

Ravesli [Ravesli](#)


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

Урок №33. Типы данных с плавающей точкой: float, double и long double

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 90710

|  42

На этом уроке мы рассмотрим типы данных с плавающей точкой в языке C++, их точность и диапазон. Выясним, что такое экспоненциальная запись и как она используется, а также рассмотрим ошибки округления и дадим определения для `nan` и `inf`.

Оглавление:

1. [Типы данных с плавающей точкой](#)
2. [Экспоненциальная запись](#)
3. [Конвертация чисел в экспоненциальную запись](#)
4. [Точность и диапазон типов с плавающей точкой](#)
5. [Ошибки округления](#)
6. [nan и inf](#)
7. [Заключение](#)
8. [Тест](#)

Типы данных с плавающей точкой

[Целочисленные типы данных](#) отлично подходят для работы с целыми числами, но есть ведь еще и дробные числа. И тут нам на помощь приходит **тип данных с плавающей точкой** (или «тип данных с плавающей запятой», англ. «*floating point*»). Переменная такого типа может хранить любые действительные дробные значения, например: 4320.0, -3.33 или 0.01226. Почему точка «плавающая»? Дело в том, точка/запятая перемещается («плавает») между цифрами, разделяя целую и дробную части значения.

Есть три типа данных с плавающей точкой: **float**, **double** и **long double**. Язык C++ определяет только их минимальный размер (как и с целочисленными типами). Типы данных с плавающей точкой всегда являются **signed** (т.е. могут хранить как положительные, так и отрицательные числа).

Категория	Тип	Минимальный размер	Типичный размер
Тип данных с плавающей точкой	float	4 байта	4 байта
	double	8 байт	8 байт
	long double	8 байт	8, 12 или 16 байт

Объявление переменных разных типов данных с плавающей точкой:

```
1 float fValue;
2 double dValue;
3 long double dValue2;
```

Если нужно использовать целое число с переменной типа с плавающей точкой, то тогда после этого числа нужно поставить разделительную точку и ноль. Это позволяет различать переменные целочисленных типов от переменных типов с плавающей запятой:

```
1 int n(5); // 5 - это целочисленный тип
2 double d(5.0); // 5.0 - это тип данных с плавающей точкой (по умолчанию double)
3 float f(5.0f); // 5.0 - это тип данных с плавающей точкой ("f" от "float")
```

Обратите внимание, литералы типа с плавающей точкой по умолчанию относятся к типу double. f в конце числа означает тип float.

Экспоненциальная запись

Экспоненциальная запись очень полезна для написания длинных чисел в краткой форме. Числа в экспоненциальной записи имеют следующий вид: **мантисса** $\times 10^{\text{экспонент}}$. Например, рассмотрим **выражение** 1.2×10^4 . Значение 1.2 — это **мантисса** (или «*значащая часть числа*»), а 4 — это **экспонент** (или «*порядок числа*»). Результатом этого выражения является значение 12000.

Обычно, в экспоненциальной записи, в целой части находится только одна цифра, все остальные пишутся после разделительной точки (в дробной части).

Рассмотрим массу Земли. В десятичной системе счисления она представлена как 597360000000000000000000 кг. Согласитесь, очень большое число (даже слишком большое, чтобы поместиться в целочисленную переменную размером 8 байт). Это число даже трудно читать (там 19 или 20 нулей?). Но используя экспоненциальную запись, массу Земли можно представить, как 5.9736×10^{24} кг (что гораздо легче воспринимается, согласитесь). Еще одним преимуществом экспоненциальной записи является сравнение двух очень больших или очень маленьких чисел — для этого достаточно просто сравнить их экспоненты.

В языке C++ буква e /E означает, что число 10 нужно возвести в степень, которая следует за этой буквой. Например, 1.2×10^4 эквивалентно 1.2e4, значение 5.9736×10^{24} еще можно записать как 5.9736e24.

Для чисел меньше единицы экспонент может быть отрицательным. Например, $5e-2$ эквивалентно 5×10^{-2} , что, в свою очередь, означает $5 / 10^2$ или 0.05. Масса электрона равна $9.1093822e-31$ кг.

На практике экспоненциальная запись может использоваться в операциях присваивания следующим образом:

```
1 double d1(5000.0);  
2 double d2(5e3); // другой способ присвоить значение 5000  
3  
4 double d3(0.05);  
5 double d4(5e-2); // другой способ присвоить значение 0.05
```

Конвертация чисел в экспоненциальную запись

Для конвертации чисел в экспоненциальную запись необходимо следовать процедуре, указанной ниже:

- Ваш экспонент начинается с нуля.
- Переместите разделительную точку (которая разделяет целую и дробную части) влево, чтобы слева от нее осталась только одна ненулевая цифра:
 - ➔ каждое перемещение точки влево увеличивает экспонент на 1;
 - ➔ каждое перемещение точки вправо уменьшает экспонент на 1.
- Откиньте все нули перед первой ненулевой цифрой в целой части.
- Откиньте все конечные нули в правой (дробной) части, *только если исходное число является целым (без разделительной точки)*.

Рассмотрим примеры:

Исходное число: 42030

Перемещаем разделительную точку на 4 цифры влево: 4.2030e4

Слева (в целой части) нет нулей: 4.2030e4

Отбрасываем конечный нуль в дробной части: 4.203e4 (4 значащие цифры)

Исходное число: 0.0078900

Перемещаем разделительную точку на 3 цифры вправо: 0007.8900e-3

Отбрасываем нули слева: 7.8900e-3

Не отбрасываем нули справа (исходное число является дробным): 7.8900e-3 (5 значащих цифр)

Исходное число: 600.410

Перемещаем разделительную точку на 2 цифры влево: 6.00410e2

Слева нет нулей: 6.00410e2

Нули справа оставляем: 6.00410e2 (6 значащих цифр)

Самое главное, что нужно запомнить — это то, что цифры в мантиссе (часть перед e) называются **значащими цифрами**. Количество значащих цифр определяет **точность** самого значения. Чем больше цифр в мантиссе, тем точнее значение.

Точность и диапазон типов с плавающей точкой

Рассмотрим дробь $1/3$. Десятичное представление этого числа — $0.3333333333333333...$ (с тройками до бесконечности). Бесконечное число требует бесконечной памяти для хранения, а у нас в запасе, как правило, 4 или 8 байт. Переменные типа с плавающей запятой могут хранить только определенное количество значащих цифр, остальные — отбрасываются. **Точность определяется количеством значащих цифр**, которые представляют число без потери данных.

Когда мы выводим переменные типа с плавающей точкой, то точность объекта `cout`, по умолчанию, составляет 6. Т.е. на экране мы увидим только 6 значащих цифр, остальные — потеряются. Например:

```
1 #include <iostream>
2
3 int main()
4 {
5     float f;
6     f = 9.87654321f;
7     std::cout << f << std::endl;
8     f = 987.654321f;
9     std::cout << f << std::endl;
10    f = 987654.321f;
11    std::cout << f << std::endl;
12    f = 9876543.21f;
13    std::cout << f << std::endl;
14    f = 0.0000987654321f;
15    std::cout << f << std::endl;
16    return 0;
17 }
```

Результат выполнения программы:

```
9.87654
987.654
987654
9.87654e+06
9.87654e-05
```

Обратите внимание, каждое из вышеприведенных значений имеет только 6 значащих цифр (цифры перед `e`, а не перед точкой).

Также, в некоторых случаях, `cout` сам может выводить числа в экспоненциальной записи. В зависимости от компилятора, экспонент может быть дополнен нулями. Например, $9.87654e+06$ — это то же самое, что и $9.87654e6$ (просто с добавленным нулем и знаком `+`). Минимальное количество цифр экспонента определяется компилятором (Visual Studio использует 2, другие компиляторы могут использовать 3).

Также мы можем переопределить точность `cout`, используя **функцию `std::setprecision()`**, которая находится в **заголовочном файле `iomanip`**:

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     std::cout << std::setprecision(16); // задаем точность в 16 цифр
```

Результат выполнения программы:

Так как мы увеличили точность до 16, то каждая переменная выводится 16-ю цифрами. Но, как вы можете видеть, исходные числа имеют больше цифр!

Точность зависит от размера типа данных (в типе float точность меньше, чем в типе double) и от присваиваемого значения:

- ➔ **точность float**: от 6 до 9 цифр (в основном 7);
- ➔ **точность double**: от 15 до 18 цифр (в основном 16);
- ➔ **точность long double**: 15, 18 или 33 цифры (в зависимости от того, сколько байт занимает тип данных на компьютере).

Этот принцип относится не только к дробным числам, но и ко всем значениям, которые имеют слишком большое количество значащих цифр. Например:

Результат:

123456792

Но ведь 123456792 больше чем 123456789, не так ли? Значение 123456789.0 имеет 10 значащих цифр, но точность float равна 7. Поэтому мы и получили другое число, произошла потеря данных!

Следовательно, нужно быть осторожными, когда вы используете переменные типа с плавающей точкой вместе с очень большими/очень маленькими числами, которые требуют большей точности, чем их текущий тип данных может предложить.

Диапазон и точность типов данных с плавающей точкой, согласно [стандарту IEEE 754](#):

Размер	Диапазон	Точность
4 байта	от $\pm 1.18 \times 10^{-38}$ до $\pm 3.4 \times 10^{38}$	6-9 значащих цифр (в основном 7)

8 байт от $\pm 2.23 \times 10^{-308}$ до $\pm 1.80 \times 10^{308}$ 15-18 значащих цифр (в основном 16)

80 бит (12 байт) от $\pm 3.36 \times 10^{-4932}$ до $\pm 1.18 \times 10^{4932}$ 18-21 значащих цифр

16 байт от $\pm 3.36 \times 10^{-4932}$ до $\pm 1.18 \times 10^{4932}$ 33-36 значащих цифр

Может показаться немного странным, что 12-байтовая переменная типа с плавающей точкой имеет тот же диапазон, что и 16-байтовая переменная. Это потому, что они имеют одинаковое количество бит, выделенных для экспонента (только в 16-байтовой переменной точность будет выше).

Правило: Используйте по умолчанию тип double вместо типа float, так как его точность выше.

Ошибки округления

Рассмотрим дробь $1/10$. В десятичной системе счисления эту дробь можно представить, как 0.1. В двоичной системе счисления эта дробь представлена в виде бесконечной последовательности — 0.00011001100110011... Именно из-за подобных разногласий в представлении чисел в разных системах счисления, у нас могут возникать проблемы с точностью. Например:

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     double d(0.1);
7     std::cout << d << std::endl; // используем точность cout по умолчанию (6 цифр)
8     std::cout << std::setprecision(17);
9     std::cout << d << std::endl;
10    return 0;
11 }
```

Результат выполнения программы:

0.1
0.100000000000000001

Первый cout выводит 0.1 (что и ожидаемо). После того, как мы изменили для объекта cout точность вывода до 17 цифр, мы увидели, что значением переменной d является не совсем 0.1! Подобное происходит из-за ограничений в количестве выделяемой памяти для переменных типа double, а также из-за необходимости «округлять» числа. По факту мы получили типичную **ошибку округления**.

Подобные ошибки могут иметь неожиданные последствия:

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     std::cout << std::setprecision(17);
7
8     double d1(1.0);
```

```
9 | std::cout << d1 << std::endl;
10 |
11 | double d2(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1); // должно
12 | std::cout << d2 << std::endl;
13 | }
```

Результат выполнения программы:

```
1
0.99999999999999989
```

Хотя мы ожидали, что `d1` и `d2` окажутся равными, но это не так. А что, если бы нам довелось сравнивать эти переменные и, исходя из результата, выполнять определенный сценарий? В таком случае ошибок нам не миновать.

Математические операции (например, сложение или умножение), как правило, только увеличивают масштаб этих ошибок. Даже если `0.1` имеет погрешность в 17-й значащей цифре, то при выполнении операции сложения десять раз, ошибка округления переместится к 16-й значащей цифре.

nan и inf

Есть две специальные категории чисел типа с плавающей запятой:

- **inf** (или «*бесконечность*», от англ «*infinity*»), которая может быть либо положительной, либо отрицательной.
- **nan** (или «*не число*», от англ «*not a number*»). Их есть несколько видов (обсуждать все виды здесь мы не будем).

Рассмотрим примеры на практике:

```
1 | #include <iostream>
2 |
3 | int main()
4 | {
5 |     double zero = 0.0;
6 |     double posinf = 5.0 / zero; // положительная бесконечность
7 |     std::cout << posinf << "\n";
8 |
9 |     double neginf = -5.0 / zero; // отрицательная бесконечность
10 |    std::cout << neginf << "\n";
11 |
12 |    double nan = zero / zero; // не число (математически некорректно)
13 |    std::cout << nan << "\n";
14 |
15 |    return 0;
16 | }
```

Результат выполнения программы:

```
inf
-inf
-nan(ind)
```

`inf` означает «бесконечность», а `ind` означает «неопределенный» (от англ. «*indeterminate*»). Обратите внимание, результаты вывода `inf` и `nan` зависят от компилятора/архитектуры компьютера, поэтому ваш результат выполнения вышеприведенной программы может отличаться от моего результата.

Заключение

Переменные типа с плавающей точкой отлично подходят для хранения очень больших или очень маленьких (в том числе и дробных) чисел до тех пор, пока они имеют ограниченное количество значащих цифр (не превышают точность определенного типа данных).

Переменные типа с плавающей точкой могут иметь небольшие ошибки округления, даже если точность типа не превышена. В большинстве случаев такие ошибки остаются незамеченными, так как они не столь значительны. Но следует помнить, что сравнение переменных типов с плавающей точкой может иметь неопределенные последствия/результаты (а выполнение математических операций с такими переменными может только увеличить масштаб этих ошибок).

Тест

Запишите следующие числа в экспоненциальной записи в стиле языка C++ (используя букву `e`, как символ экспонента) и определите, сколько значащих цифр имеет каждое из следующих чисел:

- 34.50
- 0.004000
- 123.005
- 146000
- 146000.001
- 0.0000000008
- 34500.0

Ответ

- 3.450e1 (4 значащие цифры).
- 4.000e-3 (4 значащие цифры).
- 1.23005e2 (6 значащих цифр).
- 1.46e5 (3 значащие цифры).
- 1.46000001e5 (9 значащих цифр).
- 8e-10 (1 значащая цифра). Здесь мантисса не 8.0, а 8, поэтому число и имеет только 1 значащую цифру.

→ 3.45000e4 (6 значащих цифр). Здесь конечные нули не игнорируются, так как в исходном числе есть точка, которая разделяет целую и дробную части. Хотя эта точка никак не влияет на само число, она влияет на его точность. Если бы исходное число было указано, как 34500, то ответ равнялся бы 3.45e4.

Оценить статью:

★★★★★ (376 оценок, среднее: 4,84 из 5)



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

[Урок №34. Логический тип данных bool](#) →



Комментариев: 42



1. Яна:

[15 мая 2020 в 13:18](#)

Доброго времени суток. Решила написать простенькую программу определения эффективности пестицидов. Всё работает, только хотелось бы чтобы результат округлялся до целого числа, а не просто отбрасывало все что после запятой.

```
1 #include <iostream>
2 #include <cmath>
3
4 // getValueFromUser получает значение от пользователя, а затем возвращает его
5
6 int getValueFromUser()
7 {
8     std::cout << "Enter an integer: ";
9     int x;
10    std::cin >> x;
11    return x;
12 }
13
14 float main()
15 {
16     int a = getValueFromUser(); // первый вызов функции getValueFromUser
17     int b = getValueFromUser(); // второй вызов функции getValueFromUser
18     double f(100 * (a - b) / a); // формула биол.эф. пестицидов
19     std::cout << "(" << a << "-" << b << ")" << "/" << a << "*100" << "= "; //
20     std::cout << f; // вывод результата
21
22     return 0;
23 }
```

[Ответить](#)

1. *Виктор:*
[27 мая 2020 в 21:44](#)

Вы можете просмотреть правила округления чисел и внести дополнения в ваш код. Но как мне видится будет это непростое решение.

[Ответить](#)

2. *tesla:*
[28 мая 2020 в 20:01](#)

<http://www.cplusplus.com/reference/cmath/round/>

[Ответить](#)

3. *Сергей:*
[5 июня 2020 в 23:17](#)

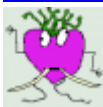
Я немного не по теме, но очень важное замечание: вот Вы программист в компании и собрались в отпуск\больничный\декрет и т.д. Я тоже программист в этой же компании и мне поручили работать над кодом, который писали Вы. Я открываю проект и первое, что делаю это читаю Ваши комментарии к коду, чтобы понять Вашу логику, как Вы провели пользователя через программу, возникавшие трудности и их решения. И что же я вижу? "Первый вызов функции...", "второй вызов функции...", "получает значение от пользователя и возвращает в caller..." Что?!! Ваши комментарии просто дублируют то, чем занимается каждый стейтмент? Это самые бесполезные комментарии, которые возможно придумать! Не пишите комментарий, который отвечает на вопрос "Что?" Пишите комментарии, которые отвечают на вопросы "зачем", "как" и "почему". Вместо "получает значение от пользователя" напишите "пользователь вводит массу\объем\площадь и т.д." комментарии типа "первый вызов функции" вообще не нужны. Или пометьте, что переменные a и b символизируют (те же массу\объем\площадь и т.д.). Таким образом Вы мало того, что оживите Ваш код, придав сухим цифрам и буквам смысла жизни, но и облегчите читаемость и воспринимаемость кода не только другим людям, но и, поверьте, самой себе.

[Ответить](#)

4. *ШО?:*
[10 июня 2020 в 15:00](#)

`floor()` — функция возвращающая округлённое число в большую сторону

требуется `#include cmath`

[Ответить](#)

5. *Илья:*
[23 сентября 2020 в 21:02](#)

Ошибка! В скобках f все действия будут проводиться в целых числах. В самом конце компилятор приведёт-таки полученное целое число к виду числа с плавающей точкой.

[Ответить](#)



2. *Борис:*

[25 апреля 2020 в 21:25](#)

Хорошая статья! Конечно, не такая подробная как могла бы быть, но здесь сразу описаны многие подводные камни компьютерной арифметики с плавающей точкой, о которых новички даже не знают. Причём так не только в C++, в других языках всё так же с этими стандартными типами. Это действительно довольно коварная тема. Подробнее можно глянуть например тут:

<https://habr.com/ru/post/112953/>

<https://habr.com/ru/post/322984/>

[Ответить](#)



3. *Хей Лонг:*

[9 апреля 2020 в 17:30](#)

Простите, я всё-таки немного глупый и недоверчивый. $34,50 = 34,5$ насколько мне известно из математики. пишу программу, вычитающую два вводимых double. получается так: $3.45e1 - 3.450e1 = 0$. на всякий случай ввёл в обратном порядке, ответ не изменился. выходит, правые нули в значимых цифрах после точки не нужны. либо речь ведётся не о математических величинах, а о формах графического представления, либо мне для понимания нужен другой пример.

$3.45e1$

$3.45000e1$

Их разность: 34.5 минус $34.5 = 0$

Для продолжения нажмите любую клавишу . . .

[Ответить](#)



1. *Константин:*

[12 мая 2020 в 23:17](#)

Хей, Джо (бррр в смысле лонг)! Количество знаков справа от точки предписывают какой, например, измерительный инструмент необходим для точного измерения расстояний (циркуль землемера, сантиметр портного, микрометр инструментальщика, колаидр ядерного физика)

[Ответить](#)



2. *Виктор:*

[27 мая 2020 в 21:48](#)

Эти нули говорят о точности числа. В целом при вычислениях будет определяться до какого разряда производить округления. 34.5 -до десятых, 34.50 до сотых

[Ответить](#)



1. Роман:

[21 сентября 2020 в 07:45](#)

Какой точности? Можно хоть миллион нулей добавить, суть от этого не изменится, потому что значащих цифр (разрядов) там три.

А про округление там ничего не сказано, поэтому по умолчанию сокращается по максимуму. Учите матчасть.

[Ответить](#)



4. Sasha:

[9 января 2020 в 00:33](#)

Скажите, а что делать если число не помещается даже в long long. Вчера увидел задачку, где целочисленной переменной по условию варьируется от 0 до 100^{100} . Это же никакая переменная в себя не вместит

[Ответить](#)



1. Верус:

[2 мая 2020 в 05:19](#)

Сохранять цифры в массивах и написать руками функции для арифметических операций с ними (это называется длинная арифметика)

[Ответить](#)



5. Абакар:

[24 декабря 2019 в 23:05](#)

Здравствуйте. А почему если поменять double на float, то ошибка уходит и выводит 1?

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     std::cout << std::setprecision(17);
7
8     double d1(1.0);
9     std::cout << d1 << std::endl;
10
11     double d2(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1); // д
12     std::cout << d2 << std::endl;
13 }
```

Понять не могу?

Но float не работает в коде котором ниже

```
1 #include <iostream>
2 #include <iomanip> // для std::setprecision()
3
4 int main()
5 {
6     double d(0.1);
7     std::cout << d << std::endl; // используем точность cout по умолчанию (6 цифр)
8     std::cout << std::setprecision(17);
9     std::cout << d << std::endl;
10    return 0;
11 }
```

Объясните пожалуйста

[Ответить](#)



6. *Владимир:*

[28 ноября 2019 в 01:29](#)

Еще один вопрос по ошибкам округления. Правильно ли я понимаю, что они вызваны просто включением слишком большой точности (17 знаков), и, как следствие переполнение. В разряды ниже 15 знака вследствие этого попадает просто случайный мусор, который округляется и имеем эти фокусы. По крайней мере, многократно экспериментируя с кодом из урока, при понижении точности до 15 знаков и ниже подобных неточностей округления не наблюдал.

[Ответить](#)



7. *Владимир:*

[28 ноября 2019 в 01:09](#)

Всем добра! Такой вопрос относительно nan & inf. Правильно ли я понял, что некорректным в C++ есть деление только на целочисленный ноль, а деление на 0 типа с плавающей точкой корректно и в результате мы и получаем эти величины?

[Ответить](#)

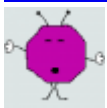


8. *Максим:*

[26 августа 2019 в 08:35](#)

Все очень мильно

[Ответить](#)



9. *Александр:*

[30 января 2019 в 12:39](#)

поправка: std::setprecision это не функция... это такой вид аргумента (операнда) для оператора вывода в поток — манипулятор потока.

есть функция-член для объекта `std::cout` — `precision`
использование:

```
1 | std::cout.precision(6); // устанавливает точность 6
```

преимущество — не нужно подключать никаких дополнительных библиотек. Этот метод реализован в любом объекте `std::ostream`

[Ответить](#)



1. Константин:
[9 марта 2019 в 19:47](#)

Ой-ё-ёй, Александр, постойте! Позвольте два вопросика Вам задать: 1- Я отдельным файлом создаю `patespase` (ну само-собой подключаю его где нужно в программе) в котором пользователь в режиме `RUNTIME` заносит некие константы, востребованные на всём протяжении работы проги и практически во всех функциях (файлах). Так как их перечень довольно большой, легко перепутать в какую ячейку какое значение вносить, а выводить на экран предложение внести желаемое значение в ПОИМЕНОВАННУЮ ячейку я никак не могу сообразить способ... 2 — Как активировать `debugging` в `Code::Blocks 17.12`? Можешь подсказать?

[Ответить](#)



10. somebox:
[13 ноября 2018 в 05:00](#)

Я вот не понял, если мы определили переменную как `float`, зачем в литерале использовать `f`?

[Ответить](#)



1. vLasTT:
[3 января 2019 в 12:19](#)

В C++, при инициализации переменных типа `float` в следующем стиле :

```
1 | float f = 0.0;
```

`f` будет иметь тип `double`. Вы можете убедиться в этом, наведя курсор на переменную. Поэтому, чтобы иметь тип `float`, нужно добавить суффикс `f` и тогда переменная будет иметь тип `float`.

[Ответить](#)



1. Борис:
[25 апреля 2020 в 20:57](#)

Нет. Ты видимо навёл курсор не на переменную, а на значение (цифры). Оно действительно изначально понимается как `double`, но тип переменной объявлен как `float`, значит при выполнении это число будет сконвертировано в `float`, чтобы быть присвоенным `float`-переменной.

[Ответить](#)

11. Светлана:

[26 октября 2018 в 15:01](#)

Здравствуйте.

у меня не работает ни мой вариант программы, ни ваш. Делаю с обычными числами — всё ок. Экспоненциальная запись — вот такая вот штука происходит:

Enter your number:

3e3

Enter your number:

Your result is:-858993457

Для продолжения нажмите любую клавишу . . .

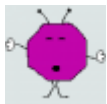
вот сама программа:

```

1  #include <iostream>
2
3  int yourNumber()
4  {
5      printf("Enter your number:\n");
6      long long x;
7      scanf_s("%d", &x);
8      return x;
9  }
10 void summa(long long result)
11 {
12     printf("Your result is:%d \n", result);
13 }
14 int main()
15 {
16     long long x = yourNumber();
17     long long y = yourNumber();
18     summa(x + y);
19     return 0;
20 }

```

прошу прощения, но я не нашла как загрузить его сюда в том виде, в котором загружаете вы

[Ответить](#)

1. Александр:

[30 января 2019 в 12:41](#)

Вы пытаетесь прочитать целое число, а вводите дробное...

[Ответить](#)

12. Константин:

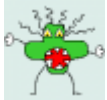
[10 сентября 2018 в 13:41](#)

```
#include <Здравствуй, Юра>
```

Подготавливая к представлению $1/10$ в десятичной системе я слева пишу "1", далее слева на право ставлю уголок, справа вверху пишу "10", снизу уголка записываю результаты деления... В итоге вашему вниманию представлено число 0.1! А как в двоичной системе происходит подготовка числа к представлению?

Ещё вопрос — в десятичной системе правило округления, например, 1.1 .2 .3 .4 округляется до 1, а 1.5 .6 .7 .8 .9 округляются до 2. В C++ тоже есть правила округления или лишнее просто отсекается?

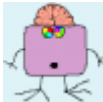
```
#endif
```

[Ответить](#)


13. *Ярослав:*

[30 июля 2018 в 21:40](#)

А зачем нули в конце числа (уже после запятой и после остальных цифр, например, как последние 2 нуля в 0.0078900) оставлять? В математике-то их же отбрасывают обычно (то есть до 0.00789)

[Ответить](#)


1. *Елена:*

[5 августа 2018 в 15:42](#)

Ноль тоже считается, так как 34.50 было плавающим числом. Поэтому $3.450e1$, а не $3.45e1$.

[Ответить](#)


1. *S:*

[22 декабря 2018 в 22:26](#)

Хахаха,умно конечно...(паходу типер я никрапастер*sad*)

[Ответить](#)


14. *Виктор:*

[30 июля 2018 в 09:11](#)

В тесте задание "а", разве крайний правый значащий? В ответе написано 4 значащих цифры, но ведь их 3

[Ответить](#)


15. *Андрей:*

[17 июля 2018 в 15:40](#)

Скажите, а как избежать небольших ошибок округления, если необходима абсолютная математическая точность при работе с числами с плавающей запятой?

[Ответить](#)



16. Алибек:

[11 мая 2018 в 13:38](#)

а в чем разница между int и long если оба 4 байта

[Ответить](#)

1. Юрий:

[13 мая 2018 в 20:03](#)

Отличаются тем, что имеют разный размер на разных ОС/платформах. Т.е. по факту, int должен был бы занимать 2 байта, а long — 4, но в системе Windows они ничем не отличаются, так как оба занимают 4 байта. Однако если вы будете разрабатывать кроссплатформенную программу или приложение, то лучше использовать long вместо int, если требуется больший диапазон значений.

[Ответить](#)

17. Sergey Groysman:

[7 апреля 2018 в 20:07](#)

Юрий, прошу прощения, что "разбросал" вопросы, а не собрал вместе. Программа понимает только ввод "правильных" чисел или при вводе можно ввести 3e3 вместо 3000? Просто у меня при вводе экспоненциальной записи числа получается какая-то ерунда, программа её не понимает. Спасибо.

[Ответить](#)

1. Юрий:

[11 апреля 2018 в 13:49](#)

Вводить числа с указанием буквы «е» можно, только эта буква «е» пишется на английской раскладке. Если вы пишете «е» с русской раскладки, то получите ошибку.

Ничего, так даже лучше (насчет разбросанных вопросов). Пожалуйста 😊

[Ответить](#)

18. Sergey Groysman:

[7 апреля 2018 в 19:56](#)

Юрий, ещё раз здравствуйте.
Я написал простую программу:

```
1 #include "stdafx.h"
2 #include <iostream>
3
4 int readNumber()
```

```
5 {
6     std::cout << "Please, enter an integer and press button ENTER: " << std::endl;
7     int x;
8     std::cin >> x;
9     return x;
10 }
11
12 void writeAnswer(int x)
13 {
14     std::cout << "Sum of your numbers is " << x << std::endl;
15 }
16
17 int main()
18 {
19     int x = readNumber();
20     int y = readNumber();
21     writeAnswer(x + y);
22
23     return 0;
24 }
```

поскольку переменные простые, то для проверки я ввёл: 3000000000 (3e9) для первого числа и на второе уже места не осталось. Просто вышла какая-то ерунда.

Но я решил изменить "размер" места под большие числа и немного переделал программу:

```
1 #include "stdafx.h"
2 #include <iostream>
3
4 int readNumber()
5 {
6     std::cout << "Please, enter an integer and press button ENTER: " << std::endl;
7     long x;
8     std::cin >> x;
9     return x;
10 }
11
12 void writeAnswer(int x)
13 {
14     std::cout << "Sum of your numbers is " << x << std::endl;
15 }
16
17 int main()
18 {
19     long x = readNumber();
20     long y = readNumber();
21     writeAnswer(x + y);
22
23     return 0;
24 }
```

программа отладку прошла, но проблема осталась. Где я не правильно сделал (возможно не доделал) или я тогда не совсем понял, ГДЕ мы указываем переменным "размер" оперируемыми

числами.
Спасибо.

[Ответить](#)

R

1. Юрий:

[11 апреля 2018 в 13:41](#)

Привет. По порядку.

1. Переменная `int`, скорее всего, на вашем компьютере занимает 4 байта (у меня так). Диапазон для 4 байтов — от -2 147 483 648 до 2 147 483 647 (из [урока 31](#)). Вы ввели 3 000 000 000 — это превышает допустимый диапазон для значений типа `int`, поэтому вы и получили ошибку.

2. Тип `long` также занимает 4 байта, соответственно, диапазон тот же, что и для `int`. Тип `long long` занимает 8 байтов, его и следовало бы использовать. Диапазон `long long` — от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

3. Во второй программе вам нужно использовать тип `long long` и указывать его не только в `main()` при определении переменных `x` и `y`, но и для функций `readNumber()` и `writeAnswer`, а то получится, что указали `long long` для `x` и `y`, а функция, которая принимает данные от пользователя и затем возвращает их — имеет тип `long` (произойдет ошибка, так как число обрежется самой функцией).

Рабочая программа:

```
1  #include "stdafx.h"
2  #include <iostream>
3
4  long long readNumber()
5  {
6      std::cout << "Please, enter an integer and press button ENTER: " << std::endl;
7      long long x;
8      std::cin >> x;
9      return x;
10 }
11
12 void writeAnswer(long long x)
13 {
14     std::cout << "Sum of your numbers is " << x << std::endl;
15 }
16
17 int main()
18 {
19
20     long long x = readNumber();
21     long long y = readNumber();
22
23     writeAnswer(x + y);
24
25     return 0;
```

26 | }

[Ответить](#)19. *Sergey Groysman:*[7 апреля 2018 в 18:44](#)

Юрий, доброго дня.

Подскажите, если для хранения переменных целочисленной и с плавающей точкой достаточно 4 байт и разбег чисел соответственно от $+2e9$ до $-2e9$, то наиболее часто встречающиеся и используемые команды `long` и `float`, правильно? если уж понадобятся "запредельные" цифры то `long long` или `doble` (`long double`)?

И ещё, а они не сокращаются например до `f` для `float` или `l` для `long`, например `fx`; при присвоении переменной с плавающей точкой?

Спасибо.

[Ответить](#)1. *Юрий:*[11 апреля 2018 в 13:26](#)

Привет, Сергей.

1. Наиболее часто встречающийся целочисленный тип — `int`. Тип с плавающей запятой — `double`. Но это ведь не значит, что нужно использовать только эти типы данных. Для каждой задачи отдельно подбирается наиболее подходящий тип.

2. Если понадобятся запредельные цифры, то да — `long long` для целочисленного типа и `long double` для типа с плавающей запятой.

3. Нет, типы данных не сокращаются по умолчанию вообще. Но вы можете использовать псевдонимы для типов (детальнее об этом в [уроке 60](#)).

Пожалуйста 😊

[Ответить](#)20. *aleksandr:*[16 марта 2018 в 10:41](#)

Скажите пожалуйста, в дальнейших уроках будете более глубоко разбирать числа с плавающей точкой ?

Или это на первых порах не нужно ?

[Ответить](#)1. *Юрий:*[17 марта 2018 в 00:54](#)

В дальнейшем, урока поподробнее о числах с плавающей запятой, нежели этот, не будет. То, что нужно — здесь изложено.

[Ответить](#)21. *Виталий:*[29 сентября 2017 в 01:51](#)

Пока что это самая тяжёлая статья из всех 33 мной прочитанных.

[Ответить](#)1. *cybersatori:*[7 января 2020 в 06:26](#)

+

[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