LEARNING floating-point

#floating-point

Table of Contents

About

Chapter 1: Getting started with floating-point

Remarks

This section provides an overview of what floating-point is, and why a developer might want to use it.

It should also mention any large subjects within floating-point, and link out to the related topics. Since the Documentation for floating-point is new, you may need to create initial versions of those related topics.

Examples

Overview

What is floating point?

There are two types of numbers:

- fixed point where a certain number of digits are available before and after the radix point.
- floating point where a certain number of digits are available for the mantissa and for the exponent.

An example using decimal digits with three decimal places before the decimal point and two decimal places after the decimal place:

- 0 would be represented as 000.00
- 0.123 would be represented as 000.12
- 0.00123 would be represented as 000.00
- 1 would be represented as 001.00
- 1.123 would be represented as 001.12
- 1.00123 would be represented as 001.00
- 123.456 would be represented as 123.45
- 1234.56 is an error because it would be stored as 234.56 and that is just wrong

An example using decimal digits with five decimal places for the mantissa and one decimal place for the exponent:

- 0 could be represented as .00000 x 10^0
- 0.1 could be represented as .10000 x 10^0
- 0.0000123456 could be represented as .12345 x 10^-4
- 0.00000000123456 could be represented as .12345 x 10^-9
- 0.00000000001 is an error because the exponent isn't large enough to store the number
- 1 could be represented as .10000 x 10^1

- 1.123 could be represented as .11230 x 10^1
- 1.00123 could be represented as .10012 x 10^1
- 123.45678 could be represented as .12345 x 10^2
- 123456789.1 could be represented as .12345 x 10^9
- 1000000000 is an error because the exponent isn't large enough to store the number

So a floating point number can represent numbers with very different magnitudes (0.000000000123456 and 123456789.1) with the same amount of relative accuracy.

Fixed point numbers are useful when a particular number of decimal places are always needed regardless of the magnitude of the number (money for example). Floating point numbers are useful when the magnitude varies and accuracy is still needed. For example: to a road engineer distances are measured in meters and .01 of a meter is insignificant but to a microchip designer the difference between 0.0000001 meters and .000000001 meters is huge - and a physicist might need to use huge numbers and very, very tiny numbers in the same calculation. Accuracy at many different magnitudes is what makes floating point numbers useful.

How it works

Computers don't use decimal - they use binary and that causes problems for floating point because not every decimal number can be represented exactly by a floating point number and that introduces rounding errors into the calculations.

Having done all the examples in decimal it is important to note that because they are binary, instead of storing a floating point number as a sum of decimal fractions:

```
123.875 = 1/10^-2 + 2/10^-1 + 3/10^0 + 8/10^1 + 7/10^2 + 5/10^3
```

computers store floating point numbers as a sum of binary fractions:

```
123.875 = 1/2^-6 + 1/2^-5 + 1/2^-4 + 1/2^-3 + 1/2^-1 + 1/2^0 + 1/2^1 + 1/2^2 + 1/2^3
```

There are many different ways of storing bit patters that represent those fractions but the one most computers use now is based on the IEEE-754 standard. It has rules for storing both decimal and binary representations and for different size data types.

The way normal numbers are stored using the IEEE standard is:

- one bit for the sign - stored in the MSB, 1 means negative and 0 means positive
- some bits for the exponent - the bias is subtracted to get both positive and negative exponents
- some bits for the mantissa - digits after the decimal place with an implicit 1 before the decimal place.

To allow a more gradual underflow, denormalized numbers (when the exponent bits are all zero) are treated specially: the exponent is set to -126 and the implicit leading 1 before the decimal place is NOT added to the mantissa.

32 bit IEEE-754 Floating Point Numbers

For a normal 32 bit IEEE-754 floating point number:

- bit 32 is the sign
- bits 24-31 are the exponent - the bias is 127
- bits 1-23 are the mantissa

So a normal number is calculated as:

```
-1^sign * 2^(exponent-bias) * 1.mantissa
```

If the bit pattern were:

```
0 10000101 11101111100000000000000
```

Then the value is:

```
-1^0 * 2^(133-127) * 1.111011111
-1^0 *     2^6     * (1 + 1/2 + 1/4 + 1/8 + 1/32 + 1/64 + 1/128 + 1/256 + 1/512)
   1 *      64     * 991/512
         123.875
```

There are some special values:

```
0 11111111 11111111111111111111111 = NaN
0 11111111 00000000000000000000000 = +infinity
1 11111111 00000000000000000000000 = -infinity
0 00000000 00000000000000000000000 = +Zero
1 00000000 00000000000000000000000 = -Zero
```

Specifics of the 32 bit IEEE-754 format can be found at:

- http://floating-point-gui.de/formats/fp
- https://en.wikipedia.org/wiki/Single-precision_floating-point_format
- https://en.wikipedia.org/wiki/IEEE_floating_point

Read Getting started with floating-point online: https://riptutorial.com/floating-point/topic/8816/getting-started-with-floating-point

Credits

S. No	Chapters	Contributors
1	Getting started with floating-point	Community, Jerry Jeremiah