

Ravesli [Ravesli](#)


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


Урок №56. Явное преобразование типов данных

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 14 Сен 2020 |

 62215

[13](#)

Из предыдущего урока мы уже знаем, что компилятор в определенных случаях выполняет **неявное преобразование типов данных**.

Оглавление:

1. [Вступление](#)
2. [Операторы явного преобразования типов данных](#)
3. [Конвертация C-style](#)
4. [Оператор static_cast](#)
5. [Использование операторов явного преобразования в неявном преобразовании](#)
6. [Заключение](#)
7. [Тест](#)

Вступление

Когда вы хотите изменить один тип данных на другой, более крупный (по размеру/диапазону), то неявное преобразование является хорошим вариантом.

Но многие начинающие программисты часто пытаются сделать что-то вроде следующего: `float x = 11 / 3;`. Однако, поскольку 11 и 3 являются целыми числами, никакого числового расширения не происходит. Выполняется целочисленное деление `11 / 3`, результатом которого будет значение 3, которое затем неявно преобразуется в `3.0` и присвоится переменной `x`!

В случае, когда вы используете **литералы** (такие как 11 или 3), замена одного или обоих целочисленных литералов значением **типа с плавающей точкой** (11.0 или 3.0) приведет к конвертации обоих

операндов в значения типа с плавающей точкой и выполнится деление типа с плавающей точкой.

Но что будет, если использовать переменные? Например:

```
1 int i1 = 11;  
2 int i2 = 3;  
3 float x = i1 / i2;
```

Значением переменной `x` будет 3. Как сообщить компилятору, что мы хотим использовать деление типа с плавающей точкой вместо целочисленного деления? Правильно! Использовать один из операторов явного преобразования типов данных, чтобы указать компилятору выполнить явное преобразование.

Операторы явного преобразования типов данных

В языке C++ есть 5 видов **операций явного преобразования типов**:

- ➔ конвертация C-style;
- ➔ применение оператора `static_cast`;
- ➔ применение оператора `const_cast`;
- ➔ применение оператора `dynamic_cast`;
- ➔ применение оператора `reinterpret_cast`.

На этом уроке мы рассмотрим конвертацию C-style и оператор `static_cast`. Оператор `dynamic_cast` мы будем рассматривать, когда дойдем до указателей и наследования. Применения операторов `const_cast` и `reinterpret_cast` следует избегать, так как они полезны только в редких случаях и могут создать немало проблем, если их использовать неправильно.

Правило: Избегайте использования `const_cast` и `reinterpret_cast`, если у вас нет на это веских причин.

Конвертация C-style

В программировании на языке Си явное преобразование типов данных выполняется с помощью оператора `()`. Внутри круглых скобок мы пишем тип, в который нужно конвертировать. Этот способ конвертации типов называется **конвертацией C-style**. Например:

```
1 int i1 = 11;  
2 int i2 = 3;  
3 float x = (float)i1 / i2;
```

В программе, приведенной выше, мы используем круглые скобки, чтобы сообщить компилятору о необходимости преобразования переменной `i1` (типа `int`) в тип `float`. Поскольку переменная `i1` станет типа `float`, то `i2` также затем автоматически преобразуется в тип `float`, и выполнится деление типа с плавающей точкой!

Язык C++ также позволяет использовать этот оператор следующим образом:

```
1 int i1 = 11;
2 int i2 = 3;
3 float x = float(i1) / i2;
```

Конвертация C-style не проверяется компилятором во время компиляции, поэтому она может быть неправильно использована, например, при конвертации типов `const` или изменении типов данных, без учета их диапазонов (что приведет к [переполнению](#)).

Следовательно, конвертацию C-style лучше не использовать.

Правило: Не используйте конвертацию C-style.

Оператор `static_cast`

В языке C++ есть еще один оператор явного преобразования типов данных — **оператор `static_cast`**. Ранее, на уроке о [символьном типе данных `char`](#), мы уже использовали оператор `static_cast` для конвертации переменной типа `char` в тип `int`, выводя вместо символа целое число:

```
1 char c = 97;
2 std::cout << static_cast<int>(c) << std::endl; // в результате выведется 97, а не 'a'
```

Оператор `static_cast` лучше всего использовать для конвертации одного фундаментального типа данных в другой:

```
1 int i1 = 11;
2 int i2 = 3;
3 float x = static_cast<float>(i1) / i2;
```

Основным преимуществом оператора `static_cast` является проверка его выполнения компилятором во время компиляции, что усложняет возможность возникновения непреднамеренных проблем.

Использование операторов явного преобразования в неявном преобразовании

Если вы будете выполнять небезопасные неявные преобразования типов данных, то компилятор будет жаловаться. Например:

```
1 int i = 49;
2 char ch = i; // неявное преобразование
```

Конвертация переменной типа `int` (4 байта) в тип `char` (1 байт) потенциально опасна — компилятор выдаст предупреждение. Чтобы сообщить ему, что вы намеренно делаете что-то, что потенциально опасно (но хотите сделать это в любом случае), используйте оператор `static_cast`:

```
1 int i = 49;
```

```
2 | char ch = static_cast<char>(i);
```

В следующем случае компилятор будет жаловаться, что конвертация из типа `double` в тип `int` может привести к потере данных:

```
1 | int i = 90;  
2 | i = i / 3.6;
```

Чтобы сообщить компилятору, что мы сознательно хотим сделать это:

```
1 | int i = 90;  
2 | i = static_cast<int>(i / 3.6);
```

Заключение

Преобразования типов данных следует избегать, если это вообще возможно, поскольку всякий раз, когда выполняется подобное изменение, есть вероятность возникновения непредвиденных проблем. Но очень часто случаются ситуации, когда этого не избежать. Поэтому в таких случаях лучше использовать оператор `static_cast` вместо конвертации C-style.

Тест

В чём разница между явным и неявным преобразованием типов данных?

Ответ

Неявное преобразование происходит, когда компилятор ожидает значение одного типа, но получает значение другого типа.

Явное преобразование происходит, когда программист использует оператор явного преобразования для конвертации значения из одного типа данных в другой.

Оценить статью:



(289 оценок, среднее: 4,91 из 5)



[← Урок №55. Неявное преобразование типов данных](#)

[Урок №57. Введение в std::string →](#)



Комментариев: 13



1. *Тимур:*
[5 августа 2020 в 13:48](#)

Вот так нужно использовать конвертацию C-style

...

Правило: Не используйте конвертацию C-style.

))))))

[Ответить](#)



1. *Юрий:*
[11 августа 2020 в 20:43](#)

Это точно 😊

[Ответить](#)



2. *Jonas:*
[17 марта 2020 в 19:23](#)

В VS 2019 static_cast == C-style. Компілятором не перевіряється я пробував.

[Ответить](#)



3. *Эдуард:*
[28 ноября 2019 в 00:49](#)

Решил просто попробовать, а оно работает. Код ниже выведет 8-битное число в двоичной системе. Не совсем понимаю, bitset о котором говорилось в уроке 46 — это тоже тип данных? Если же нет, то было бы интересно узнать как и почему это работает и что еще можно использовать с оператором static_cast.

```
1 #include <iostream>
2 #include <bitset>
3
4 int main()
5 {
6     int x{ 127 };
7
8     std::cout << static_cast<std::bitset<8>>(x) << std::endl;
9
10    return 0;
11 }
```

[Ответить](#)



1. Константин:

[26 мая 2020 в 21:34](#)

Эдуард, это же аутентичная (для ПК) форма хранения данных!

[Ответить](#)



4. Алексей:

[11 июля 2019 в 13:15](#)

Странно, но это

```
1 int i = 90;  
2 i = i / 3.6;  
3 std::cout << i << "\n";
```

прошло без всяких проблем.

[Ответить](#)



1. Андрей:

[16 июля 2019 в 14:35](#)

Смотря что Вы подразумеваете под проблемой)

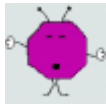
Если под проблемой Вы подразумеваете ошибку компиляции или рантайма, то да, код корректен и должен отработать, так что не удивительно, что оно "прошло без всяких проблем", однако, в действительности, проблема есть. Вам повезло со значениями и 90 делится на 3.6 без остатка, потому что имеем 25, но подели Вы 90 на 3.7 (24.32...) или 3.5 (25.71...), или ещё на какое число, данный код выдаст Вам, для 3.7 (24), а для 3.5 (25), хотя остаток есть.

Во второй строке Вы неявно приводите `i` к типу `double` при делении, за счёт дробного знаменателя, получаете вещественный результат (по сути, временный `rvalue` объект, с типом `double`), который, затем, пытаетесь присвоить переменной, тип которой как был `int`, так и остался, а, значит, будет произведено приведение вещественного результата к типу `int`.

Если Вы так и хотели — работать с целочисленным значением, то всё хорошо, в противном же случае стоит сменить тип `i`, либо, если по какой то причине этого делать не хочется, создать буфер, который будет хранить вещественный результат.

Также, Вашу вторую строчку можно сократить до `i /= 3.6;`

[Ответить](#)



5. Александр:

[2 февраля 2019 в 13:44](#)

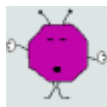
пользоваться фокусом с числовыми литералами (11.0 / 3.0 или a / 2.0) также крайне не желательно. При некоторых настройках оптимизации такое деление все равно будет произведено в целых числах

[Ответить](#)

1. Константин:

[16 февраля 2019 в 15:36](#)

Оплячки-опляпапашныя! Александр, а ну, пожалуйста, с этого момента по-подробнее (и что следует предпринять, чтобы не нарваться на "такое деление все равно будет произведено в целых числах"?)

[Ответить](#)

1. Александр:

[19 февраля 2019 в 13:28](#)

В статье же все есть :)))

Часто пользуются фокусом для вещественного деления:

```
1 int a, b;  
2 ...  
3 double res = 1.0 * a / b  
4 double av = (a + b) / 2.0
```

и тому подобным. Теоретически такое деление должно производиться в дробных числах. Но при некоторых настройках компилятора (какая-то из O+... не помню точно) этого не происходит. Если писать в аналогичных ситуациях:

```
1 double res = (double)a / b;
```

или

```
1 static_cast<double>(a) / b;
```

или что угодно другое из статьи, кроме вписывания целых числовых литералов в виде вещественных 😊

[Ответить](#)

1. Константин:

[12 марта 2019 в 05:57](#)

Благодарствую за ответ!

6. *somebox:*[17 ноября 2018 в 21:48](#)

Вопрос: а все-таки почему здесь не срабатывает преобразование во float?

```
1 int i1 = 11;  
2 int i2 = 3;  
3 float x = i1 / i2;
```

[Ответить](#)1. *Владимир:*[29 ноября 2018 в 10:28](#)

Из-за приоритетов операций (Урок 38). Сначала выполнится деление, а уже потом присваивание с неявным преобразованием.

[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 «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

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