

Явное и неявное преобразование типов данных C++

В C++ различают явное и неявное преобразование типов данных.

Неявное преобразование типов данных выполняет компилятор C++.

Явное преобразование данных выполняет сам программист.

Результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех типов данных, которые участвуют в вычислении.

Для наглядного примера представлю таблицу с преобразованиями типов данных. В таблице рассмотрим операцию деления. В качестве целочисленного типа данных возьмем **int**, ну и вещественный тип данных у нас будет **float**.

Таблица 1 — Явное и неявное преобразование типов данных в C++

х	у	Результат деления	Пример
делимое	делитель	частное	$x = 15 \ y = 2$
int	int	int	$15/2=7$
int	float	float	$15/2=7.5$
float	int	float	$15/2=7.5$

- При неявном преобразовании меняя переменные различных типов данных местами, **результат остается тот же** (в нашем случае это делимое и делитель).
- Что же касается **явного преобразования**, то оно необходимо для того чтобы выполнять некоторые манипуляции, тем самым **меняя результат вычисления.**

Еще один способ явного преобразования типов данных:

1 `float(15) / 2` // результат равен 7.5, число 15 преобразуется в вещественный тип данных `float`.

2 `double(15) / 2` // результат равен 7.5 – тоже самое!!!

В C++ также предусмотрена **унарная операция** приведения типа:

`static_cast< /*тип данных*/>(/*переменная или число*/)`

Пример: `static_cast<float>(15)/2` результат равен 7.5

Пример с переменной:

`int ret=15;`

`static_cast<float>(ret)/2 //результат равен 7.5`

В случае с переменной надо понимать, что в строке 2 переменная **ret** не преобразуется в тип данных **float**, а всего лишь создается временная копия переменной **ret** с типом данных **float**.

Рассмотрим на практике все способы явного и неявного преобразования типов данных.

```
1 // pryobrazovanie.cpp: определяет точку входа для консольного приложения.
2
3 #include "stdafx.h"
4 #include <iostream>
5 #include <iomanip>
6 using namespace std;
7
8 int _tmain(int argc, _TCHAR* argv[])
9 {
10     int int_value15 = 15, int_value2 = 2; // объявляем две переменные типа int
11     float float_value15 = 15, float_value2 = 2; // объявляем две переменные типа float
12     cout << fixed << setprecision(2) // определяем, при выводе чисел с плавающей точкой
13         << "15 / 2 = " << int_value15 / int_value2 << endl // неявное преобразование
14         << "15 / 2 = " << int_value15 / float_value2 << endl // неявное преобразование
15         << "15 / 2 = " << float_value15 / int_value2 << endl // неявное преобразование
16         << "15 / 2 = " << float_value15 / float_value2 << endl; // неявное преобразование
17     cout << "15.0 / 2 = " << 15.0 / 2 << endl // явное преобразование типа данных
18         << "15 / 2.0 = " << 15 / 2.0 << endl; // явное преобразование типа данных
19     cout << "float(int_value15) / int_value2 = " << float(int_value15) / int_value2
20         << "15 / double(2) = " << 15 / double(2) << endl;
21     cout << "static_cast<float>(15) / 2 = " << static_cast<float>(15) / 2 << endl //
22         << "static_cast<char>(15) = " << static_cast<char>(15) << endl // можно преобразовать
23         << "static_cast<char>(20) = " << static_cast<char>(20) << endl; // в скобках
24     system("pause");
25     return 0;
26 }
```

В **строке 5** подключена библиотека манипуляций ввода/вывода `<iomanip>`, эта библиотека нужна для использования различных манипуляторов, в нашем случае — `fixed setprecision()`. В **строке 10** специально созданы две переменные типа `int`, аналогично создал две переменный типа `float` в **строке 11**, эти переменные нужны будут для преобразования их значений в другие типы данных. В **строке 12** после оператора `cout` и операции сдвига в поток вывода `<<` стоят два манипулятора `fixed` и `setprecision()`. Манипулятор `fixed` — это не параметризованный манипулятор, так как никаких параметров не принимает, пишется без круглых скобок. Данный манипулятор применяется в паре с параметризованным манипулятором `setprecision()` и выполняет фиксированное отображение разрядов после запятой. А манипулятор `setprecision()` отображает количество знаков после запятой, причём то, которое указано в скобках. В **строках 13, 14, 15, 16** показаны примеры неявного преобразования типов данных, эти примеры взяты из **таблицы 1**. В **строках 17, 18** показан один из способов явного преобразования данных. Суть такого способа заключается в том, что нужно дописать запятую и ноль к целому числу.

В строках 19, 20 явное преобразование выполняется посредством использования приводимых типов как функций, внутри скобочек которых, нужно указать значение или переменную, которую необходимо преобразовать. В строках 21, 22, 23 выполняется явное преобразование типов данных с помощью унарной операции преобразования данных. В круглых скобочках указывается, переменная или значение, которое нужно преобразовать, а в обрамлении знаков < > тип данных, к которому нужно преобразовать.

В строках 22, 23 выполняется унарная операция преобразования данных, причём преобразуются числа 15 и 20 к [типу данных](#) char. Этот тип данных пока вам не известен, но запомните, что char — тип данных для хранения [СИМВОЛОВ](#). Так вот, из **рисунка 1** видно, что в конце появились символы. Эти символы получились путём преобразования чисел в char. Числами являлись коды из [таблицы ASCII](#). Таким образом, если необходимо вывести какой-нибудь символ из таблицы ASCII, это можно сделать как показано в **строках 22, 23**, при этом подставив только нужный код.