

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)


## Урок №47. Блоки стейтментов (составные операторы)

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 11 Сен 2020 |

 23204

[1](#)  [3](#)

На этом уроке мы рассмотрим блоки стейтментов в языке C++.

Оглавление:

1. [Блоки стейтментов](#)
2. [Вложенные блоки](#)
3. [Блоки и операторы if](#)
4. [Количество уровней вложенности блоков](#)
5. [Заключение](#)

### Блоки стейтментов

**Блоки стейтментов** (или «*составные операторы*») — это группа стейтментов, которые обрабатываются компилятором как одна инструкция. Блок начинается с символа { и заканчивается символом }, стейтменты находятся внутри. Блоки могут использоваться в любом месте, где разрешено использовать один стейтмент. В конце составного оператора точка с запятой не ставится.

Вы уже видели пример блоков при написании функций, поскольку тело функции является блоком:

```
1 int add(int x, int y)
2 { // начало блока
3     return x + y;
4 } // конец блока
5
6 int main()
7 { // начало блока
8
9     // Несколько стейтментов
```

```
10 | int value(0);
11 | add(3, 4);
12 |
13 | return 0;
14 |
15 | } // конец блока (без точки с запятой)
```

## Вложенные блоки

Хотя функции не могут быть вложены в другие функции, блоки *могут* быть вложены в другие блоки:

```
1 | int add(int x, int y)
2 | { // начало блока
3 |     return x + y;
4 | } // конец блока
5 |
6 | int main()
7 | { // начало внешнего блока
8 |
9 |     // Несколько стейтментов
10 |    int value {};
11 |
12 |    { // начало внутреннего/вложенного блока
13 |        add(3, 4);
14 |    } // конец внутреннего/вложенного блока
15 |
16 |    return 0;
17 |
18 | } // конец внешнего блока
```

При использовании вложенных блоков, блок, который содержит внутри себя другой блок, называется **внешним блоком**, а тот, который содержится внутри этого блока — **внутренний/вложенный блок**.

## Блоки и операторы if

Один из наиболее распространенных вариантов использования блоков связан с [операторами if](#). По умолчанию оператор if выполняет один стейтмент, если условие имеет значение true. С помощью блока мы можем сделать так, чтобы выполнялось сразу несколько стейтментов, если условие имеет значение true, например:

```
1 | #include <iostream>
2 |
3 | int main()
4 | {
5 |     std::cout << "Enter an integer: ";
6 |     int value;
7 |     std::cin >> value;
8 |
9 |     if (value >= 0)
```

```
10 { // начало вложенного блока
11     std::cout << value << " is a positive integer (or zero)" << std::endl;
12     std::cout << "Double this number is " << value * 2 << std::endl;
13 } // конец вложенного блока
14 else
15 { // начало другого вложенного блока
16     std::cout << value << " is a negative integer" << std::endl;
17     std::cout << "The positive of this number is " << -value << std::endl;
18 } // конец другого вложенного блока
19
20 return 0;
21 }
```

Если ввести число 3, то программа выведет:

```
Enter an integer: 3
3 is a positive integer (or zero)
Double this number is 6
```

Если ввести число -4, то программа выведет:

```
Enter an integer: -4
-4 is a negative integer
The positive of this number is 4
```

## Количество уровней вложенности блоков

Можно даже размещать вложенные блоки внутри других вложенных блоков:

```
1  #include <iostream>
2
3  int main()
4  { // 1-й уровень вложенности блоков
5      std::cout << "Enter an integer: ";
6      int value {};
7      std::cin >> value;
8
9      if (value > 0)
10     { // 2-й уровень вложенности блоков
11         if ((value % 2) == 0)
12         { // 3-й уровень вложенности блоков
13             std::cout << value << " is positive and even\n";
14         }
15         else
16         { // также 3-й уровень вложенности блоков
17             std::cout << value << " is positive and odd\n";
18         }
19     }
20
21     return 0;
```

22 | }

**Уровень вложенности функции** (или «*глубина вложенности функции*») — это максимальное количество блоков, которые могут находиться в любой точке функции (включая внешний блок). В вышеприведенной функции есть 4 блока, но уровень вложенности равен 3.

По факту, ограничений на количество вложенных блоков нет. Однако не рекомендуется делать больше 3-х уровней вложенности (максимум 4). Если ваша функция нуждается в большем количестве уровней вложенности, то эту функцию лучше разбить на несколько подфункций!

## Заключение

Блоки стейтментов позволяют выполнить сразу несколько стейтментов там, где можно использовать лишь один. Они чрезвычайно полезны, когда нужно выполнить сразу несколько инструкций вместе.

Оценить статью:

★★★★★ (261 оценок, среднее: 4,94 из 5)

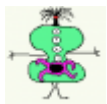


[← Глава №3. Итоговый тест](#)

[Урок №48. Локальные переменные, область видимости и продолжительность жизни](#)



## Комментариев: 3



1. Александр:

[26 июля 2018 в 11:05](#)

Как будь-то если написать

```
1 { // начало вложенного блока
2     std::cout << value << " is a positive integer (or zero)" << std::endl;
3     std::cout << "Double this number is " << value * 2 << std::endl;
4 }
```

без этих скобок — ничего не заработает..

[Ответить](#)



1. Владимир:

[28 ноября 2018 в 23:21](#)

Заработать-то заработает, да вот только не так, как хотел бы программист.

[Ответить](#)1. *Алексей:*[9 июля 2019 в 14:32](#)

Дело совсем не в том, что "а вот только не так, как хотел бы программист".  
Тут легко запутаться при случаи большой проги.

Скажем таких блоков 500, вложены по 3-4.

Догадайся, что это скобка закрывает. Не догадаешься, писать надо в один ряд их.

[Ответить](#)

## Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены \*

Имя \*

Email \*

Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию





☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

TELEGRAM  КАНАЛ



ПАБЛИК 

## ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «Same Game»](#)

-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020