The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

COMPUTER SYSTEM PROGRAMMING
ECE454 – FALL 2023
ASSIGNMENT-1

Dues are defined at the end of this guideline.

Objectives

- Reviewing/Working with the Compilation Flags (-g, -O0, -O1, -O2, -O3, ...)
- Reviewing/Working with the Profiling Tools (time, gprof, gcov)
- Reviewing/Working with the Inspection Tool (objdump)

1.1 Build and Test

In this section, you will review the CMake, an open-source cross-platform packaging system to control the compilation process of a source code. It will generate Makefile automatically based on the computer configuration. Use the instructions below to build the required Makefile for Lab-1 (you can find the typical source codes and also relevant Makefiles in the "*Lab1-code.zip*"):

```
> cd lab1                    // Navigate to the lab1 assignment directory you intend to work with
> mkdir bin && cd bin        // Make a new directory called bin, then navigate inside
> cmake ../                  // Use cmake to generate Makefile automatically
```

After the Makefile is automatically generated, simply run the Makefile and an executable version of your source code should appear within the bin folder.

Q1- List the command to run that executable version of your source code and display the result. [1 mark]

1.2 Measuring Compilation Time

In this section, you will use /usr/bin/time to measure the compilation time and performance.

Note- the number in the output that ends in "user" is the runtime in seconds in the "user-mode". This is the time that you should use in this report.

Note- since you are measuring performance for a real system, the measurement results are a bit different at each execution time due to system variability. It is recommended to measure on an unloaded machine.

For every timing measurement, runs at least 5 times and then find the average among the results, but only report the final average with proper description.

-Measure compilation times using these compilation flags: 1) gprof, 2) gcov, 3) -g, 4) -O2, 5) -O3, 6) –Os
Be sure to regenerate the make file and run "make clean" in between each build to ensure that all files are rebuilt properly.
-To build the gprof version, use the flags: -g -pg -no-pie
-To build the gcov version, use the flags: -g -fprofile-arcs -ftest-coverage
Note- In this section, which compilation flags should be used for CMake?
Note- Find out the proper compilation flags in the CMake or stackoverflow documents.

Q2- Report the 6 compilation time measurements using the slowest method of compilation as a baseline. Report the speedup for each of the other five measurements (compared to the baseline). For example if gcov is the slowest, and -g is twice as fast as gcov, then the speedup for -g relative to gcov is 2.0. [1 mark]
Q3- Which compilation time is the slowest and why (describe briefly)? [1 mark]
Q4- Which compilation time is the fastest and why (describe briefly)? [1mark]
Q5- Which of gprof and gcov is faster and why (describe briefly)? [1 mark]

1.3 Measuring Program Size

- In this section, you will use "ls -l" to measure the size of all six versions of binary files generated in the previous section.

Q6- Report the six size measurements using the smallest method of compilation as a baseline. Report the relative size increase for each of the six measurements. Eg., if -g is the smallest, and gprof is twice the size of -g, then the relative size increase for gprof relative to -g is 2.0. [1 mark]
Q7- Which size is the smallest and why (describe briefly)? [1 mark]
Q8- Which size is the largest and why (describe briefly)? [1 mark]
Q9- Which of gprof and gcov is smaller and why (describe briefly)? [1 mark]

1.4 Measuring Performance

- In this section, you will measure the run-time of all six versions compiled in the previous section.

Q10- Report the six measurements using the slowest measurement as a baseline, also report the speedup for each version. [1 mark]
Q11- Which version is the slowest and why (describe briefly)? [1 mark]
Q12- Which version is the fastest and why (describe briefly)? [1 mark]
Q13- Which of grof and gcov is faster and why (describe briefly)? [1 mark]

1.5 Profiling with gprof

- In this section, you will compile the gprofsupport using flags "-g -pg", "-O2 -pg"and "-O3 -pg" respectively. Then run each of these versions to collect the gprof results (no need to measure the time for any version).

Q14- For each version, list the top 3 functions: the function name and percentage of execution time. [1 mark]
Q15- For the "number-one" function for "-O3 pg" (the one with the greatest percentage execution time), how does its percentage execution time compare with the percentage execution time for the same function in the "-g pg" version? How is this possible? What transformation did the compiler do and to which functions? [1 mark]

1.6 Inspect Assembly

- In this section, you will use objdump to list the assembly for the -g and -O3 versions. Run it as: "objdump -d OBJ/main.o", to list the assembly instructions for the file main.c.

Q16- Count the instructions for the "number-one" function identified in the previous question and report the counts. Also report about the reduction rate in the number of instructions for –O3 version. For example if -O3 version has half as many instructions as the -g version, the reduction rate is 2.0 times. [1 mark]

1.7 Profiling with gcov

- In this section, you will use gcov to get the per-line execution counts of the "number-one" function from the -O3 version (but use the "-g" version to gather the gcovprofile). After running the gcovversion, execute the gcov program to generate a profile of the appropriate file (eg., run "gcov -o OBJ -b main.c" to profile the file main.c). Running gcov will create main.c.gcov(for main.c).

Note- If you run the gcov program multiple times it will add to the counts in main.c.gcov; you have to remove the .gcda and .gcno files in OBJ/ to start counting from zero.

Q17- Based only on the gcov results, list the functions in the order that you would focus on optimizing them for the provided Lab1 inputs and why. Identify each location by its line number in the original source file. [1 mark]

1.8 Get familiar with GCC man page

- In this section, use the man gcc shell command or view gcc manual page online at: https://linux.die.net/man/

Q18 (optional)- Name the shortest GCC compiler flag (i.e, -xx) to enable a compiler optimization that requires memory alignment. How many bytes does the data need to be aligned? [1 bonus Mark]

Deliverables & Submission for Assignment-1 [Total 20 Marks = weight 5%]

-The teams should submit their final report through the submission folder on Quercus by:

Final Report due on Thursday Sep28, at 8:59am	[3 Marks]

-In addition to the Final Report, A1will be evaluated by the TAs through Q/A in the following Labs:

Thursday Sep21 (partial evaluation, only Q1)	[1 Mark]
Thursday Sep28 (final work evaluation)	[16 Marks]

-The teams should save all their answers in a final report (plain text format) under this naming rule: "PRAxx_Tyy_Az.txt" where "xx" is the Lab section (01 for 12-3pm, and 02 for 9am-12pm), "yy is the Team number and "z" is the Assignment number, e.g. "PRA01_T06_A1.txt" is the name for submission from Team06 for A1 working in the Lab on Thursday 12-3pm.
-Make sure to add the names and student numbers of your Team members, and the Lab section your team attends at the top part of your final report. Only ONE final submission is required from each team.

Good Luck.

Useful Resources:
GCC Man Page:
https://linux.die.net/man/1/gcc
https://man7.org/linux/man-pages/man1/gcc.1.html

Gprof:
https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html
https://www.tutorialspoint.com/unix_commands/gprof.htm

Gcov:
https://gcc.gnu.org/onlinedocs/gcc-11.1.0/gcc/Invoking-Gcov.html
https://github.com/gcc-mirror/gcc/blob/releases/gcc-11.1.0/gcc/gcov.c