The COMP 412 Lab 1 Autograder
Fall 2023

This code is a modification of the original that was written by Lung Li when he was a graduate student at Rice.

This distribution contains the autograder and autotimer for Lab 1 in COMP 412. The autograder includes a subset of the files on which your submission will be graded. The autotimer includes the full set of files that will be used in grading. (The timing files are also available in the directory ~comp412/studentsI/ILOC/Scalability on CLEAR.)

You can use the autograder and autotimer to determine whether your tar-file submission will work with the autograder and autotimer. To build your copy of this infrastructure:

1. In your file system on CLEAR, create a new directory where you can unpack the auto-grader and test your submission. In a shell,, run the commands:

   ```
   mkdir <name>
   cd  <name>
   tar -xvf ~comp412/students/lab1/autograder/L1AG.tar
   ```

   where `<name>` is the new directory's name.

   The README file will contain directions on how to use the autograder to test your submission. Much of what is in this file is also found in the README file. You might do well to read both, in case one of them omits something.

2. Go to the subdirectory "auto_grade" and update two variables in the file auto_grade.py. The string "base_name" should contain the full path to the directory where you unpacked the tar file — that is, the directory that contains the file "Grader". The string base_name should end in a slash ('/').

   Next, check the date in the initialization of "normal_deadline". It should be set to the on-time deadline for code in this project.

3. Go to the subdirectory "auto_time" and update the string "base_name" in file auto_time.py so that it specifies the directory where you unpacked the tar file in step 1. The base_name should include the full path name and should end in a slash ("/").

4. Put tar files to be graded in the "TarFileGoesHere" subdirectory

Running the Autograder

Invoke "./Grader" in the base_path directory. It will attempt to grade each of the tar files that it finds. The results are in a timestamped log file. The log file's name begins with "Grader-" and ends with ".log". The log should be self-explanatory.

Be sure to check that the grader has your name and netid correct. It reads these you're your README file. (Note that the file must be named “README” not “README.md” or “README.txt” or “readme” or …)

The auto-grader penalizes a lab for an error in a correct line and for failure to find an error in an incorrect line. It considers multiple messages for a single line as a single occurrence.

For a working lab, all of the error counts on the correct files should be zeroes. Similarly, the pairs for the error files should be a pair where the first number is zero. See the grading rubric in the lab handout.

The autograder also runs a quick scalability test. It runs the submission on files with 8,000, 16,000, 32,000, 64,000, and 128,000 lines of input. If the lab is scaling linearly, you would expect the time for a given file to be no more than twice as large as the next smaller file.

If the times are all the same (and quite small) that indicates that the submission, itself, is likely not working.

Running the Autotimer

Once your submission passes the autograder (and, only after it passes the autograder), you can run the autotimer to obtain a more accurate picture of scalability and efficiency. You invoke the autotimer with the command “./Timer” in the base_path directory. It creates a log file whose name begins with the string “Timer-“.

The autotimer runs your submission on each of 8 input files, ranging in size from 1,000 lines to 128,000 lines of ILOC. It does not look at correctness. It times your submission on each file five times and keeps the minimum time. It looks at the results and assesses (roughly) if the executable ran in linear time, quadratic time, or something larger than quadratic time. Its ability to recognize those categories is reasonable. In some cases, I have to plot the data and look at the curve to double-check its judgement. (In particular, on a busy system, there may be a significant amount of noise in the timings of the small blocks.)

The autotimer cuts off any run that takes more than 60 seconds. If you see a 60 second run, it is likely that the submission did not complete the run (and that the timing results are worse than those represented in the log file). In any event, your submission should not take more than 60 seconds to run on any of the test files.

The log file scores the submission on both scalability and efficiency. It applies the language specific cutoffs from the Lab 1 handout in its efficiency score. If the line that contains the efficiency score does not have the correct implementation language, please contact the instructors via Piazza.

If a tar files is ungradeable, the log file may provide insight into why the test failed. Common problems include:

1. The tar file is incorrect. It needs to unpack into the current directory. The auto-grader creates a test directory, moves into that directory, and unpacks the tar file. It expects your code, README, Makefile (if any), and script (if any) to be in that directory. If you created the tar file from one level higher in your file system tree, the auto-grader will not find the files that it needs.
2. Your README file does not contain the required NAME and NETID fields. The auto-grader uses these fields to assign your points to your NetId. These show up early in the log file. Check that they are correct.

   Note that this file must be a simple ASCII text file named "README". It has no suffix (e.g., not "README.md"). You can rename a file using the "mv" command or the "cp" command in Linux.
3. Your submission does not run straight out of the box. The auto-grader is going to look for a Makefile. If it finds one, it will execute a "make clean" followed by a "make build". Those should produce an executable named ./412fe that the auto-grader can run.
4. Your Makefile or your script contain absolute pathnames. Remember, the auto-grader will build and execute your lab in some distant and unknown part of the file system. Check manually for absolute pathnames. (If you have one, and you test the code somewhere else in your file system, it will almost certainly work because you have access to the original file in its original location. That won't happen when **comp412** tries to run the code.)