**using std::endl;**

**void swap\_values(int\* a, int\* b) // формальный параметр - указатель**

**{**

**int temp=\*a; // требуется разыменование указателей**

**\*a=\*b;**

**\*b=temp;**

**}**

**int main()**

**{**

**int x=10, y=20;**

**swap\_values(&x, &y); // требуется операция & , так как**

**// параметры передаются по адресу**

**cout<<x<<" "<<y<<endl; // результат: 20 10**

**return 0;**

**}**

**Вариант 2**

**#include <iostream>**

**using std::cout;**

**using std::endl;**

**void swap\_values(int& a, int& b) // формальный параметр - ссылка**

**{**

**int temp=a; // НЕ требуется разыменование указателей**

**a=b;**

**b=temp;**

**}**

**int main()**

**{**

**int x=10, y=20;**

**swap\_values(x, y); // НЕ требуется операция &**

**cout<<x<<" "<<y<<endl; // результат: 20 10**

**return 0;**

**}**

### Передача массивов в качестве параметров

Во многих случаях передавать значения в функции выгоднее по адресу, а не по значению. Особенно эффективна передача по адресу, если передаваемые данные имеют большой размер, например массивы данных. Однако, при этом досаточно часто требуется защитить передаваемые данные от изменения. Для этого в прототипе функции используется неконстантный указатель на константные данные.

Программа с передачей массива в качестве параметра

**Пример 1. Передача массива по адресу с указанием его размера**

**#include <iostream>**

**using std::endl;**

**using std::cout;**

**int summa(const int\* x, const int n); // массив всегда передается по**

**//адресу, необходимо передать и размер массива**

**int main() {**

**const int ArraySize=5; // размер массива**

**int a[ArraySize] = {1, 3, 4, 5, 2}; // инициализация массива**

**for (int i=0; i<ArraySize;++i) // вывод значений элементов**

**cout<<a[i]<<endl; // массива**

**cout << "Сумма элементов массива: "<<summa(a, ArraySize)<<endl;**

**return 0;**

**}**

**int summa(const int\* x, const int n)**

**// можно и так: int summa(const int x[], const int n)**

**{**

**int total=0; // сумма значений элементов массива**

**for (int i=0; i<n; ++i) // цикл вычисления суммы**

**total+=x[i];**

**return total;**

**}**

**Пример 2. Передача массива с использованием фунций std::begin(), std::end()**

**#include <iostream>**

**void print(int\*, int\*);**

**int main()**

**{**

**int nums[]{ 1, 2, 3, 4, 5 };**

**int\* begin{ std::begin(nums) };**

**int\* end{ std::end(nums) };**

**print(begin, end);**

**}**

**void print(int\* begin, int\* end)**

**{**

**for (int\* ptr{ begin }; ptr != end; ptr++)**

**{**

**std::cout << \*ptr << "\t";**

**}**

**}**

**Пример 3. Передача массива по ссылке**

**#include <iostream>**

**void print(const int(&)[5]); // массив строго с 5 элементами**

**int main()**

**{**

**int nums1[]{ 1, 2, 3, 4, 5 };**

**print(nums1);**

**}**

**void print(const int(&numbers)[5])**

**{**

**for (unsigned i{}; i < 5; i++)**

**{**

**std::cout << numbers[i] << "\t";**

**}**

**}**

# СОВЕТЫ ПРОФЕССИОНАЛА

**IconAttention**

### Зачем использовать функции?

* Самая важная причина – снижение сложности программ.
* Минимизация размера кода программы.
* Предотвращение дублирования кода.
* Облегчение сопровождения программы.
* Снижение числа ошибок.

### Рекомендации при написании функций:

* Код, который появляется более одного раза в программе, лучше переписать в виде функции. Например, если мы получаем данные от пользователя несколько раз одним и тем же способом, то это отличный вариант для написания отдельной функции.
* Код, который используется для сортировки чего-либо, лучше записать в виде отдельной функции. Например, если у нас есть список вещей, которые нужно отсортировать — пишем функцию сортировки, куда передаём несортированный список и откуда получаем отсортированный.
* Функция должна выполнять одно (и только одно) задание.
* Когда функция становится слишком большой, сложной или непонятной – её следует разбить на несколько подфункций. Это называется **рефакторинг** кода.

### Как правильно использовать параметры?

* Передавайте параметры в следующем порядке: входные значения – изменяемые значения – выходные значения.
* Передавайте переменные состояния или кода ошибки последними.
* Не используйте параметры функции в качестве рабочих переменных.
* Ограничивайте число параметров примерно семью.
* Убедитесь, что фактические параметры соответствуют формальным.
* Используйте глобальные данные только в том случае, если вы не можете без них обойтись.