CMPSC 210
Principles of Computer Organization
Fall 2016
John Wenskovitch
`http://cs.allegheny.edu/~jwenskovitch/teaching/CMPSC210`

Lab 3 – Performance, Binary, and C

Due (via Bitbucket) Wednesday, 21 September 2016
30 points

Lab Goals

- Practice another performance problem
- Convert some numbers between decimal and binary
- Write a simple C program

Assignment Details

Our course content for the past week has been a bit scattered: we've covered performance, transistors and low-level computational structures, some of the details of compiling, and wrapped up with converting from binary to decimal and back again. Because of the diverse lecture content, this lab will also include a good deal of diverse content.

Part 1: Performance (10 points)

From the 5th edition of the Patterson and Hennessy textbook, please answer all parts to Exercise 1.7 on page 56. For partial credit, include all of your work rather than just each answer.

Part 2: Number Conversions (10 points)

Please answer each of the following questions about change-of-base conversions (for one point each). Again, please show all of your work for partial credit.

1. Convert the unsigned binary number $(101010)_2$ to decimal.
2. Convert the unsigned binary number $(101101101101)_2$ to decimal.
3. Convert the decimal number $(73)_{10}$ to binary.
4. Convert the decimal number $(1001)_{10}$ to binary.
5. Assuming that the binary number is unsigned, how many binary digits (bits) will it take to store the equivalent of the decimal number $(257)_{10}$? How do you know?

6. Assuming that the binary number is signed using the sign+magnitude method, how many bits will it take to store the equivalent of the decimal number $(257)_{10}$? How do you know?
7. Assuming that the binary number is signed using the twos-complement method, how many bits will it take to store the equivalent of the decimal number $(257)_{10}$? How do you know?
8. Convert the sign+magnitude binary number $(101010)_2$ to decimal.
9. Convert the twos-complement binary number $(101101101101)_2$ to decimal.
10. Convert the decimal number $(-42)_{10}$ to twos-complement binary.

Part 3: Simple C (10 points)

The `PrintIntegerInput.c` code that you looked at in class on Friday introduced a new concept, input and output of variables. Let's take a closer look at the code here:

```
int number;
printf("Enter an integer: ");          // displays the formatted output
scanf("%d", &number);                  // reads and stores the formatted input
printf("You entered: %d", number);     // displays the formatted output
```

First, we declare `number` as an integer. When we want to read in an integer into `number`, we use the `scanf()` function. The `scanf()` function takes two parameters. First is `%d`, which is a placeholder indicating that we want to read in an integer. Second is `&number`, which indicates where we want to store the input value. The `&` symbol references an address, so we are telling C to take the input value and store it at the memory address corresponding to the `number` variable.

We use a similar `%d` placeholder in the `printf()` function. The `printf()` function included two parameters. First was the string that we wanted to print, including the placeholder inside the string: `"You entered:  %d"`. Second was the variable that will replace that placeholder, `number`, this time without the `&` symbol because we are using the variable value rather than the address. There are placeholders for each datatype in C: `%d` is used for **d**ecimal numbers, `%f` is used for **f**loating point numbers, `%s` is used for **s**trings, and more. You can also use multiple placeholders in the same output, such as `printf("%d is less than %d", number1, number5)`.

This exercise is inspired by the example in section 1.3 of Kernighan and Ritchie (page 13). Write a C program that (1) asks the user to input a number between 1 and 50, and (2) prints a table of values of the functions $1/x^3$, $\sqrt{x}$, $\log_3 x$ (base 3 logarithm), and 1.2^x for integers `x` in the range 1 to that input. Your output should have aligned columns, with the decimal points aligned in each column. Each column should have a column header centered above the data.

This requires two bits of formatting knowledge. First, you can insert tabs in C strings in the same way that you insert them in Java, with the `\t` escape character. Second, you can use C placeholders to format your numbers. For example, you can format a floating point number to have 5 digits before the decimal point and 2 digits after with the placeholder `%5.2f`.

This also requires some mathematical and programming knowledge. You can use the `pow()` function to compute a power in C with identical syntax to how it is used in Java: $3^4 =$ `pow(3,4)`.

In order to use the `pow()` function, you need to both include the `math.h` library and use the `-lm` flag when compiling to link the math library.

Similarly, `for` loops work and have syntax in C identical to their operation and syntax in Java. The only exception is that the `gcc` compiler doesn't like you to declare a counter variable inside the loop declaration. In other words, you can't do `"for (int i = 0; ..."`; you have to declare the `i` variable elsewhere. Alternatively, you can use the `-std=c99` or `-std=gnu99` flags when you compile. It's less of a hassle to just declare the variable outside of the loop.

To compute $\log_3 x$, you can use the change of base formula ($\log_3 x = \frac{log(x)}{log(3)}$). To compute $\sqrt{x}$, you can either use the `sqrt()` function or use `pow()` with a .5 exponent.

Here is some example output, which you should attempt to match:

```
jwenskovitch$ gcc table.c -o table -lm
jwenskovitch$ ./table

Enter an integer between 1 and 50:
32

     x      1/x^3      sqrt(x)     log_3(x)      1.2^x
     -      -----      -------     --------      -----
     1     1.00000     1.00000      0.00000     1.20000
     2     0.12500     1.41421      0.63093     1.44000
     3     0.03704     1.73205      1.00000     1.72800
     4     0.01563     2.00000      1.26186     2.07360
   ...       ...         ...          ...         ...
    32     0.00003     5.65685      3.15465   341.82189
```

Submission Details

For this assignment, your submission to your BitBucket repository should include the following:

1. Your responses to the questions in Part 1. [If you have hand-written the solutions to these exercises, please submit them to TA Victor Zheng instead of BitBucket.]
2. Your responses to the questions in Part 2. [If you have hand-written the solutions to these exercises, please submit them to TA Victor Zheng instead of BitBucket.]
3. Your implementation of the `NumberTable.c` code.

Before you turn in this assignment, you also must ensure that the course instructor has read access to your BitBucket repository that is named according to the convention `cs210f2016-<your user name>`.