

Floating-Point Numbers; C Program; Linux

CS 350, Lab 2, Due Fri Feb 3 (2400 hrs)*

Submit your *.c file to the Lab 2 assignment folder on Blackboard. Include the answers to the written questions and your program output as separate files or as comments in your *.c file. (To submit multiple files, zip them together first.)

* Read the [course syllabus](#) to see how to get an automatic extension to Mon Feb 6.

A. Why?

- Floating-point numbers are one of the basic datatypes supported by modern computers.

B. Outcomes

At the end of this lab you should:

- Be able to translate floating-point numbers to/from binary, decimal and IEEE format.
- Understand some of the precision problems that come up with floating-point numbers.

C. Questions [100 points total]

Part 1 [50 points]

1. [6 = 2*3 points] Do Question 2.3 in the textbook (page 43): (a) Assume that there are about 400 students in your class. If every student is to be assigned a unique bit pattern, what is the minimum number of bits required to do this? (b) How many more students can be admitted to the class without requiring additional bits for each student's unique bit pattern?
2. [6 = 3*2 points] Do Question 2.16 in the textbook (page 44): Write the results of the following additions as both 8-bit binary and decimal numbers. For each part, use standard unsigned binary addition as described in Section 2.5.1. (a) Add the 1's complement representation of 7 to the 1's complement representation of -7. (b) Add the signed magnitude representation of 7 to the signed magnitude representation of -7. (c) Add the 2's complement representation of 7 to the 2's complement representation of -7. (Note: For readability, you can add spaces between groups of bits, as in 0000 0011; the spaces don't change the meaning of the bitstring.)
3. [6 points] Let hex BED0 0000 represent an IEEE 32-bit floating-point number. What is the equivalent decimal number? I.e., read the hex number as a 32-bit string, then convert bitstring that to decimal using the IEEE floating-point representation. (Feel free to leave the answer in fractional form.)
4. [6 points] What is the base 2 representation of $5^{43}/64$?
5. [5 points] What is the IEEE 32-bit floating-point representation of $5^{43}/64$, in binary and hex?

6. [6 = 3*2 points] Let $X = 11.11111\ 11111\ 11111\ 11111\ 111$. [Ignore the spaces — they're just there for readability.] (a) Why is it impossible to represent X **exactly** in 32-bit IEEE floating-point? (b) and (c) What are the two binary numbers closest to X that we *can* represent?
7. [15 = 5*3 points] Let's say we're working with floating-point binary numbers with 5 significant bits. (E.g., we can represent 1.0000, 1.0001, 1.0010, and 1.0011, but not or 1.00000 1.00001.) Let's assume we truncate bits if we can't represent them. E.g., we'd represent 1.00001 by 1.0000, and we say that truncation introduced an error. On the other hand, truncating 1.00000 to 1.0000 does not introduce an error; the result is "exact".
- What is $1.1111 + .11111$? Was there an (truncation) error?
 - What is $.11111 + .11111$? Was there an error?
 - What is $1.1111 + (.11111 + .11111)$? Was there an error?
 - What is $(1.1111 + .11111) + .11111$? Was there an error?
 - Is floating-point addition associative, in general?

Part 2: [50 pts] Write a C program that does the following:

- [2 pts] Prompt the user for a string of 0's and 1's or q (for quit). (Use `printf`.)
- [6 pts] Read a string, stopping when it hits white space. (Represent the string as an array of characters and read using `scanf`.)
- [2 pts] If the string is "q", stop the program. (Compare using `strcmp`.)
- [10 pts] Otherwise, treat the bitstring read in as an unsigned binary integer and translate it into decimal. Print the string, its length, and its decimal value. E.g., for input 101, you print length 3 and value 5.
An easy way to do the translation is to work through the bits from left to right. Starting with a partial result of 0, multiply the result by 2 and add in the current bit. Then move onto the next bit and repeat. E.g., for 0110, we start with the leftmost 0 and a result of 0, so $\text{result} = 0*2+0 = 0$, then $\text{result} = 0*2+1 = 1$, then $\text{result} = 2*1+0 = 2$, then $\text{result} = 2*2+1 = 5$.
- [3 pts] If the string contains anything other than a '0' or '1', complain, display the character, go to step (g).
- [15 pts] Now treat the bitstring as a 2's complement number — take its negative and print out the resulting bitstring. E.g., for input 110, you should print 010.
- Repeat steps (a) – (e) until the user enters q for quit.
- [12 pts] Run your program on (binary strings for) inputs 12, 18, 13, 17, 24, 10.

Extra credit (5 points)

- Allow the string to contain internal blanks. Include the blanks when you echo the input and its complement. E.g., if the input was `11 00`, you'd print the complement as `01 00`.
 - Note you can't use `scanf` to read the input string anymore, since it stops at blanks. You can use `fgets(array, n, stdin)`, where `array` is a character array and `n` is its size. This reads characters from the "file" stream called standard input (`stdin`) into the given array of characters. It stops when it reads `'\0'` or if it reads `n-1` non-`'\0'` characters; in either case, the string-terminating `'\0'` is also written to the array.
-

D. Linux

- For this lab, read sections 1, 2, 3.1–3.3, 3.7–3.8, and 3.10 of the introduction to Linux document by Dr. Matt Beckman. (Thanks, Matt!)