Why doesn’t reduction by modulo work for floating point values?

devblogs.microsoft.com/oldnewthing/20231106-00

November 6, 2023

Raymond Chen

It is commonly known that if you want to perform a calculation modulo an integer, you can apply the modulo operation at each step of the calculation, instead of carrying a huge value and then reducing it at the end.

```
uint32_t MultiplyMod10(uint32_t a, uint32_t b)
{
    // Naïve version: Overflow danger
    return (a * b) % 10;

    // Reduce modulo 10 after each step.
    return ((a % 10) * (b % 10)) % 10;
}
```

But this trick doesn’t work for floating point values.

```
constexpr double TwoPi = 6.283185307179586476925286766559005 8L;

double MultiplyModulo2Pi(double a, double b)
{
    // Naïve version gives correct answer
    return fmod(a * b, TwoPi);

    // But this gives the wrong answer
    return fmod(fmod(a, TwoPi) * fmod(b, TwoPi), TwoPi);
}
```

For example, trying to calculate 7.5 × 10 mod 2*π* using the naïve formula gives the correct value of 75 mod 2*π* ≈ 5.8850. But using the alternate formula gives the incorrect value of (1.2168 × 3.7168) mod 2*π* ≈ 4.5227.

A simpler demonstration of this problem is calculating (½ × 2) mod 2. The naïve version gives you the correct 1 mod 2 = 1. The flawed incorrect version gives you (½ mod 2) (2 mod 2) mod 2 = (½ × 0) mod 2 = 0 mod 2 = 0.

What’s going on? Is math broken?

Math is not broken. Your expectations are broken.

Let’s see why the integer modulo technique works.

a mod n = c means that $a = c + kn$ for some integer k.

Therefore

$(a \bmod n)\ (b \bmod n) =$
$(c_1 + k_1n)\ (c_2 + k_2n) =$
$c_1c_2 + c_1k_2n + c_2k_1n + k_1k_2n^2 =$
$c_1c_2 + (c_1k_2 + c_2k_1 + k_1k_2n)n =$
$c_1c_2 + k_3n$, where $k_3 = c_1k_2 + c_2k_1 + k_1k_2n$.

If a, b, and n are all integers, then so too will be $c_1k_2 + c_2k_1 + k_1k_2n$, and the final expression shows that then (a mod n) (b mod n) mod n = ab mod n.

But if a, b, and n are not all integers, then that expression for k_3 might not end up being an integer, and then the conclusion breaks down.

In other words, taking the modulus after each operation works with integers because the trick relies on the fact that adding and multiplying integers produces another integer. Once you throw floating point values into the formula, the end result is not guaranteed to be an integer any more.

Bonus chatter: Mathematically, the modulus operation is a ring homomorphism from the ring of integers to the ring of integers modulo n. Ring operations are addition, subtraction, and multiplication. Note that the trick doesn’t work with division: (2 ÷ 2) mod 2 = 1 mod 2 = 1, but ((2 mod 2) ÷ (2 mod 2)) mod 2 = (0 ÷ 0) mod 2 = nonsense.