

Ravesli [Ravesli](#)


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

## Урок №32. Фиксированный размер целочисленных типов данных

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 44744

[15](#)

На уроке о [целочисленных типах данных](#) мы говорили, что C++ гарантирует только их минимальный размер — они могут занимать и больше, в зависимости от компилятора и/или архитектуры компьютера.

Оглавление:

1. [Почему размер целочисленных типов не является фиксированным?](#)
2. [Разве это не глупо?](#)
3. [Целочисленные типы фиксированного размера](#)
4. [Предупреждение насчёт std::int8\\_t и std::uint8\\_t](#)
5. [Недостатки целочисленных типов фиксированного размера](#)
6. [Спор насчет unsigned](#)

### Почему размер целочисленных типов не является фиксированным?

Если говорить в общем, то всё еще началось с языка Си, когда производительность имела первостепенное значение. В языке Си намеренно оставили размер целочисленных типов нефиксированным для того, чтобы компилятор мог самостоятельно подобрать наиболее подходящий размер для определенного типа данных в зависимости от компьютерной архитектуры.

### Разве это не глупо?

Возможно. Программистам не всегда удобно иметь дело с переменными, размер которых варьируется в зависимости от компьютерной архитектуры.

## Целочисленные типы фиксированного размера

Чтобы решить вопрос кроссплатформенности, в язык C++ добавили набор **целочисленных типов фиксированного размера**, которые гарантированно имеют один и тот же размер на любой архитектуре:

Название	Тип	Диапазон значений
int8_t	1 байт signed	от -128 до 127
uint8_t	1 байт unsigned	от 0 до 255
int16_t	2 байта signed	от -32 768 до 32 767
uint16_t	2 байта unsigned	от 0 до 65 535
int32_t	4 байта signed	от -2 147 483 648 до 2 147 483 647
uint32_t	4 байта unsigned	от 0 до 4 294 967 295
int64_t	8 байт signed	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
uint64_t	8 байт unsigned	от 0 до 18 446 744 073 709 551 615

Начиная с C++11 доступ к этим типам осуществляется через подключение [заголовочного файла cstdint](#) (находятся эти типы данных в [пространстве имен std](#)). Рассмотрим пример на практике:

```
1 #include <iostream>
2 #include <cstdint>
3
4 int main()
5 {
6     std::int16_t i(5); // прямая инициализация
7     std::cout << i << std::endl;
8     return 0;
9 }
```

Поскольку целочисленные типы фиксированного размера были добавлены еще до C++11, то некоторые старые компиляторы предоставляют доступ к ним через подключение заголовочного файла stdint.h.

Если ваш компилятор не поддерживает cstdint или stdint.h, то вы можете скачать кроссплатформенный заголовочный файл [pstdint.h](#). Просто [подключите его](#) к вашему проекту, и он самостоятельно определит целочисленные типы фиксированного размера для вашей системы/архитектуры.

## Предупреждение насчет std::int8\_t и std::uint8\_t

По определенным причинам в C++ большинство компиляторов определяют и обрабатывают типы int8\_t и uint8\_t идентично типам char signed и char unsigned (соответственно), но это происходит далеко не во всех случаях. Следовательно, [std::cin и std::cout](#) могут работать не так, как вы ожидаете. Например:

```
1 #include <iostream>
```

```
2 #include <cstdint>
3
4 int main()
5 {
6     std::int8_t myint = 65;
7     std::cout << myint << std::endl;
8
9     return 0;
10 }
```

На большинстве компьютеров с различными архитектурами результат выполнения этой программы следующий:

А

Т.е. программа, приведенная выше, обрабатывает `myint` как переменную типа `char`. Однако на некоторых компьютерах результат может быть следующим:

65

Поэтому идеальным вариантом будет избегать использования `std::int8_t` и `std::uint8_t` вообще (используйте вместо них `std::int16_t` или `std::uint16_t`). Однако, если вы все же используете `std::int8_t` или `std::uint8_t`, то будьте осторожны с любой функцией, которая может интерпретировать `std::int8_t` или `std::uint8_t` как символьный тип, вместо целочисленного (например, с объектами `std::cin` и `std::cout`).

**Правило: Избегайте использования `std::int8_t` и `std::uint8_t`. Если вы используете эти типы, то будьте внимательны, так как в некоторых случаях они могут быть обработаны как тип `char`.**

## Недостатки целочисленных типов фиксированного размера

Целочисленные типы фиксированного размера могут не поддерживаться на определенных архитектурах (где они не имеют возможности быть представлены). Также эти типы могут быть менее производительными, чем фундаментальные типы данных, на определенных архитектурах.

## Спор насчет `unsigned`

Многие разработчики (и даже большие организации) считают, что программисты должны избегать использования целочисленных типов `unsigned` вообще. Главная причина — непредсказуемое поведение и результаты, которые могут возникнуть при «смешивании» целочисленных типов `signed` и `unsigned` в программе.

Рассмотрим следующий фрагмент кода:

```
1 void doSomething(unsigned int x)
2 {
3     // Выполнение некоего кода x раз
4 }
5
6 int main()
7 {
8     doSomething(-1);
```

Что произойдет в этом случае? - 1 преобразуется в другое большое число (скорее всего в 4 294 967 295). Но самое грустное в этом то, что предотвратить это мы не сможем. Язык C++ свободно конвертирует числа с типами `unsigned` в типы `signed` и наоборот без проверки диапазона допустимых значений определенного типа данных. А это, в свою очередь, может привести к переполнению.

Бьёрн Страуструп, создатель языка C++, считает, что: «Использовать тип `unsigned` (вместо `signed`) для получения еще одного бита для представления положительных целых чисел, почти никогда не является хорошей идеей».

Это не означает, что вы должны избегать использования типов `unsigned` вообще — нет. Но если вы их используете, то используйте только там, где это действительно имеет смысл, а также позаботьтесь о том, чтобы не допустить «смешивания» типов `unsigned` с типами `signed` (как в вышеприведенном примере).

Оценить статью:

★★★★★ (397 оценок, среднее: 4,97 из 5)



← [Урок №31. Целочисленные типы данных: short, int и long](#)

[Урок №33. Типы данных с плавающей точкой: float, double и long double](#) →



## Комментариев: 15



1. *Дмитрий:*

[8 апреля 2020 в 18:57](#)

Юрий, а почему в этой программе выводится "А"?  
Я что-то где-то упустил?

```
1 #include <iostream>
2 #include <cstdint>
3
4 int main()
5 {
6     std::int8_t myint = 65;
7     std::cout << myint << std::endl;
8
9     return 0;
10 }
```

[Ответить](#)



1. *Ли́за Су:*

[28 мая 2020 в 20:50](#)

Да, упустил. Прочитайте

'Предупреждение насчёт std::int8\_t и std::uint8\_t'

Думаю, поздно ответил

[Ответить](#)



2. *A-Lex:*

[22 ноября 2019 в 09:07](#)

```

1 //Программа для деления целых чисел с заданной разрядностью после запятой,
2 //с использованием ТОЛЬКО целочисленного типа данных
3
4 #include <iostream>
5 #include <math.h> //Для функции возведения в степень
6 //другого решения не нашёл, а целочисленный тип данных
7
8 using namespace std; //Чтобы не писать постоянно std::
9
10 //Подпрограмма для введения количества разрядов после запятой
11 int number_of_decimal_places()
12 {
13     int number;
14
15     cout << "Enter the number of decimal places: " << endl;
16     cin >> number;
17
18     return pow(10, number); //Возведение в степень
19 }
20
21 int main()
22 {
23     int x, y, z, a, b;
24     cin >> x;
25     cin >> y;
26
27     z = x / y; //Значение целой части, т. е. целая часть
28     a = x % y; //Земельный остаток
29
30     b = a * number_of_decimal_places() / y; //Значение после запятой с заданной разрядностью
31
32     cout << x << " / " << y << " = " << z << ", " << b << endl;
33
34     return 0;
35 }
36
37 //В разных IDE с разными компиляторами программа выдаёт разные значения
38

```

```

39 //При введении:
40 //x -      5
41 //y -      7
42 //number - 2 (количество разрядов после запятой)
43 //Code::Blocks 17.12 (GNU GCC Compiler) выдаёт --> 0,70
44 //хотя должно быть --> 0,71 (посчитал на калькуляторе)
45 //Visual Studio 2019 выдаёт правильный результат --> 0,71
46 //С помощью пошаговой отладки выяснил:
47 // "return pow(10, number);" возводит 10 во вторую степень с ошибкой --> 10^2=99
48 // "return pow(10, number);" возводит 10 в третью степень с ошибкой --> 10^3=1000
49 //Функция pow - имеет формат double, а моя подпрограмма возвращает int
//по ходу загвоздка где-то тут (преобразование форматов), но почему в другой IDE

```

### [Ответить](#)



1. **Sasha:**

[8 января 2020 в 23:54](#)

Очень странно

### [Ответить](#)



2. **Sasha:**

[9 января 2020 в 19:04](#)

Понял из-за чего могла возникнуть ошибка. Функция pow принимает в качестве аргументов числа типа double, а как мы знаем с ними могут быть ошибки округления. И компилятор code::blocks и visual studio обрабатывают это по-разному. Если написать свою функцию возведения в степень для целых чисел, все ок

### [Ответить](#)



3. **Александр:**

[28 августа 2020 в 17:37](#)

Зачем запихивать пользовательский ввод и возведение в степень с последующим возвратом, в отдельную функцию. А потом эту функцию вписывать в присвоение между математическими операторами ? Задача ведь намного проще стоит.

Все что нужно это:

```

1  int main()
2  {
3  int x, y, z, a, b;
4  int number;
5
6  cout << "enter the number to be divided: ";
7  cin >> x;    // 5
8  cout << "enter divisor ";
9  cin >> y;    // 7

```

```

10
11 z = x / y;
12 a = x % y;
13
14 cout << "enter the number of decimal places: ";
15 cin >> number;    //2
16
17 b = a * pow(10,number) / y;
18 cout << x << "/" << y << " = " << z << ", " << b << endl;
19 }
20
21
22 // Компилируется и работает в Code::Blocks 17.00+

```

[Ответить](#)



3. *AleksTs:*

[20 октября 2019 в 09:44](#)

Спасибо за вашу проделанную работу!!! Все cool !

[Ответить](#)

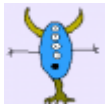


1. *Юрий:*

[20 октября 2019 в 13:12](#)

Пожалуйста))

[Ответить](#)



4. *zashiki:*

[15 июля 2019 в 15:42](#)

Подскажите!

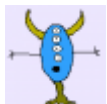
Использую VS express 2017

Если unsigned переменной, к примеру, для 16-битного short, присвоить отрицательное число, то программа изменит на соответствующее unsigned положительное, т.е. -1 конвертируется в 65535. И тут все понятно.

Но если signed переменной, к примеру, для 16-битного short, присвоить число больше предела signed, к примеру 65535, то оно не конвертируется в -1, а просто конвертируется в предел 32767, и так с любым больше предела!

Это какая то машинная особенность или просто особенность VS?

[Ответить](#)



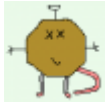
1. *zashiki:*

[15 июля 2019 в 17:52](#)

прошу прощения, проблема, оказывается, только тогда, если переменная присваивается пользователем, т.е. через `cin`

Но все равно вопрос остается.

[Ответить](#)



5. *Алексей:*

[13 декабря 2018 в 05:56](#)

Здравствуйте Юрий, у меня раньше (чуть меньше половины года) спокойно работал Русский язык. Сейчас же он выводит вот это: Яырслш Ярф ШчыршЫ... и т. д. Подскажите решение, пожалуйста.

[Ответить](#)



6. *David:*

[13 сентября 2018 в 21:56](#)

Юрий, можно вас спросить, можно ли восстановить потерянные знания по c++?

[Ответить](#)



1. *Юрий:*

[14 сентября 2018 в 18:59](#)

Можно, если приложить соответствующих усилий.

[Ответить](#)

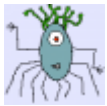


7. *Константин:*

[10 сентября 2018 в 09:47](#)

Юра, а как я пойму, что передо мной чума, в смысле собственная версия компилятора фикс цел числ типа?

[Ответить](#)



8. *Алексей:*

[13 июня 2018 в 14:11](#)

По поводу фиксированные типов, наткнулся тут на статейку:

"Процессоры с 36-битной архитектурой как правило имеют 9-битный байт, а в некоторых DSP от Texas Instruments байты состоят из 16 или 32 бит. Древние архитектуры могут иметь короткие байты из 4, 5 или 7 бит."

И для таких систем приведенные типы не будут работать, так как в них не существует `int8_t = 8` байтам.

Поэтому надежнее использовать:

`int_leastN_t (int_least8_t ... int_least64_t; uint_least8_t ... uint_least64_t)` — минимальный целочисленный тип шириной не менее N бит.



и `int_fastN_t(int_fast8_t ... int_fast64_t; uint_fast8_t ... uint_fast64_t)` — самый быстро обрабатываемый системой целочисленный тип шириной не менее N бит.

[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