**The f32: how floating-point arithmetic broke my Rust code**

[[](https://medium.com/@aizon?source=post_page-----19b9cf0bb76c--------------------------------)](https://medium.com/@aizon?source=post_page-----19b9cf0bb76c--------------------------------)

[Onotieose Izormen](https://medium.com/@aizon?source=post_page-----19b9cf0bb76c--------------------------------)

·

Follow

3 min read

·

Jan 14

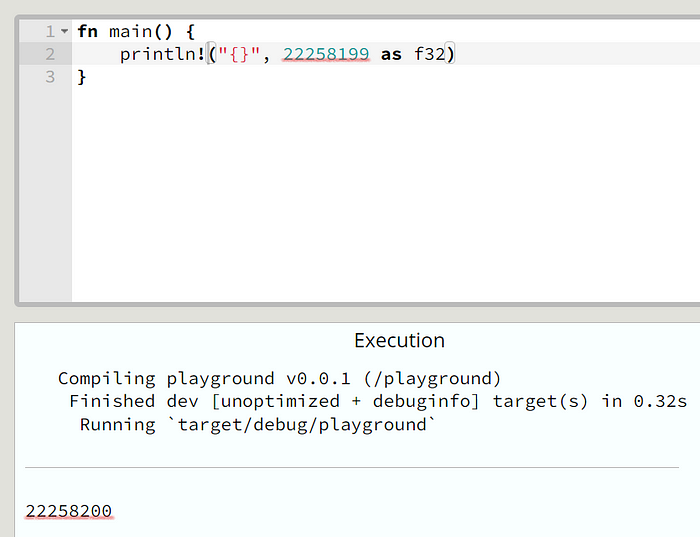
3



On the 2nd of January 2023, I made a quick decision to solve a [\*1200 problem on codeforces](https://codeforces.com/contest/492/problem/B). I read the problem, analysed carefully, wrote some code, and *Submit!*…My code failed. I tried once again and hit “Submit”, it failed once again. At this point I was fully confident in my code and decided to give the error a look. Here’s what I saw:

Output:  
22258200  
  
Answer:  
22258199.5000000000  
  
Checker Log:  
wrong answer 1st numbers differ - expected: '22258199.5000000', found: '22258200.0000000', error = '0.0000000'

The output of the code was the correct answer rounded up to the nearest unit. This seemed rather suspicious, hence I tried several other cases on my local machine, all producing the same result — The correct answer, rounded up to the nearest unit.



I asked about this on [twitter](https://twitter.com/HippyDreaded/status/1742114222899712148) and received a few responses. The simple answer : “Use a float64 instead of a float32”. But I also received [this response](https://twitter.com/sanguine_skies/status/1742124455592087774) from [@sanguine\_skies](https://twitter.com/sanguine_skies) which sent me on a journey of understanding foating-point arithmetic. Here’s what I learnt:

In computing, **floating-point arithmetic** (**FP**) is arithmetic that represents subsets of real numbers using an integer with a fixed precision/length, called the **significand/mantissa**, multiplied by an integer exponent of a fixed base. Numbers of this form are called **floating-point numbers**. e.g

12.345 is a floating point number, represented as an integer with five digits of precision(i.e length of “12345" is 5) multiplied by an integer exponent of a fixed base (in this case, exponent: -3; base: 10)

**12.345 = 12345 x 10⁻³**

In my solution to the problem, I used Rust’s f32 type which is a 32-bit floating-point type. This type has **24 bits of mantissa**, and this means it can only represent exactly, integers whose absolute value is less than, or equal to **2²⁴**( *approx.****16777216***). The number I have above is 22258199 which is about 22 million, much more than the 16 million which f32 can represent exactly.

So what happens when you try to convert to f32, a number that cannot be exactly represented as f32? Simply put, the processor rounds to the nearest representable f32 number (which in this case is 22258200). This **rounding error** in the output caused my code to continually fail. You can read more about Rust’s f32 type [here](https://doc.rust-lang.org/std/primitive.f32.html).

Now, if we convert 22258199 to f64 instead, we get back its exact value. This is because the f64 type holds **52 bits of mantissa** hence it can exactly represent integers whose absolute value is less than or equal to 2⁵²(*approx.***4.5035996e+15**) which is a whole lot more than can be represented by an f32 and exactly what we need to prevent rounding errors in our output(*at least for this present scenario)*.



If you’re a bit more curious about floating-point arithmetic and its inner workings, you could check out [**IEEE Standard for Floating-Point Arithmetic** (**IEEE 754**)](https://en.wikipedia.org/wiki/IEEE_754).

Thanks for reading! I hope you enjoyed this article and learned a thing or two about floating-point arithmetic. ✨

[Floating Points](https://medium.com/tag/floating-points?source=post_page-----19b9cf0bb76c---------------floating_points-----------------)

[Rust](https://medium.com/tag/rust?source=post_page-----19b9cf0bb76c---------------rust-----------------)

[Codeforces](https://medium.com/tag/codeforces?source=post_page-----19b9cf0bb76c---------------codeforces-----------------)

[F32](https://medium.com/tag/f32?source=post_page-----19b9cf0bb76c---------------f32-----------------)

[Precision](https://medium.com/tag/precision?source=post_page-----19b9cf0bb76c---------------precision-----------------)