4.7 — Introduction to scientific notation

BY ALEX ON APRIL 23RD, 2019 | LAST MODIFIED BY ALEX ON DECEMBER 11TH, 2019

Before we talk about our next subject, we're going to sidebar into the topic of scientific notation.

Scientific notation is a useful shorthand for writing lengthy numbers in a concise manner. And although scientific notation may seem foreign at first, understanding scientific notation will help you understand how floating point numbers work, and more importantly, what their limitations are.

Numbers in scientific notation take the following form: *significand* x $10^{exponent}$. For example, in the scientific notation `1.2 x 10^4`, `1.2` is the significand and 4 is the exponent. Since 10^4 evaluates to 10,000, 1.2 x 10^4 evaluates to 12,000.

By convention, numbers in scientific notation are written with one digit before the decimal, and the rest of the digits afterward.

Consider the mass of the Earth. In decimal notation, we'd write this as `5973600000000000000000000` kg. That's a really large number (too big to fit even in an 8 byte integer). It's also hard to read (is that 19 or 20 zeros?). Even with separators (5,973,600,000,000,000,000,000,000) the number is still hard to read.

In scientific notation, this would be written as `5.9736 x 10^24` kg, which is much easier to read. Scientific notation has the added benefit of making it easier to compare the magnitude of two really large or really small numbers simply by comparing the exponent.

Because it can be hard to type or display exponents in C++, we use the letter 'e' (or sometimes 'E') to represent the "times 10 to the power of" part of the equation. For example, `1.2 x 10^4` would be written as `1.2e4`, and `5.9736 x 10^24` would be written as `5.9736e24`.

For numbers smaller than 1, the exponent can be negative. The number `5e-2` is equivalent to `5 * 10^-2`, which is `5 / 10^2`, or `0.05`. The mass of an electron is `9.1093822e-31` kg.

How to convert numbers to scientific notation

Use the following procedure:

- Your exponent starts at zero.
- Slide the decimal so there is only one non-zero digit to the left of the decimal.
  - Each place you slide the decimal to the left increases the exponent by 1.
  - Each place you slide the decimal to the right decreases the exponent by 1.
- Trim off any leading zeros (on the left end of the significand)
- Trim off any trailing zeros (on the right end of the significand) only if the original number had no decimal point. We're assuming they're not significant unless otherwise specified.

Here's some examples:

```
Start with: 42030
Slide decimal left 4 spaces: 4.2030e4
No leading zeros to trim: 4.2030e4
Trim trailing zeros: 4.203e4 (4 significant digits)


Start with: 0.0078900
Slide decimal right 3 spaces: 0007.8900e-3
```

```
Trim leading zeros: 7.8900e-3
Don't trim trailing zeros: 7.8900e-3 (5 significant digits)


Start with: 600.410
Slide decimal left 2 spaces: 6.00410e2
No leading zeros to trim: 6.00410e2
Don't trim trailing zeros: 6.00410e2 (6 significant digits)
```

Here's the most important thing to understand: The digits in the significand (the part before the 'e') are called the **significant digits**. The number of significant digits defines a number's **precision**. The more digits in the significand, the more precise a number is.

Precision and trailing zeros after the decimal

Consider the case where we ask two lab assistants each to weigh the same apple. One returns and says the apple weighs 87 grams. The other returns and says the apple weighs 87.00 grams. Let's assume the weighing is correct. In the former case, the actual weight of the apple could be anywhere between 86.50 and 87.49 grams. Maybe the scale was only precise to the nearest gram. Or maybe our assistant rounded a bit. In the latter case, we are confident about the actual weight of the apple to a much higher degree (it weighs between 86.9950 and 87.0049 grams, which has much less variability).

So in standard scientific notation, we prefer to keep trailing zeros after a decimal, because those digits impart useful information about the precision of the number.

However, in C++, 87 and 87.000 are treated exactly the same, and the compiler will store the same value for each. There's no technical reason why we should prefer one over the other (though there might be scientific reasons, if you're using the source code as documentation).

Now that we've covered scientific notation, we're ready to cover floating point numbers.

Quiz time

Question #1

Convert the following numbers to C++ style scientific notation (using an e to represent the exponent) and determine how many significant digits each has (keep trailing zeros after the decimal):

a) 34.50

Show Solution

b) 0.004000

Show Solution

c) 123.005

Show Solution

d) 146000

Show Solution

e) 146000.001

Show Solution

f) 0.0000000008

Show Solution

g) 34500.0

Show Solution

4.8 -- Floating point numbers

Index

4.6 -- Fixed-width integers and size_t

C++ TUTORIAL | PRINT THIS POST

19 comments to 4.7 — Introduction to scientific notation

Scarlet Johnson
February 7, 2020 at 11:36 am · Reply

Explain this "though there might be scientific reasons, if you’re using the source code as documentation".

Alex
February 9, 2020 at 11:39 am · Reply

The C++ compiler doesn't care whether you use 87 or 87.000. But, in standard scientific notation, we prefer to keep the trailing zeros since they're useful in determining actual

precision. If someone is reading your program, they might care that it was 87 vs 87.000 (e.g. if they're looking to understand how precise you were being in the first place).

Bruno
February 3, 2020 at 1:32 pm · Reply

On question F "f) 0.0000000008"
I answered 0.8e-9. Is is too wrong? For programming?
I don't remember if i should stick to 8 or 0.8 on "scientific" terms also. 0.8 feels very natural to me.

"Trim off any leading zeros (on the left end of the significand)
Trim off any trailing zeros (on the right end of the significand) only if the original number had no decimal point. We're assuming they're not significant unless otherwise specified."

This goes for programming and scientific terms right? So much time since i learned this, i actually forgot the standards.

koe
December 17, 2019 at 3:55 pm · Reply

Let's say I have a program that reads values with trailing zeroes from a file, does some math on them, then prints to the console. This chapter suggests the value printed to console will not have trailing zeroes if the math was done with floating point numbers, since C++ doesn't care about those.

If I wanted to correctly maintain significant digits throughout mathematical operations, how would I do so?

nascardriver
December 18, 2019 at 4:33 am · Reply

You'll have to count the digits after the decimal point and keep track of them. I wrote you a type and iostream functions that preserve the precision. You'll understand how it works after chapter 9.

```
1  #include <charconv> // std::from_chars
2  #include <cstdlib> // std::strtod
3  #include <iomanip> // std::setprecision
4  #include <iostream>
5  #include <string>
6  #include <system_error> // std::errc
7
8  struct SPreciseDouble
9  {
10   double dbValue{};
11   int iPrecision{};
12 };
13
14 std::istream& operator>>(std::istream& s, SPreciseDouble& db)
15 {
16   std::string str{};
17
18   s >> str;
19
20   // Use std::from_chars if possible.
21   if (auto [p, ec]{ std::from_chars(str.c_str(), str.c_str() + str.length(), &db.dbVa
22   {
23     s.setstate(std::ios::failbit);
24     return s;
25   }
```

```
26
27    // If std::from_chars is not available
28    // char* szEnd{};
29    // db.dbValue = std::strtod(str.c_str(), &szEnd);
30    // if (szEnd != (str.c_str() + str.length()))
31    // {
32    //   s.setstate(std::ios::failbit);
33    //   return s;
34    // }
35
36    if (auto found{ str.find('.') }; found != std::string::npos)
37    {
38      db.iPrecision = static_cast<int>((str.length() - found) - 1);
39    }
40
41    return s;
42  }
43
44  std::ostream& operator<<(std::ostream& s, const SPreciseDouble& db)
45  {
46    std::cout << std::fixed << std::setprecision(db.iPrecision) << db.dbValue;
47
48    return s;
49  }
50
51  int main()
52  {
53    SPreciseDouble db{};
54
55    std::cin >> db;
56
57    db.dbValue = 14.0;
58
59    std::cout << db;
60
61    return 0;
62  }
```

I wasn't able to test `std::from_chars`, because my standard implementation is lacking behind. If it doesn't work, comment line 21-25 and uncomment 28-34.

Example input/output

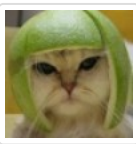
```
10.000
14.000

1.12
14.00

1
14
```

Ylis taas vauhdis
December 10, 2019 at 9:19 am · Reply

Hello
I think there is a logic error with the roundings. The weight of the second apple was 87.000g, so it's at the precision of 5 numbers.
The apple cant be between 86.9950g and 87.0049g, since the first one rounded to the precision of 5 numbers

would be 86.995 and the second one 87.005. Therefore the second apple can be between weights of 86.9995g to 87.00049.

If I am mistaken please remove my comment to not confuse others. I hope my english was understandable, not my first nor second language ;)

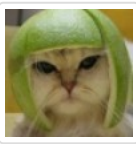


Alex
December 11, 2019 at 10:20 pm · Reply

I believe you are correct. So as not to make the numbers too hard to read, I reduced the precision of "87.000" to "87.00".

Lee
November 28, 2019 at 9:35 am · Reply

In the last example of the quiz, I don't understand why it is 3.45000e4 instead of 3.45e4, in the former lecture you said that it doesn't matter if the number is whole(integer) or has .000 at the end. E.g. 34500 and 34500.0 will be same to compiler and both evaluated to the same value.

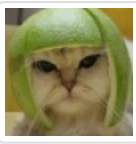


Alex
December 2, 2019 at 5:41 pm · Reply

True, but the quiz instructions say: "keep trailing zeros after the decimal"

Paul
September 23, 2019 at 4:36 pm · Reply

There is error in top of the page saying 10e4 is 10000 when its really 100000



Alex
September 24, 2019 at 9:22 am · Reply

It's not saying that. It's saying 10^4 = 10,000, and so 1.2 x 10^4 = 12,000.

Yann
September 1, 2019 at 11:45 am · Reply

Hello,

For the quiz, at question 1d, I don't fully agree with you when you state "trailing zeros in a whole number with no decimal are not significant.".

IMO, if 146000 has only 3 significant digits, it means it can be a rounding of any value between 145500 and 146499 included... So for me, 146000 has really 6 significant digits.

Thanks for your great tutorial. I rediscover C++ after long years where I didn't practice it and is a good refresh and an opportunity to learn what have been introduced with newer C++ flavors.

Raton
January 27, 2020 at 10:46 am · Reply

I had the exact same reaction, but I think it's just easier to count it as having 3 significant digits. Because if we count the trailing zeros in whole numbers with no decimal as

significant digits, then if you only want 3 significant digits you'd have to use a power of ten, and maybe it's just more practical not to.
That being said, it means that if you want to display that number with 6 significant digits, you'd use the scientific notation (146e3), and if you want only 3, you'd write the number without the power of ten (146000), so maybe it's not a big deal (even though it's not very consistent with the rest).

1357908642
August 17, 2019 at 6:51 am · Reply

I just noticed there is a lack of comments on this lesson,
probably because there is a bunch of math included.

Just wanted to say hi and thank you for all of this OP.

Kamogelo Tlotleng
June 26, 2019 at 4:51 am · Reply

Thank you for this lesson as I was a bit disappointed that you kept mentioning scientific notation/Avogardo's number without giving any proper explanations, when you were giving your float variable lessons.

I am glad you finally made a lesson on this.

2+2=5
May 18, 2019 at 11:28 am · Reply

This is being a math lesson (for kids) more than being a C++ lesson
(Although kids wont learn CPP but...Meh :/

p1n
June 16, 2019 at 9:28 am · Reply

I'm 16 and i'm learning it.

Rares Vanca
June 30, 2019 at 10:42 pm · Reply

I'm 13 and i am learning c++. I started coding at 10.

Jose
August 2, 2019 at 6:20 pm · Reply

I´m 39, I couldn´t study when I was younger so I´m thankful for this lesson.