AMSC/CMSC 662 Homework 3 , Fall 2013
Due: 9:30am Tuesday, October 8.

1. (10 points)

1a. Translate `AllSolve.m` into the C programming language, and run it on the problem that was used to grade Homework 2. The calling sequence should be

```
void AllSolve(double (*f)(double),
                double a, double b, double L, double tol,
                double* Roots, double* PossibleRoots,
                int sizeRoots, int* nroots, int* npossroots)
```

The arguments `a, b, L, tol, Roots, PossibleRoots` were defined in Homework 2. The function `f` has one argument, `x` and returns $f(x)$. The input argument `sizeRoots` is the size of the 1-dimensional array `Roots` and also the size of `PossibleRoots`.

The output argument `nroots` stores the number of roots found, and `npossroots` stores the number of possible roots found. So, for example, if `sizeRoots` is 100 and `nroots` is 5, then `Roots[0], Roots[1], ... , Roots[9]` contain the endpoints of 5 intervals that contain roots of $f(x) = 0$ and the remaining 90 entries of `Roots` are unused.

1b. Compare the execution time with the time needed for the Matlab version.

2. (10 points)

2a. Write a C function

```
void sharAllSolve(double (*f)(double),
                double a, double b, double L, double tol,
                double* nthreads, double* Roots, double* PossibleRoots,
                int sizeRoots, int* nroots, int* npossroots)
```

to find all solutions $x \in [a, b]$ to the equation $f(x) = 0$, as in Problem 1. Use OpenMP to share the work among `nthreads` threads by dividing the interval `[a,b]` into `nthreads` pieces and using your `AllSolve.c` function from Problem 1 on each one.

Run your program for `nthreads` $\in \{1, 2, 4, 8\}$.

2b. Report your solution, the time, and the speed-up relative to your program from Problem 1 on the function that was used to grade Homework 2.

Hints:

I suggest you translate the Matlab function `AllSolve.m` provided as a solution to Homework 2 by Tyler. Until that is available, you can begin Problem 2 by modifying the C code template `Problem2.c` found on the course website.

Timing operations in Matlab: Use `tic` and `toc`.

Timing operations in C:

```
/* Put this statement before main. */

#include <time.h>

/* Put these declaration statements at the beginning,
   when you declare other local variables. */

clock_t starttime, endtime;
double elapsed;

/* This statement substitutes for Matlab's tic. */

starttime = clock();

/* Do the work.  When working with any timer, beware of
   inaccuracies:

   --  If the work takes too long (not an issue in this
   assignment), the counter can roll over to zero (like
   the odometer on an old car) and timing will be inaccurate.

   --  If the work takes too little time, the "random" noise
   in the count will be a large percentage of the measurement.

   If necessary, do the work  S  times between the two calls
   to clock, where  S  is chosen to make the difference in
   starttime and endtime about CLOCKS_PER_SEC, and divide
   elapsed  by S.
*/

/* These two statements substitute for Matlab's toc. */
endtime = clock();
elapsed = ((double) (endtime - starttime)) / CLOCKS_PER_SEC;
```

Reference: `http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_19.html#SEC310`

Submission instructions: Send Tyler email, subject `Hmwk 3`.

Your answers to 1b and 2b should be in the body of the email message.

Include three plain-text attachments:

- `AllSolve.c` contains only the function `AllSolve` from Problem 1.
- `sharAllSolve.c` contains only the function `sharAllSolve` from Problem 2.
- `Problem2.c` contains the complete set of functions (`main`, `sharAllSolve`, etc.) used for Problem 2.

Tyler will grade your programs for documentation, clarity, efficiency, and correctness on test problems that he has chosen.

Your programs should not display any information unless there is an error. (Otherwise the timing will be unnecessarily long.)