

Ravesli [Ravesli](#)


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


Урок №55. Неявное преобразование типов данных

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 14 Сен 2020 |

 36424

[1](#)  [5](#)

Из предыдущих уроков мы уже знаем, что значение переменной хранится в виде последовательности бит, а тип переменной указывает компилятору, как интерпретировать эти биты в соответствующие значения.

Оглавление:

1. [Преобразование типов](#)
2. [Неявное преобразование типов](#)
3. [Числовое расширение](#)
4. [Числовые конверсии](#)
5. [Обработка арифметических выражений](#)

Преобразование типов

Разные типы данных могут представлять одно значение по-разному, например, значение 4 типа `int` и значение 4.0 типа `float` хранятся как совершенно разные двоичные шаблоны.

И как вы думаете, что произойдет, если сделать следующее:

```
1 | float f = 4; // инициализация переменной типа с плавающей точкой целым числом 4
```

Здесь компилятор не сможет просто скопировать биты из значения 4 типа `int` и переместить их в переменную `f` типа `float`. Вместо этого ему нужно будет преобразовать целое число 4 в число [типа с плавающей точкой](#), которое затем можно будет присвоить переменной `f`.

Процесс конвертации значений из одного типа данных в другой называется **преобразованием типов**. Преобразование типов может выполняться в следующих случаях:

Случай №1: Присваивание или инициализация переменной значением другого типа данных:

```
1 double k(4); // инициализация переменной типа double целым числом 4
2 k = 7; // присваиваем переменной типа double целое число 7
```

Случай №2: Передача значения в функцию, где тип параметра — другой:

```
1 void doSomething(long l)
2 {
3 }
4
5 doSomething(4); // передача числа 4 (тип int) в функцию с параметром типа long
```

Случай №3: Возврат из функции, где тип возвращаемого значения — другой:

```
1 float doSomething()
2 {
3     return 4.0; // передача значения 4.0 (тип double) из функции, которая возвращает
4 }
```

Случай №4: Использование бинарного оператора с операндами разных типов:

```
1 double division = 5.0 / 4; // операция деления со значениями типов double и int
```

Во всех этих случаях (и во многих других) C++ будет использовать преобразование типов.

Есть 2 основных способа преобразования типов:

- ➔ **Неявное преобразование типов**, когда компилятор автоматически конвертирует один фундаментальный тип данных в другой.
- ➔ **Явное преобразование типов**, когда разработчик использует один из операторов явного преобразования для выполнения конвертации объекта из одного типа данных в другой.

Неявное преобразование типов

Неявное преобразование типов (или «*автоматическое преобразование типов*») выполняется всякий раз, когда требуется один фундаментальный тип данных, но предоставляется другой, и пользователь не указывает компилятору, как выполнить конвертацию (не использует явное преобразование типов через операторы явного преобразования).

Есть 2 основных способа неявного преобразования типов:

- ➔ числовое расширение;
- ➔ числовая конверсия.

Числовое расширение

Когда значение из одного типа данных конвертируется в другой тип данных побольше (по размеру и по диапазону значений), то это называется **числовым расширением**. Например, тип `int` может быть расширен в тип `long`, а тип `float` может быть расширен в тип `double`:

```
1 long l(65); // расширяем значение типа int (65) в тип long
2 double d(0.11f); // расширяем значение типа float (0.11) в тип double
```

В языке C++ есть два варианта расширений:

→ **Интегральное расширение** (или «*целочисленное расширение*»). Включает в себя преобразование целочисленных типов, меньших, чем int (bool, char, unsigned char, signed char, unsigned short, signed short) в int (если это возможно) или unsigned int.

→ **Расширение типа с плавающей точкой**. Конвертация из типа float в тип double.

Интегральное расширение и расширение типа с плавающей точкой используются для преобразования «меньших по размеру» типов данных в типы int/unsigned int или double (они наиболее эффективны для выполнения разных операций).

Важно: Числовые расширения всегда безопасны и не приводят к потере данных.

Числовые конверсии

Когда мы конвертируем значение из более крупного типа данных в аналогичный, но более мелкий тип данных, или конвертация происходит между разными типами данных, то это называется **числовой конверсией**. Например:

```
1 double d = 4; // конвертируем 4 (тип int) в double
2 short s = 3; // конвертируем 3 (тип int) в short
```

В отличие от расширений, которые всегда безопасны, конверсии могут (но не всегда) привести к потере данных. Поэтому в любой программе, где выполняется неявная конверсия, компилятор будет выдавать предупреждение.

Есть много правил по выполнению числовой конверсии, но мы рассмотрим только основные.

Во всех случаях, когда происходит конвертация значения из одного типа данных в другой, который не имеет достаточного диапазона для хранения конвертируемого значения, результаты будут неожиданные. Поэтому делать так не рекомендуется. Например, рассмотрим следующую программу:

```
1 #include <iostream>
2
3 int main()
4 {
5     int i = 30000;
6     char c = i;
7
8     std::cout << static_cast<int>(c);
9
10    return 0;
11 }
```

В этом примере мы присвоили огромное целочисленное значение типа int переменной типа char (диапазон которого составляет от -128 до 127). Это приведет к переполнению и следующему результату:

Однако, если число подходит по диапазону, конвертация пройдет успешно. Например:

```
1 #include <iostream>
2
3 int main()
4 {
5     int i = 3;
6     short s = i; // конвертируем значение типа int в тип short
7     std::cout << s << std::endl;
8
9     double d = 0.1234;
10    float f = d; // конвертируем значение типа double в тип float
11    std::cout << f << std::endl;
12
13    return 0;
14 }
```

Здесь мы получим ожидаемый результат:

```
3
0.1234
```

В случаях со значениями типа с плавающей точкой могут произойти округления из-за худшей точности в меньших типах. Например:

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     float f = 0.123456789; // значение типа double - 0.123456789 имеет 9 значащих цифр
7     std::cout << std::setprecision(9) << f; // std::setprecision определен в заголовке <iomanip>
8
9     return 0;
10 }
```

В этом случае мы наблюдаем потерю в точности, так как точность типа float меньше, чем типа double:

```
0.123456791
```

Конвертация из типа int в тип float успешна до тех пор, пока значения подходят по диапазону. Например:

```
1 #include <iostream>
2
3 int main()
4 {
5     int i = 10;
6     float f = i;
7     std::cout << f;
8 }
```

```
9 |     return 0;  
10| }
```

Результат:

10

Аналогично, конвертация из `float` в `int` успешна до тех пор, пока значения подходят по диапазону. Но следует помнить, что любая дробь отбрасывается. Например:

```
1 | #include <iostream>  
2 |  
3 | int main()  
4 | {  
5 |     int i = 4.6;  
6 |     std::cout << i;  
7 |  
8 |     return 0;  
9 | }
```

Дробная часть значения (.6) игнорируется и выводится результат:

4

Обработка арифметических выражений

При обработке выражений компилятор разбивает каждое выражение на отдельные подвыражения. Арифметические операторы требуют, чтобы их операнды были одного типа данных. Чтобы это гарантировать, **компилятор использует следующие правила:**

- ➔ Если операндом является целое число меньше (по размеру/диапазону) типа `int`, то оно подвергается интегральному расширению в `int` или в `unsigned int`.
- ➔ Если операнды разных типов данных, то компилятор вычисляет операнд с наивысшим приоритетом и неявно конвертирует тип другого операнда в такой же тип, как у первого.

Приоритет типов операндов:

- ➔ `long double` (самый высокий);
- ➔ `double`;
- ➔ `float`;
- ➔ `unsigned long long`;
- ➔ `long long`;
- ➔ `unsigned long`;
- ➔ `long`;
- ➔ `unsigned int`;
- ➔ `int` (самый низкий).

Мы можем использовать **оператор typeid** (который находится в [заголовочном файле](#) `typeinfo`), чтобы узнать решающий тип в выражении.

В следующем примере у нас есть две переменные типа `short`:

```
1 #include <iostream>
2 #include <typeinfo> // для typeid
3
4 int main()
5 {
6     short x(3);
7     short y(6);
8     std::cout << typeid(x + y).name() << " " << x + y << std::endl; // вычисляем результат
9
10    return 0;
11 }
```

Поскольку значениями переменных типа `short` являются целые числа и тип `short` меньше (по размеру/диапазону) типа `int`, то он подвергается интегральному расширению в тип `int`. Результатом сложения двух `int`-ов будет тип `int`:

`int 9`

Рассмотрим другой случай:

```
1 #include <iostream>
2 #include <typeinfo> // для typeid()
3
4 int main()
5 {
6     double a(3.0);
7     short b(2);
8     std::cout << typeid(a + b).name() << " " << a + b << std::endl; // вычисляем результат
9
10    return 0;
11 }
```

Здесь `short` подвергается интегральному расширению в `int`. Однако `int` и `double` по-прежнему не совпадают. Поскольку `double` находится выше в иерархии типов, то целое число 2 преобразовывается в `2.0` (тип `double`), и сложение двух чисел типа `double` дадут число типа `double`:

`double 5`

С этой иерархией иногда могут возникать интересные ситуации, например:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << 5u - 10; // 5u означает значение 5 типа unsigned int
6
7     return 0;
8 }
```

Ожидается, что результатом выражения `5u - 10` будет `-5`, поскольку `5 - 10 = -5`, но результат:

4294967291

Здесь значение `signed int (10)` подвергается расширению в `unsigned int` (которое имеет более высокий приоритет), и выражение вычисляется как `unsigned int`. А поскольку `unsigned` — это только положительные числа, то происходит переполнение, и мы имеем то, что имеем.

Это одна из тех многих веских причин избегать использования типа `unsigned int` вообще.

Оценить статью:

★★★★★ (253 оценок, среднее: 4,92 из 5)

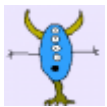


[← Урок №54. using-стейтменты](#)



[Урок №56. Явное преобразование типов данных](#)

Комментариев: 5



1. *zashiki:*

[15 августа 2019 в 15:27](#)

"конвертация — `double d = 4; // конвертируем 4 (тип int) в double`"

А почему здесь называется конвертацией (т.е. от большего к меньшему), а не расширением?

ведь `double` больше `int`

[Ответить](#)

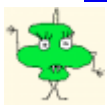


1. *Кекс:*

[21 августа 2019 в 13:41](#)

Вы путаете понятия "конвертация" и "конверсия"

[Ответить](#)



2. *Алексей:*

[10 июля 2019 в 15:19](#)

Важно: Числовые расширения всегда безопасны и не приводят к потере данных.

Это надо было бы прочитать создателям Civilization, об этом уже говорили ранее.

[Ответить](#)3. *Никита:*[29 мая 2019 в 10:44](#)

Так а зачем тогда существует тип short int , если при любой арифметической операции, он всеравно превратится в int?

[Ответить](#)1. *Евгений:*[9 июля 2019 в 21:46](#)

short int занимает в памяти в два раза меньше места.

[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