

Ravesli [Ravesli](#)


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


## Урок №31. Целочисленные типы данных: short, int и long

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 62050

[1](#)  [21](#)

На этом уроке мы рассмотрим целочисленные типы данных в языке C++, их диапазоны значений, операцию деления, а также переполнение (что это такое и примеры).

Оглавление:

1. [Целочисленные типы данных](#)
2. [Определение целочисленных переменных](#)
3. [Диапазоны значений и знак целочисленных типов данных](#)
4. [Что используется по умолчанию: signed или unsigned?](#)
5. [Переполнение](#)
6. [Примеры переполнения](#)
7. [Деление целочисленных переменных](#)

## Целочисленные типы данных

**Целочисленный тип данных** — это тип, переменные которого могут содержать только целые числа (без дробной части, например: -2, -1, 0, 1, 2). В языке C++ есть 5 основных целочисленных типов, доступных для использования:

| Категория                | Тип       | Минимальный размер              |
|--------------------------|-----------|---------------------------------|
| Символьный тип данных    | char      | 1 байт                          |
| Целочисленный тип данных | short     | 2 байта                         |
|                          | int       | 2 байта (но чаще всего 4 байта) |
|                          | long      | 4 байта                         |
|                          | long long | 8 байт                          |

**Примечание:** Тип `char` — это особый случай: он является как целочисленным, так и символьным типом данных. Об этом детально мы поговорим на одном из следующих уроков.

Основным различием между целочисленными типами, перечисленными выше, является их **размер**, чем он больше, тем больше значений сможет хранить переменная этого типа.

## Определение целочисленных переменных

Определение происходит следующим образом:

```
1 char c;  
2 short int si; // допустимо  
3 short s;      // предпочтительнее  
4 int i;  
5 long int li; // допустимо  
6 long l;       // предпочтительнее  
7 long long int lli; // допустимо  
8 long long ll;  // предпочтительнее
```

В то время как полные названия `short int`, `long int` и `long long int` могут использоваться, их сокращенные версии (без `int`) более предпочтительны для использования. К тому же постоянное добавление `int` затрудняет чтение кода (легко перепутать с именем переменной).

## Диапазоны значений и знак целочисленных типов данных

Как вы уже знаете из предыдущего урока, переменная с  $n$ -ным количеством бит может хранить  $2^n$  возможных значений. Но что это за значения? Это значения, которые находятся в диапазоне. **Диапазон** — это значения от и до, которые может хранить определенный тип данных. Диапазон целочисленной переменной определяется двумя факторами: её размером (измеряется в битах) и её **знаком** (который может быть *signed* или *unsigned*).

**Целочисленный тип signed (со знаком)** означает, что переменная может содержать как положительные, так и отрицательные числа. Чтобы объявить переменную как `signed`, используйте ключевое слово `signed`:

```
1 signed char c;  
2 signed short s;  
3 signed int i;  
4 signed long l;  
5 signed long long ll;
```

По умолчанию, ключевое слово `signed` пишется перед типом данных.

1-байтовая целочисленная переменная со знаком (`signed`) имеет диапазон значений от -128 до 127, т.е. любое значение от -128 до 127 (включительно) может храниться в ней безопасно.

В некоторых случаях мы можем заранее знать, что отрицательные числа в программе использоваться не будут. Это очень часто встречается при использовании переменных для хранения количества или размера чего-либо (например, ваш рост или вес не может быть отрицательным).

**Целочисленный тип unsigned (без знака)** может содержать только положительные числа. Чтобы объявить переменную как **unsigned**, используйте ключевое слово **unsigned**:

```
1 unsigned char c;
2 unsigned short s;
3 unsigned int i;
4 unsigned long l;
5 unsigned long long ll;
```

1-байтовая целочисленная переменная без знака (unsigned) имеет диапазон значений от 0 до 255.

Обратите внимание, объявление переменной как unsigned означает, что она не сможет содержать отрицательные числа (только положительные).

Теперь, когда вы поняли разницу между signed и unsigned, давайте рассмотрим диапазоны значений разных типов данных:

| Размер/Тип        | Диапазон значений                                          |
|-------------------|------------------------------------------------------------|
| 1 байт signed     | от -128 до 127                                             |
| 1 байт unsigned   | от 0 до 255                                                |
| 2 байта signed    | от -32 768 до 32 767                                       |
| 2 байта unsigned  | от 0 до 65 535                                             |
| 4 байта signed    | от -2 147 483 648 до 2 147 483 647                         |
| 4 байта unsigned  | от 0 до 4 294 967 295                                      |
| 8 байтов signed   | от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 |
| 8 байтов unsigned | от 0 до 18 446 744 073 709 551 615                         |

*Для математиков:* Переменная signed с n-ным количеством бит имеет диапазон от  $-(2^{n-1})$  до  $2^{n-1}-1$ .  
Переменная unsigned с n-ным количеством бит имеет диапазон от 0 до  $(2^n)-1$ .

*Для нематематиков:* Используем таблицу 😊

Начинающие программисты иногда путаются между signed и unsigned переменными. Но есть простой способ запомнить их различия. Чем отличается отрицательное число от положительного? Правильно! Минусом спереди. Если минуса нет, значит число — положительное. Следовательно, целочисленный тип со знаком (signed) означает, что минус может присутствовать, т.е. числа могут быть как положительными, так и отрицательными. Целочисленный тип без знака (unsigned) означает, что минус спереди отсутствует, т.е. числа могут быть только положительными.

## Что используется по умолчанию: signed или unsigned?

Так что же произойдет, если мы объявим переменную без указания signed или unsigned?

| Категория                | Тип   | По умолчанию                                       |
|--------------------------|-------|----------------------------------------------------|
| Символьный тип данных    | char  | signed или unsigned (в большинстве случаев signed) |
| Целочисленный тип данных | short | signed                                             |
|                          | int   | signed                                             |

long      signed  
long long signed

Все целочисленные типы данных, кроме char, являются signed по умолчанию. Тип char может быть как signed, так и unsigned (но, обычно, signed).

В большинстве случаев ключевое слово signed не пишется (оно и так используется по умолчанию).

Программисты, как правило, избегают использования целочисленных типов unsigned, если в этом нет особой надобности, так как с переменными unsigned ошибок, по статистике, возникает больше, нежели с переменными signed.

**Правило: Используйте целочисленные типы signed, вместо unsigned.**

## Переполнение

Вопрос: «Что произойдет, если мы попытаемся использовать значение, которое находится вне диапазона значений определенного типа данных?». Ответ: «Переполнение».

**Переполнение** (англ. «*overflow*») случается при потере бит из-за того, что переменной не было выделено достаточно памяти для их хранения.

На [уроке №28](#) мы говорили о том, что данные хранятся в бинарном (двоичном) формате и каждый бит может иметь только 2 возможных значения (0 или 1). Вот как выглядит диапазон чисел от 0 до 15 в десятичной и двоичной системах:

| Десятичная система | Двоичная система |
|--------------------|------------------|
| 0                  | 0                |
| 1                  | 1                |
| 2                  | 10               |
| 3                  | 11               |
| 4                  | 100              |
| 5                  | 101              |
| 6                  | 110              |
| 7                  | 111              |
| 8                  | 1000             |
| 9                  | 1001             |
| 10                 | 1010             |
| 11                 | 1011             |
| 12                 | 1100             |
| 13                 | 1101             |
| 14                 | 1110             |
| 15                 | 1111             |

Как вы можете видеть, чем больше число, тем больше ему требуется бит. Поскольку наши переменные имеют фиксированный размер, то на них накладываются ограничения на количество данных, которые они могут хранить.

## Примеры переполнения

Рассмотрим переменную `unsigned`, которая состоит из 4-х бит. Любое из двоичных чисел, перечисленных в таблице выше, поместится внутри этой переменной.

«Но что произойдет, если мы попытаемся присвоить значение, которое занимает больше 4-х бит?». Правильно! Переполнение. Наша переменная будет хранить только 4 наименее значимых (те, что справа) бита, все остальные — потеряются.

Например, если мы попытаемся поместить число 21 в нашу 4-битную переменную:

| Десятичная система | Двоичная система |
|--------------------|------------------|
| 21                 | 10101            |

Число 21 занимает 5 бит (10101). 4 бита справа (0101) поместятся в переменную, а крайний левый бит (1) просто потеряется. Т.е. наша переменная будет содержать 0101, что равно 101 (ноль спереди не считается), а это уже число 5, а не 21.

**Примечание:** О конвертации чисел из двоичной системы в десятичную и наоборот будет отдельный урок, где мы всё детально рассмотрим и обсудим.

Теперь рассмотрим пример в коде (тип `short` занимает 16 бит):

```
1 #include <iostream>
2
3 int main()
4 {
5     unsigned short x = 65535; // наибольшее значение, которое может хранить 16-битная
6     std::cout << "x was: " << x << std::endl;
7     x = x + 1; // 65536 - это число больше максимально допустимого числа из диапазона
8     std::cout << "x is now: " << x << std::endl;
9     return 0;
10 }
```

Результат выполнения программы:

x was: 65535

x is now: 0

Что случилось? Произошло переполнение, так как мы попытались присвоить переменной `x` значение больше, чем она способна в себе хранить.

**Для тех, кто хочет знать больше:** Число 65 535 в двоичной системе счисления представлено как 1111 1111 1111 1111. 65 535 — это наибольшее число, которое может хранить 2-байтовая (16 бит) целочисленная переменная без знака, так как это число использует все 16 бит. Когда мы добавляем 1, то получаем число 65 536. Число 65 536 представлено в двоичной системе как 1 0000 0000 0000 0000, и занимает 17 бит! Следовательно, самый главный бит (которым является 1) теряется, а все 16 бит справа — остаются. Комбинация 0000 0000 0000 0000 соответствует десятичному 0, что и является нашим результатом.

Аналогичным образом, мы получим переполнение, используя число меньше минимального из диапазона допустимых значений:

```
1 #include <iostream>
2
3 int main()
4 {
5     unsigned short x = 0; // наименьшее значение, которое 2-байтовая unsigned переменная
6     std::cout << "x was: " << x << std::endl;
7     x = x - 1; // переполнение!
8     std::cout << "x is now: " << x << std::endl;
9     return 0;
10 }
```

Результат выполнения программы:

```
x was: 0
x is now: 65535
```

Переполнение приводит к потере информации, а это никогда не приветствуется. Если есть хоть малейшее подозрение или предположение, что значением переменной может быть число, которое находится вне диапазона допустимых значений используемого типа данных — используйте тип данных побольше!

**Правило: Никогда не допускайте возникновения переполнения в ваших программах!**

## Деление целочисленных переменных

В языке C++ при делении двух целых чисел, где результатом является другое целое число, всё довольно предсказуемо:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << 20 / 4 << std::endl;
6     return 0;
7 }
```

Результат:

```
5
```

Но что произойдет, если в результате деления двух целых чисел мы получим дробное число? Например:

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

Результат:


1

В языке C++ при делении целых чисел результатом всегда будет другое целое число. А такие числа не могут иметь дробь (она просто отбрасывается, **не округляется!**).

Рассмотрим детально вышеприведенный пример:  $8 / 5 = 1.6$ . Но как мы уже знаем, при делении целых чисел результатом является другое целое число. Таким образом, дробная часть (0.6) значения отбрасывается и остается 1.

**Правило: Будьте осторожны при делении целых чисел, так как любая дробная часть всегда отбрасывается.**

Оценить статью:

 (415 оценок, среднее: 4,92 из 5)



[← Урок №30. Размер типов данных](#)

[Урок №32. Фиксированный размер целочисленных типов данных](#) →



## Комментариев: 21



1. *VEX:*

[23 июня 2020 в 14:23](#)

хехе, а попробуйте к знаковому 2 байтному целоисчисленной переменной со значением 32767 прибавить 1, или так-же наоборот от -32768 отнять 1

[Ответить](#)



2. *Sagynysh:*

[2 мая 2020 в 21:45](#)

Для того, чтоб числа имели дробь при делении целых чисел можно приписать ноль после точкой. Например :  $8.0/5.0 = 1.6$

[Ответить](#)



1. *Олег:*

[21 мая 2020 в 19:41](#)

Только это уже совсем другая история)

[Ответить](#)2. *unfeat:*[2 октября 2020 в 06:50](#)

Достаточно поставить точку одному из выражений. Например: 8. / 5 или 8 / 5.

[Ответить](#)1. *unfeat:*[2 октября 2020 в 06:51](#)

Остальное компилятор сам подставит)

[Ответить](#)3. *Борис:*[25 апреля 2020 в 19:30](#)

Вообще, с "железным" правилом "Никогда не допускайте возникновения переполнения в ваших программах!" — сильно погорячились. Потому что очень часто переполнение как раз помогает создать более простой и быстрый код.

Например, нужно много раз увеличивать переменную на 1 и циклически прокручивать все значения от 0 до 255. Писать условие "если равно 255, то присвоить 0" — совсем не нужно, это произойдёт само при прибавлении 1 к 255, если используется 1-байтовая беззнаковая.

Другой очень частый пример: вычисление разности двух значений миллисекундного таймера, чтобы замерить период времени. 4-байтовая переменная с таким таймером переполняется каждые 49 суток. Если система работает непрерывно, то такое может случаться. Когда считаем разность (новое значение таймера минус старое) — возможен случай, когда новое значение уже переполнилось (снова пошло с нуля), а старое ещё нет (огромное число). Но когда вычисляется разность, тут снова произойдёт переполнение (из-за того, что получилось отрицательное значение), и эти два переполнения оказывают взаимно компенсирующее действие, как будто их не было вообще. И разность всё равно будет верной. И не надо городить никаких хитрых алгоритмов.

[Ответить](#)4. *vik:*[7 августа 2019 в 12:41](#)

В некоторых источниках встречал, что в связи со знаковостью, появляются два варианта нуля: +0 и -0. Почему-то тут этот вопрос не затронут. Ему перестали предавать значение в сообществе?

[Ответить](#)1. *Владимир:*[15 ноября 2019 в 19:20](#)



Скорее всего это какой-то очень древний подход. Никогда не слышал подобного в универе.

[Ответить](#)



2. *Борис:*

[25 апреля 2020 в 19:10](#)

Потому что это относится к числам с плавающей точкой. У них отдельный бит хранит знак. В целочисленных типах такого нигде (или почти нигде) нет.

[Ответить](#)



5. *Денис:*

[28 июня 2019 в 13:12](#)

unsigned используется для экономии памяти, это же очевидно. Если знак действительно не нужен за счет дополнительно освобожденного бита, можно увеличить диапазон значений в 2 раза, что в некоторых случаях позволит использовать более "экономные" типы данных.

[Ответить](#)



6. *Ulugbek:*

[4 ноября 2018 в 18:58](#)

так если при делении дробная часть отбрасывается ,то как создать калькулятор?Если он не будут выводить дробные числа.Или ответ стоит присвоить к переменной которая будет иметь тип float?

[Ответить](#)



1. *Дед Максим:*

[19 декабря 2018 в 19:51](#)

Ну так нужно указывать другой тип переменной(не целое число). Тогда будет дробь.

[Ответить](#)



7. *Игорь...:*

[10 июля 2018 в 02:52](#)

Забавная история, почему этот урок так важен =)

В игре Civilization есть баг с механикой агрессии и миролюбия. Суть такова, что агрессивность цивилизации измерялась по шкале от 1 до 10. Девятки и десятки были у всяких Чингисханов, Монтесум и Сталиных, а у духовного пацифиста Махатмы Ганди была единичка. И ещё были модификаторы — строй «республика» уменьшает агрессивность на 1, «демократия» — на 2. Соответственно, сразу же, как только индусы открывали Демократию, у Ганди становилась агрессивность −1.

А теперь внимание. Эта переменная была однобайтная и строго неотрицательная(unsigned), от 0 до 255. Соответственно, агрессивность Махатмы Ганди становилась равна 255 из 10. Поэтому

построив у себя демократию, Ганди двигался рассудком, клепал ядрёные бомбы и умножал всех на ноль.

[Ответить](#)



1. Юрий:

[10 июля 2018 в 07:42](#)

Действительно хороший пример 😊 С unsigned нужно быть аккуратным.

[Ответить](#)



8. dbnz:

[7 июня 2018 в 17:53](#)

Как тип char может быть отрицательным?

[Ответить](#)



1. Юрий:

[7 июня 2018 в 19:25](#)

Тип char не может быть отрицательным. Он относится к целочисленным типам данных, диапазон его значений — от 0 до 127. В статье указано, что char — особый случай и о нем есть отдельный урок.

[Ответить](#)



9. Виталий:

[23 сентября 2017 в 18:37](#)

>>»Правило: Используйте целочисленные переменные signed, вместо unsigned.»

Офигенное правило.

unsigned изобрели просто так, чтоб им не пользоваться...

[Ответить](#)



1. Old G.B.:

[2 октября 2017 в 21:20](#)

Претензии к автору предъявлять нельзя, он только лишь переводит.

[Ответить](#)



1. Юрий:

[2 октября 2017 в 21:38](#)

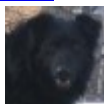
Это нормально, я ведь учусь вместе с вами. Отвечая на вопросы и претензии — лучше начинаешь понимать суть изучаемого сам 😊

[Ответить](#)

2. Юрий:

[2 октября 2017 в 21:36](#)

Unsigned полезен лишь в конкретных случаях, когда вы знаете, что его использование не приведет к проблемам и сбою в программе, когда вы можете это гарантировать. Для более широкого использования (повседневного) рекомендуется тип signed, так как он менее подвержен ошибкам и более надежный. Всё просто — можете прислушиваться, можете нет — ваше право.

[Ответить](#)

1. Cerberus:

[27 марта 2019 в 07:04](#)

На самом деле, совсем не факт. Я в последнее время стал много читать статьи от авторов одного статического анализатора (не называю имя, чтоб не было рекламы), в которых неоднократно всплывал такой простой факт: переполнение `_знаковых_` переменных — это Undefined behaviour, т.е. компилятор имеет право сделать с ним всё что угодно, если нет явной просьбы зафиксировать тот или иной вариант; переполнение `_беззнаковых_` переменных же — стандартно (модульная арифметика). Поэтому вероятность вляпаться в ошибки для знаковых переменных, скорее всего, даже выше, просто потому что UB вполне может работать "как задумано", пока не произойдёт какого-то внешнего изменения (хотя бы и обновления компилятора, в котором прилетела пачка новых оптимизаций).

[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

Электронная почта



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