Floating Point Woes

Floating Point is Simple

Floating point (FP) is a simple concept, it's just scientific notation in base 2.

In Base 10 the speed of light is $3.00 * 10^8 \ \mathrm{ms}^{-1}$ which is another way of reprensenting $300,000,000 \, \mathrm{ms}^{-1}$.

Scientific	**Integer**
$3.00 * 10^8 \ \mathrm{ms}^{-1}$	$300,000,000 \, \mathrm{ms}^{-1}$

In Base 2 we would express this as:

Floating Point	**Fixed Point**
$1.000111... * 2^{28}$	$(10001111000011010001100000000)_2$

The issue of big and small numbers is exasperated in base 2. Floating point allows us to deal with really large numbers and really small numbers without (a) requiring huge sizes or (b) requiring complicated operations for calculations.

Where the problems arise

We know that $1/3$ doesn't fit nicely into base 10:

$$\frac{1}{3} = 0.\overline{3} \approx 0.333333... \tag{1}$$

In scientfic notation we need to specify a number of *signigicant figures* (sf), i.e. how precise we want to go. So let's say 10 sf is enough so:

$$\frac{1}{3} = 3.333333333 * 10^{-1} \tag{2}$$

If you compute $1/3 + 1/3 + 1/3$ you expect 1, but if you used the scientific notation to 10 sf, you would get $0.999999999 \ (\neq 1)$. We know that it's 1, but a computer doesn't. The more significant figures we use the closer we get to 1, but we will *never reach it*.

Of course this problem is not specific to base 10, base 2 has similar issues. The confusion lies in the fact that we know base 10, we use it all the time, so we just know that $1/3 + 1/3 + 1/3 = 1$, but in base 2, the numbers that have rounding errors will be different. Remember a computer always deals with base 2 even if you don't.

In base 2, $1/10$ doesn't nicely fit in:

$$(\frac{1}{10})_{10} = (0.0\overline{0011})_2 \approx 0.000110011... \tag{3}$$

Let's use floating point representation, again to 10 sf:

$$(\frac{1}{10})_{10} = 0.00110011 * 2^{-1} \tag{4}$$

We know that $1/10 + 1/10 = 2/10$, but using our FP representation to 10 sf, you would get $(0.001100110)_2 = (0.19921875)_{10} \neq (0.2)_{10}$.

What Can I Do? (aka TL;DR)

For scientific calculations were precision (up to a point) is key double-precision FP (64 bits) or even single-precision FP (32 bits) is usually pretty good. For currency, it isn't. The solution is to treat pounds/dollars (£/$) and pence/cents (p/¢) seperately, as integers (fixed point), then divide by 100 at the end. Instead of `float amount = 12.10` use `int amountgbp = 12` and `int amountpence = 10` and if you need the amount together, concatenate them and divide by 100, `amount = amountgbp ++ amountpence / 100`

The *rule of thumb* for:

- **discrete** quantities (like currency, stock prices etc.) use **fixed point** and process them accordingly
- **continuous** quantities (like the age of the Universe or the size of a carbon molecule) use **floating point**
