All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates, but you may not look at their code. If in doubt, ask the instructor.

Acknowledge all sources you found useful.

Your code should compute the correct results.

Partial credit is available, so attempt all exercises.

Submit your code and answers in a single archive file (ZIP/TGZ/TBZ2, etc.) that contains your name in the filename (e.g. `JRandom_A1.zip`). Your answers should be a PDF file in the archive.

In this assignment, you will parallelize the computation of the histogram of RGB images.

Images are supplied to you as portable pixmap (PPM) files. The value of each pixel varies from 0 to 255. The supplied files use the binary variant of PPM, but you can obtain an ASCII version of the file using the `pnmtoplainpnm` command if you want to examine the contents.

Do not change the program interface/argument order.

Your programs will be tested on `node2x14a`, `node2x18a` and `node-ibm-822.csug`. Modify the makefile so that `make` *`binaryname`* makes the binary.

When reporting times below for each question, report times for 1, 2, 4 and 8 threads along with which machine you used to obtain the time. As in Assignment 2, repeat each experiment and report averages and standard deviation.

Check for correctness against the serial version using the `diff` command.

Exercise 1

Parallelize the `histogram` function, with a private histogram per thread. Merge the private thread histograms in the main thread of the program.

Call the binary for this problem `histo_private`.

Report time for execution for all inputs in a table.

Exercise 2

Parallelize the `histogram` function with a global, shared histogram, using lock-free atomic additions to update the values of buckets. You can use C++11 atomics, C11 atomics or GCC builtins. Note, though, that your binary must build on the CSUG machines.

Call the binary for this problem `histo_lockfree`.

Report time for execution for all inputs in a table.

Exercise 3

Parallelize the `histogram` function with a global, shared histogram, using a lock for each bucket per channel (i.e. one lock each for red, blue and green).

You may *not* use locks provided by C++, pthreads, POSIX, etc., but must implement your own.

Implement at least two different locks from MLS, Chapter 4. Make sure your locks work correctly on all machines. Report which locks you implemented.

Call the binaries for this problem `histo_lock1` and `histo_lock2`.

Report time for execution for both lock variants in a table.

END.