

Ravesli [Ravesli](#)


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


## Урок №43. Логические операторы: И, ИЛИ, НЕ

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 21 Сен 2020 |

 76252

[14](#)

На этом уроке мы рассмотрим логические операторы И, ИЛИ и НЕ в языке C++.

Оглавление:

1. [Логические операторы](#)
2. [Логический оператор НЕ](#)
3. [Логический оператор ИЛИ](#)
4. [Логический оператор И](#)
5. [Короткий цикл вычислений](#)
6. [Использование логических операторов И/ИЛИ](#)
7. [Законы Де Моргана](#)
8. [А где же побитовое исключающее ИЛИ \(XOR\)?](#)
9. [Тест](#)

## Логические операторы

В то время как [операторы сравнения](#) используются для проверки конкретного условия: ложное оно или истинное, они могут проверить только одно условие за определенный промежуток времени. Но бывают ситуации, когда нужно протестировать сразу несколько условий. Например, чтобы узнать, выиграли ли мы в лотерею, нам нужно сравнить все цифры купленного билета с выигрышными. Если в лотерее 6 цифр, то нужно выполнить 6 сравнений, все из которых должны быть true.

Также иногда нам нужно знать, является ли хоть одно из нескольких условий истинным. Например, мы не пойдем сегодня на работу, если больны или слишком устали, или если выиграли в лотерею. 😊 Нам нужно проверить, является ли хоть одно из этих 3-х условий истинным. Как это сделать? С помощью логических операторов! Они позволяют проверить сразу несколько условий за раз.

В языке C++ есть 3 логических оператора:

Оператор	Символ	Пример	Операция
Логическое НЕ	!	!x	true, если x — false и false, если x — true
Логическое И	&&	x && y	true, если x и y — true, в противном случае — false
Логическое ИЛИ		x    y	true, если x или y — true, в противном случае — false

## Логический оператор НЕ

Мы уже с ним сталкивались на [уроке №34](#).

### Логический оператор НЕ (!)

Операнд	Результат
true	false
false	true

Если операндом является true, то, после применения логического НЕ, результатом будет false. Если же операнд до применения оператора НЕ был false, то после его применения станет true. Другими словами, логический оператор НЕ меняет результат на противоположный начальному значению. Он часто используется в условных выражениях:

```
1 bool bTooLarge = (x > 100); // переменная bTooLarge будет true, если x > 100
2 if (!bTooLarge)
3     // Делаем что-нибудь с x
4 else
5     // Выводим ошибку
```

Следует помнить, что логический оператор НЕ имеет очень высокий [уровень приоритета](#). Новички часто совершают следующую ошибку:

```
1 #include <iostream>
2
3 int main()
4 {
5     int x = 5;
6     int y = 7;
7
8     if (!x == y)
9         std::cout << "x does not equal y";
10    else
11        std::cout << "x equals y";
12
13    return 0;
14 }
```

Результат выполнения программы:

x equals y

Но  $x$  ведь не равно  $y$ , как это возможно? Поскольку приоритет логического оператора НЕ выше, чем приоритет оператора равенства, то выражение  $! x == y$  обрабатывается как  $(! x) == y$ . Так как  $x$  — это 5, то  $!x$  — это 0. Условие  $0 == y$  ложное, поэтому выполняется часть `else`!

**Напоминание:** Любое ненулевое целое значение в логическом контексте является `true`. Так как  $x = 5$ , то  $x$  вычисляется как `true`, а вот  $!x = \text{false}$ , т.е. 0. Использование целых чисел в логических операциях подобным образом может запутать не только пользователя, но и самого разработчика, поэтому так не рекомендуется делать!

Правильный способ написания программы, приведенной выше:

```
1 #include <iostream>
2
3 int main()
4 {
5     int x = 5;
6     int y = 7;
7
8     if (!(x == y))
9         std::cout << "x does not equal y";
10    else
11        std::cout << "x equals y";
12
13    return 0;
14 }
```

Сначала обрабатывается  $x == y$ , а затем уже оператор НЕ изменяет результат на противоположный.

**Правило:** Если логический оператор НЕ должен работать с результатами работы других операторов, то другие операторы и их операнды должны находиться в круглых скобках.

## Логический оператор ИЛИ

Если хоть одно из двух условий является истинным, то логический оператор ИЛИ является `true`.

### Логический оператор ИЛИ (||)

Левый операнд Правый операнд Результат

false	false	false
false	true	true
true	false	true
true	true	true

Рассмотрим следующую программу:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter a number: ";
6     int value;
7     std::cin >> value;
```

```
8
9     if (value == 0 || value == 1)
10         std::cout << "You picked 0 or 1" << std::endl;
11     else
12         std::cout << "You did not pick 0 or 1" << std::endl;
13
14     return 0;
15 }
```

Здесь мы использовали логический оператор ИЛИ, чтобы проверить, является ли хоть одно из двух условий истинным: левое (`value == 0`) или правое (`value == 1`). Если хоть одно из условий — `true` или оба сразу `true`, то выполняться будет стейтмент `if`. Если ни одно из условий не является `true`, то результат — `false` и выполняться будет стейтмент `else`.

Вы можете связать сразу несколько условий:

```
1 if (value == 0 || value == 1 || value == 2 || value == 3)
2     std::cout << "You picked 0, 1, 2, or 3" << std::endl;
```

Новички иногда путают логическое ИЛИ (`| |`) с побитовым ИЛИ (`|`). Хотя у них и одинаковые названия, функции они выполняют разные.

## Логический оператор И

Только при условии, что оба операнда будут истинными, логический оператор И будет `true`. Если нет, тогда — `false`.

### Логический оператор И (&&)

#### Левый операнд Правый операнд Результат

false	false	false
false	true	false
true	false	false
true	true	true

Например, мы хотим узнать, находится ли значение переменной `x` в диапазоне от 10 до 20. Здесь у нас есть два условия: мы должны проверить, является ли `x` больше 10 и является ли `x` меньше 20.

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter a number: ";
6     int value;
7     std::cin >> value ;
8
9     if (value > 10 && value < 20)
10         std::cout << "Your value is between 10 and 20" << std::endl;
11     else
12         std::cout << "Your value is not between 10 and 20" << std::endl;
```

```
13 |  
14 |     return 0;  
15 | }
```

Если оба условия истинны, то выполняется часть `if`. Если же хоть одно или сразу оба условия ложные, то выполняется часть `else`.

Как и с логическим ИЛИ, мы можем комбинировать сразу несколько условий И:

```
1 | if (value > 10 && value < 20 && value != 16)  
2 |     // Делаем что-нибудь  
3 | else  
4 |     // Делаем что-нибудь другое
```

## Короткий цикл вычислений

Для того, чтобы логическое И возвращало `true`, оба операнда должны быть истинными. Если первый операнд вычисляется как `false`, то оператор И должен сразу возвращать `false` независимо от результата второго операнда (даже без его обработки). Это называется **коротким циклом вычисления** (англ. *«short circuit evaluation»*) и выполняется он, в первую очередь, в целях оптимизации.

Аналогично, если первый операнд логического ИЛИ является `true`, то и всё условие будет `true` (даже без обработки второго операнда).

Как и в случае с оператором ИЛИ, новички иногда путают логическое И (`&&`) с побитовым И (`&`).

## Использование логических операторов И/ИЛИ

Иногда возникают ситуации, когда смешивания логических операторов И и ИЛИ в одном выражении не избежать. Тогда следует знать о возможных проблемах, которые могут произойти.

Многие программисты думают, что логические И и ИЛИ имеют одинаковый приоритет (или забывают, что это не так), так же как и сложение/вычитание или умножение/деление. Тем не менее, приоритет логического И выше приоритета ИЛИ. Таким образом, операции с оператором И всегда будут вычисляться первыми (если только операции с ИЛИ не находятся в круглых скобках).

Рассмотрим следующее выражение: `value1 || value2 && value3`. Поскольку приоритет логического И выше, то обрабатываться выражение будет так:

```
value1 || (value2 && value3)
```

А не так:

```
(value1 || value2) && value3
```

Хорошей практикой является использование круглых скобок с операциями. Это предотвратит ошибки приоритета, увеличит читабельность кода и чётко даст понять компилятору, как следует обрабатывать выражения. Например, вместо того, чтобы писать `value1 && value2 || value3 && value4`, лучше записать `(value1 && value2) || (value3 && value4)`.

## Законы Де Моргана

Многие программисты совершают ошибку, думая, что `!(x && y)` — это то же самое, что и `!x && !y`. К сожалению, вы не можете использовать логическое НЕ подобным образом.

[Законы Де Моргана](#) гласят, что `!(x && y)` эквивалентно `!x || !y`, а `!(x || y)` эквивалентно `!x && !y`.

Другими словами, логические операторы И и ИЛИ меняются местами! В некоторых случаях, это даже полезно, так как улучшает читабельность.

## А где же побитовое исключающее ИЛИ (XOR)?

Побитовое исключающее ИЛИ (XOR) — это логический оператор, который используется в некоторых языках программирования для проверки на истинность нечётного количества условий.

### Побитовое исключающее ИЛИ (XOR)

Левый операнд Правый операнд Результат

false	false	false
false	true	true
true	false	true
true	true	false

В языке C++ нет такого оператора. В отличие от логических И/ИЛИ, к XOR не применяется короткий цикл вычислений. Однако его легко можно симитировать, используя оператор неравенства (`!=`):

```
1 | if (a != b) ... // a XOR b (предполагается, что a и b имеют тип bool)
```

Можно также расширить количество операндов:

```
1 | if (a != b != c != d) ... // a XOR b XOR c XOR d (предполагается, что a, b, c и d имеют тип bool)
```

Следует отметить, что вышеприведенные шаблоны XOR работают только, если операнды имеют логический (а не целочисленный) тип данных. Если вы хотите, чтобы это работало и с целыми числами, то используйте [оператор static\\_cast](#).

Форма XOR, которая работает и с другими типами данных (с помощью оператора `static_cast` мы можем конвертировать любой тип данных в тип `bool`):

```
1 | if (static_cast<bool>(a) != static_cast<bool>(b) != static_cast<bool>(c) != static_cast<bool>(d)) ...
```

## Тест

Какой результат следующих выражений?

→ Выражение №1: `(true && true) || false`

→ Выражение №2: `(false && true) || true`

→ Выражение №3: `(false && true) || false || true`

→ *Выражение №4:* `(5 > 6 || 4 > 3) && (7 > 8)`

→ *Выражение №5:* `!(7 > 6 || 3 > 4)`

## Ответ

**Примечание:** В ответах объяснение выполняется с помощью стрелочки ( $\Rightarrow$ ). Например, `(true || false) => true` означает, что результатом выражения `(true || false)` является `true`.

→ *Выражение №1:* `(true && true) || false => true || false => true`

→ *Выражение №2:* `(false && true) || true => false || true => true`

→ *Выражение №3:* `(false && true) || false || true => false || false || true => false || true => true`

→ *Выражение №4:* `(5 > 6 || 4 > 3) && (7 > 8) => (false || true) && false => true && false => false`

→ *Выражение №5:* `!(7 > 6 || 3 > 4) => !(true || false) => !true => false`

Оценить статью:

★★★★★ (311 оценок, среднее: 4,91 из 5)



← [Урок №42. Операторы сравнения](#)

[Урок №44. Конвертация чисел из двоичной системы в десятичную и наоборот](#) →



## Комментариев: 14



1. *Artur:*

[12 июля 2020 в 00:02](#)

спасибо огромное!  
очень понятно все изложено!

[Ответить](#)



2. *Nick:*

[21 февраля 2020 в 14:45](#)

Подскажите пожалуйста, в соответствии со стандартом C++, оператор `&&` всегда вычисляет оба операнда или только если первый из них возвращает `true`?

В выражении:

```
1 | if(f1() && f2()){....}
```

будут ли всегда выполнены обе функции f1() и f2()?

[Ответить](#)



1. *Nick:*

[24 февраля 2020 в 11:49](#)

Нашёл ответ:

если f1() или f2() возвращает false, то будет выполнена только одна из функций, но какая именно стандартом не регламентируется

[Ответить](#)



2. *Сергей:*

[16 марта 2020 в 00:25](#)

в данном выражении обе функции выполняться только в случае если первая будет true

[Ответить](#)



1. *Nick:*

[12 июля 2020 в 10:51](#)

Нет, порядок выполнения операндов в операторах в C и C++ не регламентирован. Может вначале выполниться вторая функция и если она вернёт false, первая не выполнится. Но может и в обратном порядке. Отсюда можно словить интересные побочные эффекты (side effects), поэтому в этих языках программирования вызываемые в операторах функции не должны иметь влияния друг на друга.

Другое дело, например bash или Perl и другие языки где порядок выполнения операндов в операторах регламентирован (в основном слева направо). Там действительно всегда сначала вычисляется левый операнд, а потом, если необходимо, правый

[Ответить](#)



1. *Даниил:*

[5 октября 2020 в 16:27](#)

Чувак, не вводи в заблуждение:

§5.14/1:

The && operator groups left-to-right. The operands are both contextually converted to type bool (Clause 4). The result is true if both operands are true and false otherwise.

Unlike &, && guarantees left-to-right evaluation: the second operand is not evaluated if the first operand is false.

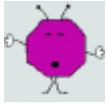


3. *AleksTs:*[4 ноября 2019 в 01:53](#)

Не перестаю восхищаться изложенному материалу! Огромное спасибо за проделанную работу!

[Ответить](#)1. *Юрий:*[4 ноября 2019 в 11:35](#)

Пожалуйста)) Мне приятно)

[Ответить](#)4. *Александр:*[1 февраля 2019 в 10:08](#)

Про рекомендацию использовать скобки, если сомневаетесь в приоритете:

Страуструп крайне не рекомендует "украшать" код скобками... лучше или разобраться в приоритетах или разбить выражение на несколько, если есть хоть какие-то сомнения.

Вместо:

```
1 | a || (b && c)
```

лучше написать:

```
1 | b1 = b && c
2 | a || b1
```

[Ответить](#)5. *Аркадий:*[3 августа 2018 в 07:33](#)

"Для реализации оператора XOR" используют ^. В c++ — это именно XOR, а не возведение в степень.

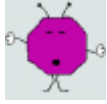
[Ответить](#)1. *Tosha:*[30 января 2019 в 17:26](#)

^ — это битовый оператор, а данная статья — о логических. Логического XOR в плюсах нет.

Для побитового исключения используется именно ^. Для реализации же логического XOR использовать ^ крайне глупо потому что:

1. Сравнивая булевы значения будет очевидней(и эффективней, хоть эти наносекунды уже никого не интересуют) написать `true != false`, чем `true ^ false`.
2. В случае сравнения не булевых значений стоит продолжать существующую логику. На

примере чисел: любое число, отличное от нуля — это true. Следовательно нельзя просто написать  $1 \wedge 2$  — потому что эта операция вернет 3, что интерпретируется, как true. Почему? Потому что любое не нулевое число должно восприниматься, как истина, следовательно  $1 \rightarrow \text{true}$ ,  $2 \rightarrow \text{true}$ . результатом XOR между ними должен быть false. Таким образом доказываем неприменимость  $\wedge$ , приходим к необходимости преобразования операндов в булевы значения и смотрим пункт 1.

[Ответить](#)

2. Александр:

[1 февраля 2019 в 10:01](#)

$\wedge$  — это побитовый оператор... ожидать от него поведения логического XOR — нарываться на потенциальные алгоритмические ошибки...

можно в качестве XOR использовать оператор  $!=$  для логических выражений (тоже некорректно работает, если Вы ожидаете неявных преобразований типов)

[Ответить](#)

6. Светлана:

[11 июня 2018 в 22:43](#)

Скажите, пожалуйста, почему в примере из урока нужно использовать  $\text{if} (!(x == y))$ , а не  $\text{if} (x != y)$  ?

[Ответить](#)

1. Юрий:

[14 июня 2018 в 22:40](#)

Можно использовать и второй вариант, но  $!=$  зачастую используется для реализации оператора XOR и в цепочке операндов, а  $!()$  для проверки на неравенство определенного выражения. Но использовать можно как первый, так и второй вариант — работать будет.

[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.

Отправить комментарий






TELEGRAM  КАНАЛ

Электронная почта



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