CMPT 470/816 HOME EXERCISE 4
PROFILING AND PERFORMANCE IMPACT OF FUNCTIONAL ABSTRACTION

Part A. Relative timing

From moodle, get the 3 sample pieces of code for the last two exercises

1. The original ("ugly") code for Conway's game of life
2. The refactored code for Conway's game of life absent flexibility concerning the rule to use for updating a cell.
3. The refactored code for Conway's game of life with flexibility concerning the rule to use for updating a cell.

We will now explore the relative performance of these code.

Please compile and measure timing on each of the above

Compilation command: gcc -std=c99 filename.c

Please test the timing with the linux "time" command. Please run each of these for 50000 iterations (if necessary, you may need to modify the code to do so, making sure that the output reporting progress only occurs every 1000 iterations).

How much time does each of the codebases above take to execute? Please comment on this, interpreting the results. Please construct a table of results, where each row has entries in successive columns giving a description of the source file, a label "Part A", "Part C O2", "Part C O3", and then the "user" time as reported via the unix "time" command. For this subpart, you are only required to fill in "Part A".

Part B. Profiling

Please re-compile the above using profiling option, using the "-pg" option of GCC. That is, you should compile as follows:

Compilation command: gcc -std=c99 -pg filename.c

After compiling each source file, please run it for 50000 iterations. Please then us the "gprof" command (giving it the executable name) to output a profiling report for the code. For example, if your executable is named lifeRefactored, the command might be

gprof lifeRefactored > gprof.lifeRefactored.txt

Please examine the profiling results text file output by gprof, paying attention to interpret the results according to the metadata included in the text file, and (where required or of interest) by the man page of gprof.

Part C. Optimization

We will now explore how the performance of these codebases change with aggressive optimization.

Please compile and measure timing on each of the above, in the same way that was performed in Part A (please note that profiling information is *not* being requested here)

Compilation command: gcc -std=c99 -O2 filename.c

Please test the timing with the linux "time" command. Please run each of these for 50000 iterations (if necessary, you may need to modify the code to do so, making sure that the output reporting progress only occurs every 1000 iterations). Please populate the entries for "Part C O2" using this information.

Now perform yet more aggressive optimization and report the results:

Compilation command: gcc -std=c99 -O3 filename.c

Due in class January 23, 2017

How much time does each of the codebases above take to execute, as judged by the "user" report from the unix "time" command? Please comment on this, interpreting the results. Please use this information to populate the entries for "Part C O3" above.

What are your high-level interpretations of the above?