

Ravesli [Ravesli](#)


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

## Урок №42. Операторы сравнения

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 21 Сен 2020 |

 46067

[1](#)  [19](#)

В языке C++ есть 6 операторов сравнения:

| Оператор         | Символ | Пример                 | Операция  |
|------------------|--------|------------------------|---|
| Больше           | >      | <code>x &gt; y</code>  | true, если x больше y, в противном случае — false       |
| Меньше           | <      | <code>x &lt; y</code>  | true, если x меньше y, в противном случае — false       |
| Больше или равно | >=     | <code>x &gt;= y</code> | true, если x больше/равно y, в противном случае — false |
| Меньше или равно | <=     | <code>x &lt;= y</code> | true, если x меньше/равно y, в противном случае — false |
| Равно            | ==     | <code>x == y</code>    | true, если x равно y, в противном случае — false        |
| Не равно         | !=     | <code>x != y</code>    | true, если x не равно y, в противном случае — false     |

Вы уже могли их видеть в коде. Они довольно простые. Каждый из этих операторов вычисляется в логическое значение true (1) или false (0).

Вот несколько примеров использования этих операторов на практике:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter an integer: ";
6     int x;
7     std::cin >> x;
8
9     std::cout << "Enter another integer: ";
10    int y;
11    std::cin >> y;
```

```
12
13     if (x == y)
14         std::cout << x << " equals " << y << "\n";
15     if (x != y)
16         std::cout << x << " does not equal " << y << "\n";
17     if (x > y)
18         std::cout << x << " is greater than " << y << "\n";
19     if (x < y)
20         std::cout << x << " is less than " << y << "\n";
21     if (x >= y)
22         std::cout << x << " is greater than or equal to " << y << "\n";
23     if (x <= y)
24         std::cout << x << " is less than or equal to " << y << "\n";
25
26     return 0;
27 }
```

Результат выполнения программы:

```
Enter an integer: 4
Enter another integer: 5
4 does not equal 5
4 is less than 5
4 is less than or equal to 5
```

Всё просто!

## Сравнение чисел типа с плавающей точкой

Сравнение значений типа с плавающей точкой с помощью любого из этих операторов — дело опасное. Почему? Из-за тех самых небольших ошибок округления, которые могут привести к неожиданным результатам. Например:

```
1  #include <iostream>
2
3  int main()
4  {
5      double d1(100 - 99.99); // ДОЛЖНО БЫТЬ 0.01
6      double d2(10 - 9.99); // ДОЛЖНО БЫТЬ 0.01
7
8      if (d1 == d2)
9          std::cout << "d1 == d2" << "\n";
10     else if (d1 > d2)
11         std::cout << "d1 > d2" << "\n";
12     else if (d1 < d2)
13         std::cout << "d1 < d2" << "\n";
14
15     return 0;
16 }
```

Вот так раз:

```
d1 > d2
```

В вышеприведенной программе,  $d1 = 0.01000000000000005116$ , а  $d2 = 0.00999999999999997868$ . Оба этих числа очень близки к  $0.1$ , но  $d1$  больше  $d2$ . Они не равны.

Иногда сравнение чисел типа с плавающей точкой бывает неизбежным. В таком случае следует использовать операторы  $>$ ,  $<$ ,  $>=$  и  $<=$  только если значения не очень близки. А вот если два операнда очень близки значениями, то результат уже может быть неожиданный. В вышеприведенном примере последствия неправильного результата незначительны, а вот с оператором равенства дела обстоят хуже, так как даже при самой маленькой неточности результат сразу меняется на противоположный ожидаемому. Не рекомендуется использовать операторы  $==$  или  $!=$  с числами типа с плавающей точкой. Вместо них следует использовать функцию, которая вычисляет, насколько *близки* эти два значения. Если они «достаточно близки», то мы считаем их равными. Значение, используемое для представления термина «достаточно близки», называется **эпсилоном**. Оно, обычно, небольшое (например,  $0.0000001$ ).

Очень часто начинающие разработчики пытаются писать свои собственные функции определения равенства чисел:

```
1 #include <cmath> // для функции fabs()
2
3 bool isAlmostEqual(double a, double b, double epsilon)
4 {
5     // Если разница между a и b меньше значения эпсилонa, то тогда a и b - "достаточно близки"
6     return fabs(a - b) <= epsilon;
7 }
```

**Примечание:** Функция `fabs()` — это функция из **заголовочного файла** `cmath`, которая возвращает абсолютное значение параметра. `fabs(a - b)` возвращает положительное число, как разницу между  $a$  и  $b$ .

Функция `isAlmostEqual()` из примера, приведенного выше, сравнивает разницу ( $a - b$ ) и эпсилон, вычисляя, таким образом, *близость* чисел. Если  $a$  и  $b$  достаточно близки, то функция возвращает `true`.

Хоть это и рабочий вариант, но он не идеален. Эпсилон  $0.000001$  подходит для чисел около  $1.0$ , но будет слишком большим для чисел типа  $0.0000001$  и слишком малым для чисел типа  $10000$ . Это означает, что каждый раз, при вызове функции, нам нужно будет выбирать наиболее соответствующий входным данным функции эпсилон.

Дональд Кнут, известный учёный, предложил следующий способ в своей книге «Искусство программирования, том 2: Получисленные алгоритмы» (1968):

```
1 #include <cmath> // для функции fabs()
2
3 // Возвращаем true, если разница между a и b в пределах процента эпсилонa
4 bool approximatelyEqual(double a, double b, double epsilon)
5 {
6     return fabs(a - b) <= ( (fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * epsilon);
7 }
```

Здесь, вместо использования эпсилонa как абсолютного числа, мы используем его как множитель, чтобы подстроиться под входные данные.

Рассмотрим детально, как работает функция `approximatelyEqual()`. Слева от оператора `<=` абсолютное значение ( $a - b$ ) сообщает нам разницу между  $a$  и  $b$  (положительное число). Справа от `<=` нам нужно вычислить эpsilon, т.е. наибольшее значение «близости чисел», которое мы готовы принять. Для этого алгоритм выбирает большее из чисел  $a$  и  $b$  (как приблизительный показатель общей величины чисел), а затем умножает его на эpsilon. В этой функции эpsilon представляет собой процентное соотношение. Например, если термин «достаточно близко» означает, что  $a$  и  $b$  находятся в пределах 1% разницы (больше или меньше), то мы вводим эpsilon 1% ( $1\% = 1/100 = 0.01$ ). Его значение можно легко регулировать, в зависимости от обстоятельств (например,  $0.01\% =$  эpsilon  $0.0001$ ). Чтобы сделать неравенство (`!=`) вместо равенства — просто вызовите эту функцию, используя логический оператор НЕ (`!`), чтобы *перевернуть* результат:

```
1 if (!approximatelyEqual(a, b, 0.001))
2     std::cout << a << " is not equal to " << b << "\n";
```

Но и функция `approximatelyEqual()` тоже не идеальна, особенно, когда дело доходит до чисел, близких к нулю:

```
1 #include <iostream>
2 #include <cmath> // для функции fabs()
3
4 // Возвращаем true, если разница между a и b в пределах процента эpsilon
5 bool approximatelyEqual(double a, double b, double epsilon)
6 {
7     return fabs(a - b) <= ((fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * epsilon);
8 }
9
10 int main()
11 {
12     // Значение a очень близкое к 1.0, но, из-за ошибок округления, чуть меньше 1.0
13     double a = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
14
15     // Во-первых, давайте сравним значение a (почти 1.0) с 1.0
16     std::cout << approximatelyEqual(a, 1.0, 1e-8) << "\n";
17
18     // Во-вторых, давайте сравним значение a - 1.0 (почти 0.0) с 0.0
19     std::cout << approximatelyEqual(a - 1.0, 0.0, 1e-8) << "\n";
20 }
```

Возможно, вы удивитесь, но результат:

```
1
0
```

Второй вызов не сработал так, как ожидалось. Математика просто ломается, когда дело доходит до нулей.

Но и этого можно избежать, используя как абсолютный эpsilon (то, что мы делали в первом способе), так и относительный (способ Кнута) вместе:

```
1 // Возвращаем true, если разница между a и b меньше absEpsilon или в пределах relEpsilon
2 bool approximatelyEqualAbsRel(double a, double b, double absEpsilon, double relEpsilon)
3 {
4     // Проверяем числа на их близость - это нужно в тех случаях, когда сравниваемые
```

```

5   double diff = fabs(a - b);
6   if (diff <= absEpsilon)
7       return true;
8
9   // В противном случае, возвращаемся к алгоритму Кнута
10  return diff <= ( (fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * relEpsilon);
11 }

```

Здесь мы добавили новый параметр — `absEpsilon`. Сначала мы сравниваем `a` и `b` с `absEpsilon`, который должен быть задан как очень маленькое число (например, `1e-12`). Таким образом, мы решаем случаи, когда `a` и `b` — нулевые значения или близки к нулю. Если это не так, то мы возвращаемся к алгоритму Кнута.

Протестируем:

```

1  #include <iostream>
2  #include <cmath> // для функции fabs()
3
4  // Возвращаем true, если разница между a и b в пределах процента эпсилон
5  bool approximatelyEqual(double a, double b, double epsilon)
6  {
7      return fabs(a - b) <= ((fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * epsilon);
8  }
9
10 // Возвращаем true, если разница между a и b меньше absEpsilon или в пределах relEpsilon
11 bool approximatelyEqualAbsRel(double a, double b, double absEpsilon, double relEpsilon)
12 {
13     // Проверяем числа на их близость - это нужно в случаях, когда сравниваемые числа
14     double diff = fabs(a - b);
15     if (diff <= absEpsilon)
16         return true;
17
18     // В противном случае, возвращаемся к алгоритму Кнута
19     return diff <= ((fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * relEpsilon);
20 }
21
22 int main()
23 {
24     // Значение a очень близко к 1.0, но, из-за ошибок округления, чуть меньше 1.0
25     double a = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
26
27     std::cout << approximatelyEqual(a, 1.0, 1e-8) << "\n"; // сравниваем "почти
28     std::cout << approximatelyEqual(a - 1.0, 0.0, 1e-8) << "\n"; // сравниваем "почти
29     std::cout << approximatelyEqualAbsRel(a - 1.0, 0.0, 1e-12, 1e-8) << "\n"; // сре
30 }

```

Результат:

```

1
0
1

```

С удачно подобранным `absEpsilon`, функция `approximatelyEqualAbsRel()` обрабатывает близкие к нулю и нулевые значения корректно.

Сравнение чисел типа с плавающей точкой — сложная тема, и нет одного идеального алгоритма, который подойдет в любой ситуации. Однако для большинства случаев с которыми вы будете сталкиваться, функции `approximatelyEqualAbsRel()` должно быть достаточно.

Оценить статью:

 (260 оценок, среднее: 4,79 из 5)



[← Урок №41. Условный тернарный оператор, оператор sizeof и Запятая](#)



[Урок №43. Логические операторы: И, ИЛИ, НЕ →](#)

## Комментариев: 19



1. Андрей:

[26 ноября 2019 в 17:20](#)

А почему нельзя взять за вычисляемый эпсилон среднее арифметическое абсолютных значений сравниваемых величин умноженное на эпсилон? Код вроде попроще будет.

[Ответить](#)



1. Sasha:

[9 января 2020 в 20:13](#)

Можно и так наверно, но мне кажется тут берется большее число, потому что всегда надо рассматривать худший случай

[Ответить](#)



2. Эдуард:

[15 ноября 2019 в 14:14](#)

Если при сравнении чисел указать тип `float` вместо `double`, то результатом будет `true`, даже при обычном сравнении. Это специфика компилятора или есть еще что-то?

[Ответить](#)



1. Александр:

[3 января 2020 в 21:06](#)

Я тоже заметил что float точный, думаю нужно просто запомнить что double и long double имеют такие костыли.

[Ответить](#)



2. *Борис:*

[28 апреля 2020 в 00:23](#)

Почему так уверены? У float будет всё то же самое. Принцип хранения таких чисел ведь одинаковый, что флоат что дабл. А в данном случае у вас просто удачное совпадение. Попробуйте с другими числами и найдёте "неудачные".

[Ответить](#)



3. *Артём:*

[19 октября 2019 в 18:34](#)

Возможно, вы удивитесь, но результат:

1  
0

Второй вызов не сработал так, как ожидалось. Математика просто ломается, когда дело доходит до нулей.

Почему?

[Ответить](#)



1. *Sasha:*

[9 января 2020 в 20:15](#)

Потому что почти 1 (допустим 0.9) —  $1 = -0.1$ . Да это действительно меньше нуля и функция по логике должна возвращать true, но если посмотреть внимательнее можно заметить, что там берется модуль. То есть:  $\text{fabs}(-0.1) = 0.1$ , а это уже больше нуля

[Ответить](#)



4. *Алексей:*

[5 июля 2019 в 16:42](#)

Тяжеловата тема, но интересно.

Наибольшая сложность — не знаешь сразу куда применять.

[Ответить](#)



5. *Сергей:*

[16 июня 2019 в 18:16](#)

Тема интересная, но не сразу дается. Код понял "примерно" т.е. поверхностно, чует сердце, буду к нему еще возвращаться. Принцип понятен сразу: как в тестере крутилка: 2 вольта, 20 вольт, 200

вольт и т.д. Воспоминание о аналоговых входах МК меня немного огорчило: там как раз и надо сравнивать небольшие напряжения. Например АКБ -зарядился или нет, сел или еще пойдет... теперь понимаю, почему так часто врут индикаторы заряда батарей. Спасибо, очередной интересный урок!

[Ответить](#)



1. *Юрий:*

[18 июня 2019 в 13:24](#)

Пожалуйста 😊 Главное — не за цикливайтесь, если что — вернётесь позже к этому уроку.

[Ответить](#)



6. *Вячеслав:*

[31 января 2019 в 21:04](#)

интересно для написания торгового робота на криптобирже нужно применять функцию `approximatelyEqualAbsRel()` или нет?

[Ответить](#)

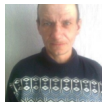


1. *Юрий:*

[1 февраля 2019 в 01:48](#)

Вы пишете ботов на C++ для криптобирж?

[Ответить](#)



1. *Вячеслав:*

[1 февраля 2019 в 15:28](#)

хочу попробовать

[Ответить](#)



7. *Дед Максим:*

[6 января 2019 в 16:01](#)

Первый урок, который я вообще не понял :). Видимо, из-за того, что не выспался. Код вообще не понятен. Пытаюсь — не выходит(

[Ответить](#)



1. *Константин:*

[25 марта 2019 в 15:36](#)

Алло, Дед Максим! Ты когда пишешь рукой на листочек строку текста и приближаешься к правому краю и видишь, что последнее слово (если будешь продолжать таким же почерком) не помещается в строку, что делаешь? Правильно. Прижмистей буквы друг к



другу тулишь. Это аналоговое представление значений. Цифровое же (то, которое в ЭВМ) — это когда все знаки и расстояния между ними строго одинаковы. И теперь представь себе, что точность — это ширина листа (если листок в клеточку, вообще, идеальная аналогия цифрового представления значений!) И вот тебе надо сравнить заряд электрона и заряд бозона. Что надо сделать? Правильно! Взять листочки по-шире, т.е. установить по-больше точность, иначе не влезающие цифры пропадут и вместо сравниваемых значений вообще какая-то дурь осядет. Но это ещё пол-беды! Подоплёка машинных "мансов" в том, что ЭВМ втихаря дописывает в клеточки левые цифры для заполнения пустующих после значащих цифр клеточек. Ну естественно результаты сравнения  $100 - 99.99$  и  $10 - 9.99$  с такими мансами будут не корректными! Да, дык о чём это я? А, вот пример: Требуется сравнить две трёхлитровых банки с жидкостью (молоко, самогон — по вкусу:-). Задаёмся граничным условием — если разница залитых объёмов не превышает одну пипетку (эпсилон) принимаем объёмы как равные. Пипетка — это абсолютный эпсилон, а объём пипетки/объём банки — это относительный эпсилон. А если объёмы сопоставимы с пипеткой (близки нулю)? Тогда Гулливер ловит лилипута, аннексирует у него пипетку (`absEpsilon`) и если разница меньше этого `absEpsilon`, то значения объёмов за "ноль" сойдут — не похмелишься (не наешься)!

[Ответить](#)



8. *Oleksiy:*

[15 августа 2018 в 12:46](#)

Радует то, что в реальной жизни чаще требуется сравнивать целые числа. А когда доходит до чисел с плавающей точкой, то там почти всегда не важно ">" или ">=".

[Ответить](#)



1. *Юрий:*

[21 августа 2018 в 21:29](#)

Ну это в реальной жизни 😊 Та и в реальной жизни бывают исключения.

[Ответить](#)



9. *Георгий:*

[29 июля 2018 в 14:21](#)

Кажется у меня отключился мозг после строчки: "Очень часто начинающие разработчики пытаются писать свои собственные функции определения равенства чисел."

[Ответить](#)



1. *Юрий:*

[29 июля 2018 в 14:31](#)

Почему? 😊

[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