# CSCE 3600: Systems Programming

## Minor Assignment 7 – Compiler Execution & Construction

### Due: 11:59 PM on Monday, April 30, 2018

1. **(50 Points)** Please read the accompanying PDF document **Linux time Description** prior to continuing with this problem. Then, consider the given CPU bound (i.e., calculation driven as opposed to user input) program compOpt.c that has several loops and typically takes a few seconds to run on our CSE Linux machines. Note that this program requires the –lm flag when compiling to use the math library.
   1. Compile as directed below and run the program using the time command. For example, time ./a.out. Record the “real”, “user”, and “sys” times for each of these program executions.
      1. Compile using gcc only (with the –lm flag).
      2. Compile using gcc –Os (with the –lm flag)**.**
      3. Compile using gcc –O2 (with the –lm flag).
      4. Compile using gcc –O3 (with the –lm flag).

Using your knowledge on compiler optimization (or refer to the man pages for gcc or your lecture notes) to compare and contrast the timing results for each of the different optimizations (i.e., why were some parts – real, user, sys – faster or slower for the various optimizations?). Note that you may want to look at the code and perhaps which specific optimizations are enabled for each one.

* 1. Now, modify the compOpt.c source code to add the following statement just prior to one of the for loops:

#pragma omp parallel for

Then compile using gcc –fopenmp (with the –lm flag) and run using the time command. The –fopenmp flag allows the program to run as many threads as available cores/CPUs. Did it make a difference? What about compiling with this option, plus the optimizing flags in the first part of this problem? Compare and contrast perhaps why the timing results were different (and why). Include the modified compOpt.c file in your submission.

1. **(50 Points)** On Linux, flex is the fast lexical analyzer generator. The file scanner.lex contains the description of tokens to generate a simple and very basic scanner using flex. To generate the scanner, use the following command:

flex scanner.lex

You should notice that flex created the file lex.yy.c in your current directory. Now compile this file into an executable program as follows:

gcc –o scanner lex.yy.c

Note that you must have the zcalc.h header file in the same directory for the compile to be successful. Now you can run the scanner executable and see what it does by typing in text and checking if it is recognized as a token. You may want to view the scanner.lex file to see how tokens are specified. The scanner will output the type of symbol that it recognized for the input that you typed. You may use Ctrl-D to terminate the scanner.

* 1. Run the scanner to enter operators, words, parentheses, etc., but make sure you get results for at least all of the specified tokens and take a screen shot of your output (note that you may do so using the script command with no options, then your screen shot is saved in a file called typescript).
  2. Find at least one common operator or symbol that you use when writing your own programs that is not handled by the scanner and describe what happens. Now modify the scanner.lex file to add support for your operator or symbol and take a screen shot of your output with the now supported operator or symbol. Also include your modified scanner.lex file in your submission.

**REQUIREMENTS:**

* This is an individual programming assignment that must be the sole work of the individual student. Any instance of academic dishonesty will result in a grade of “F” for the course, along with a report filed into the Academic Integrity Database.

**SUBMISSION:**

* You will electronically submit your solution (i.e., this Word document with your answers added), your modified compOpt.c and scanner.lex files, plus the screen shot of the scanner running to the **Minor 7** dropbox in Canvas by the due date.