Lab 02 Specification – Performance Assessment and C Basic Programming
Due (via your git repo) no later than 8 a.m., Friday, 14 September 2018.
50 points

Lab Goals

- Practice another performance problem
- Write a simple C program

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

Learning Assignment

To do well on this assignment, you should also read

- PH chapter 01 - section 1.6
- KR chapter 01 - section 1.5, 1.6
- Class discussion notes, slides, and in-class coding files.

Assignment Details

In this lab assignment, you will solve another performance problem. Next, you will get a chance to work on a simple C program.

Part 1: Performance (15 points)

From the 5th edition of the Patterson and Hennessy textbook, please answer all parts to Exercise 1.7 on page 56. It is expected that you present your solution as detailed as possible. You can consider, the level of details to be similar to in-class activity solution file and examples provided in slides. Look at **submission details** section for more detailed information on how to prepare the solution file for this part.

Part 2: Simple C (20 points)

Let's take a closer look at `PrintIntegerInput.c` code below:

```
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
```

[A copy of `PrintIntegerInput.c` file is provided in the lab repository.]

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:

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

Part 3: Palindrome (15 points)

Write a simple C program called Palindrome.c that performs the following functionality:

1. prompt the user to type in an input string and store the string into a variable firstString.
2. reverse the string *firstString* and store it into a new variable called *secondString*.
3. compare both strings firstString and secondString and validate if they are equal. If they are equal then print "Strings are palindrome", else print "Strings are not palindrome". For example "DAD", "MOM", "CIVIC", "KAYAK", etc.. are palindromes.
4. You are only allowed to use character arrays to implement this part. The usage of any external string libraries, implementation using strcmp, pointers, and other library based functions is not acceptable.
5. You can assume that the length of firstString and secondString is always less than 20. Based on this assuption, the array size can be chosen.

Submission Details

1. You should produce your solution file to Part 1, in the form of PDF file. Other file formats such as JPEG, PNG, DOCX, $\cdots$ submissions would not be accepted. Your solution file should strictly be typed and any images of hand-written text is not acceptable. Although not a requirement, it is highly recommended to use **LATEX** to create your solution file. **LATEX** is an unique text editor that provides an exclusive feature to publish highly professional PDF files based on TEX programming commands. One distinguishing feature of TEX commands are that, it provides a platform to write complex mathematical equations in a very elegant and professional manner. A sample tex file created by Prof. Roberts from University of Adelaide is provided in the repository for your reference. There are other great samples on the web, a google search should lead you to several other examples.
2. Your two programs and the PDF file must be uploaded to your repository no later than 8 a.m., Friday, Sep. 7.
3. Questions about the lab? Bring them to class on Tuesday morning!

Before you turn in this assignment, you also must ensure that the course instructor has read access to your git repository that is named according to the convention `cs200F2018-<your user name>`.

Grading Rubric

1. If you complete Part 1 fully, as per the requirement outlined above, you will receive a total of 15 points. Each sub question is worth 5 points. There will be 0 points awarded for this question, if you do not follow the submission requirement.
2. If you complete Part 2 fully, as per the requirement outlined above, you will receive a total of 20 points.
3. If you complete Part 3 fully, as per the requirement outlined above, you will receive a total of 15 points.
4. Failure to upload the lab assignment code to your git repo, will lead you to receive ”0” points given for the lab submission.
5. There will be a partial credit (30% of the points allocated for the part) given if your code in (Part 2 and Part 3) compiles correctly but not able to produce the expected result.
6. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before due date and make sure to follow the honor code policy described in the syllabus.
7. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.