CS 101: Computer Programming and Utilization

Jan-Apr 2017

Sharat
(piazza.com/iitb.ac.in/summer2017/cs101iitb/home)

Lecture 8: Numbers (Continued)

About These Slides

- Based on Chapter 3 of the book
  An Introduction to Programming Through C++
  by Abhiram Ranade (Tata McGraw Hill, 2014)

- Original slides by Abhiram Ranade
  –First update by Varsha Apte
  –Second update by Uday Khedker
  –Third update by Sunita Sarawagi

Data Representation

What happens when you say

float x = 23.2

double y = 1.3E27

Model for Today’s Demo

1. We will “open up” the computer program
   - Compile using the “-g” flag
   - Run using the emacs debugger which allows step by step instruction
   - Example char letter = ‘A’;
2. We will use a calculator
   - Some steps will be ‘invisible’
   - Example real numbers
3. In both cases, we will need audience participation

Real Numbers

- The digits in the fraction 0.234 are 2*1/10 + 3*1/100 + 4*1/1000
  – Digits can be recovered by multiplying by 10
- Recall that we have limited number of bits
- So some numbers can never be exactly represented (even) in decimal system e.g. 1/7 is approx 0.142857 142857
- We don't expect 0.33+0.33+0.33 = 1 even though ⅓ +⅓+⅓= 1
- In binary, 0.1+0.2 != 0.3

Fractions In Binary

- Powers on the right side of the point are negative:

8	4	2	1	1/2	1/4	1/8	1/16

- Binary 0.1 = $0 + 1 \times 2^{-1}$ = 0.5 in decimal

- In Binary 0.11 = $0 \times 1 + 1 \times 2^{-1} + 1 \times 2^{-2}$

  = 0.5 + 0.25 = 0.75 in decimal

Converting Decimal Fractions

1. Multiply by 2. The whole number part of the result is the first binary digit to the right of the point.
2. Disregard the whole number part and multiply by 2 once again. The whole number part of this new result is the second binary digit to the right of the point.
3. Continue this process until we get a zero as our decimal part or until we recognize an infinite repeating pattern.

Converting Decimal Fractions

1. Because .625 x 2 = 1.25, the first binary digit to the right of the point is a 1. So far, we have .625 = .1???
2. Because .25 x 2 = 0.50, the second binary digit to the right of the point is a 0. So far, we have .625 = .10?? . . .
3. Because .50 x 2 = 1.00, the third binary digit to the right of the point is a 1. So now we have .625 = .101?? . . .
4. We do not need a Step 4 because we had 0 as the fractional part of our result. Hence the representation of .625 = .101 (Double check the answer)

Converting Decimal Fractions

1. Because .1 x 2 = 0.2, the first binary digit to the right of the point is a 0. So far, we have .1 (decimal) = .0??? . . . (base 2) .
2. Next we disregard the whole number part of the previous result (0 in this case) and multiply by 2 once again. Because .2 x 2 = 0.4, the second binary digit to the right of the point is also a 0. So far, we have .1 (decimal) = .00?? . . . (base 2) .
3. Because .4 x 2 = 0.8, the third binary digit to the right of the point is also a 0. So now we have .1 (decimal) = .000?? . . . (base 2) .
4. Because .8 x 2 = 1.6, the fourth binary digit to the right of the point is a 1. So now we have .1 (decimal) = .0001?? . . . (base 2) .
5. Because .6 x 2 = 1.2, the fifth binary digit to the right of the point is a 1. So now we have .1 (decimal) = .00011?? . . . (base 2) .
6. Let's make an important observation here. Notice that this next step to be performed (multiply .2 x 2) is exactly the same action we had in step 2. We are then bound to repeat steps 2-5.
7. Therefore .1 (decimal) = .00011001100110011 . . . (base 2) .

Large Real Numbers

- Large integers were handled by increasing word size
- But real numbers can be small, or extremely large
  - To an engineer building a highway, it does not matter whether it's 10 meters or 10.0001 meters wide
  - To someone designing a microchip, 0.0001 meters is a huge difference. But she'll never have to deal with a distance larger than 0.1 meters.
  - A physicist needs to use the speed of light (about 300000000) and Newton's gravitational constant (about 0.0000000000667) together in the same calculation.

http://floating-point-gui.de/formats/fp/

Representing Real numbers

- Use an analogue of scientific notation:
  significand * 10^{exponent}, e.g. 6.022 * 10^{22}
- For us the significand (mantissa) and exponent are in binary
  significand * 2^{exponent}
- Single precision: store significand in 24 bits, exponent in 8 bits. Fits in one word!
- Double precision: store significand in 53 bits, exponent in 11 bits. Fits in a double word!
- Actual representation: more complex. "IEEE Floating Point Standard"

Example

- Let us represent the number 3450 = 3.45 x 10^3
- First: Convert to binary:
- 3450 = $2^{11} + 2^{10} + 2^8 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$

11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	1	1	0	1	0

- Thus 3450 in binary = 110101111010
- 3450 in significand-exponent notation: how?
- 1.10101111010 x 2^{1011}
  - 10 in binary is 2 in decimal
  - 1011 in binary is 11 in decimal, we have to move the "binary point" 11 places to the right
  - **we don't care about the last zero once we move the binary point**

Example Continued

- Use 23 bits for magnitude of significand, 1 bit for sign
- Use 7 bits for magnitude of exponent, 1 bit for sign

  0 0001011   01 10101111010000000000000
- Decimal point is assumed after 2nd bit.
- **In fact,** even the "1" before the decimal point is assumed. If you look inside the computer you see **Official Tool**
- 010001010   10101111010000000000000
- The blue part is **11** +127 = 138, The black part is 5742592
- The exponent does not have a sign but a bias. This, and the bit sequence, allows floating-point numbers to be compared and sorted correctly even when interpreting them as integers.

Concluding Remarks

- Key idea 1: Current/charge/voltage values in the computer circuits represent bits (0 or 1).
- Key idea 2: Use numerical codes to represent non numerical entities
  - letters and other symbols: ASCII code
  - In fact, even the program written in “English” gets converted to numbers. So we have operations to perform on the computer and operation codes
- Key idea 3: Radix based system
  - Integers can be represented using sequence of bits. In a fixed number of bits you can represent positive integers in a fixed range.
  - If you dedicate a bit to representing the sign, the range of representable numbers changes.

Concluding Remarks

- Key idea 4:
  - Real numbers are represented approximately.
  - Because we need very large numbers and very small numbers, we cannot have a fixed location for the “decimal point” (or “binary point”). If you want more precision or greater range, you need to use larger number of bits.