Handing In Assignments

CS 358, Spring 2020

Instructor:	Sam McCauley	**Time:**	TR 11:20-12:35
Email:	sam@cs.williams.edu	**Place:**	TCL 206

This document is to help with understanding how assignments will be submitted and evaluated in this course. This document may change over the course of the year as problems with the submission system come up and are addressed.

All assignments will be submitted using git. Please contact me if you have questions about git or if you have not used it before.

Since I will be both pushing new assignments to you over git, as well as receiving your submissions, good repository practice is extremely important. **Please always run git pull before pushing.** This is generally a good habit, and will avoid a large number of potential headaches. If you believe there are problems with your repository, or if you run into a conflict you don't know how to solve, please contact me.

TL;DR Version of rules

This file is a somewhat detailed description of what's going on, so this is a summary of the important rules for quick reference.

- Always pull before pushing to the repo.
- Only edit the makefile to change compilers or compiler options.
- Only work in the directory for the current assignment (or past assignments).
- Don't edit or create any file called `errors.txt`.

Repository

Each student in the course will have a single repository for assignment submissions. (Unlike some CS courses at Williams, we will not have a new repository for each assignment.)

The repository has 12 directories, labelled `Assignment1` ... `Assignment12`. Each directory will be used for the corresponding assignment. Please do not do anything in directories for assignments that have not yet been handed out. (This instruction is to prevent conflicts.)

Each assignment directory contains a file called `NAME.txt`. The first line of this file will be the "name" displayed on the leaderboard on the website. If this is left blank, the name of your repo will be used instead. What you put in this line will be placed verbatim in an html file, so (for now) try to stick to plain text (without reserved characters like &) or html-friendly commands. The ability to name yourself is intended to provide the option of freedom and anonymity–please use your best judgment and stick to professional names. Bear in mind that the course website is public.

When an assignment is released, I will push to your assignment folder the files necessary to complete the assignment. This will generally include a `test.c` file which contains a simple `main()` method to take in file input and call a function that (should) complete the assignment task. It will also contain a header file for the assignment, and a source file with a function stub. There will also generally be two data files: one for testing, and one for timing. The testing data file is intended to make sure your program works on small, human-readable instances and in edge cases; the timing data file is intended to allow you to test how quickly your program runs.

How your code will be timed

(Note: this section discusses timing for simplicity; there may be some assignments measured by a different objective. The same ideas will apply.)

When referring to timing, we will generally be using CPU time in this course. This means that other processes running on the system will usually only have a minor effect on your score. However, this means that parallelism and multithreading will not improve performance.

The timing of your code will use your makefile, and your source code files—but will not use your test.c file. The script will copy over files in your Assignment directory so they can be tested, but will not copy any files in subdirectories. Therefore, please keep your source code files on the top level of the appropriate Assignment directory. (I believe that the techniques in this class are simple enough that multiple directories are not necessary, but please let me know if this is an inconvenience, or if your (say) IDE has issues with this. It is likely easy for me to accommodate corner cases.)

As in the `test.c` file that you have access to, your program will be tested first for correctness on small edge-case inputs without any timer. Then, your program will be timed in a larger input. These tests must *all* run in 60 seconds (of *wall clock time*) for your submission to be considered correct. (The test script will first attempt to run the entire program with a 60 second timeout, and will run it again without the timeout only if your program completes.)

These tests will use two additional data files, which are not public to students. One is intended to test correctness and edge cases; the other is intended to test for time.

Your score will only be changed if you achieve a better time—the leaderboard displays a "high water mark."

Schedule for updating the leaderboard

I have scheduled regular tests on the lab computers. Your code will be tested at least twice a day: at 9AM and 9PM. On days with a deadline, there will be an additional test at 7PM, and a further test after the deadline to obtain the final scores. I may run additional tests throughout the day, especially if there are problems with a previous test.

These tests will always test the most current version in your repository, so remember to run `git push` (always after first running `git pull`) regularly to make sure your changes are reflected. Each test will be timestamped on the website. Please tell me if there appears to be a problem.

Rules for modifying makefiles (and comments on hacking this setup)

This testing setup is somewhat obviously not secure, as the ability to modify makefiles means that you have the ability to run arbitrary code on the testing machine. Please only modify the makefiles to include new source code files, or to change compiler options like flags or what compiler you're using.

Hacking this setup is currently fairly trivial, to the point that it's probably not worth doing. For example, you could fairly easily write a program that mimics the testing file, printing that you got the correct solution in zero seconds; you could then write a makefile that compiles this source file instead of test.c. Or, you could have the makefile run a simple linux script to find the private testing file and email its contents to you, allowing you to create a function that outputs the correct function immediately. I believe it's also likely that there is a security leak in the information given in `errors.txt`. Submissions using these techniques would (of course) would not count for any credit.

Learning about errors

Any errors that occur while running `make` will be pushed back to you in a file called `errors.txt`.

If you appear to be getting unexpected results in the actual program execution, come talk to me. I do keep track of the output of how your actual programs run; it is just not pushed back to you.

Hardware

All testing will be done on the computers in TCL 312. There are two types of computers in that lab; the two types have subtle differences in their hardware. (It is very unlikely that these differences will be relevant at any point in this course, however. I believe your results on all TCL 312 machines will be consistent.) Your code will be run on machines with the newer hardware; the computers with this setup are named: siri, rathi, reina, amerifax, brava, barzona, zebu, kuri, devon.

For the purposes of correctness and granular optimization, your computers at home will likely give accurate results. However, for very fine-grained optimization, you may want to do your testing on a lab computer. This will allow you to see how well your code runs (and get more accurate feedback) without waiting for a new test script to run.

The lab computers can be accessed remotely using `ssh`. The computers listed above can be accessed using their name; for example, to log into devon, I would run `ssh sam@devon.cs.williams.edu`. (My CS username is `sam`; replace with your own.)

Be aware: these computers can only be accessed **from on campus.** If you are not on campus, you should use a VPN (instructions here: https://oit.williams.edu/help-guides/wifi-and-wired-connections/vpn/), or a proxy (instructions here: https://library.williams.edu/computing/proxy/). My experience with VPNs and proxies is that they can be finicky; test them beforehand if you foresee that you will need them.

External Sources and Libraries

You are allowed to use external sources for your code, including each other and resources like stackoverflow. However, you must cite any source you use you use. Furthermore, you must understand and be able to explain any code you use. The Williams Honor Code and Williams Computer Science Honor Code apply to these citations.

You may use external libraries in your coding assignments. This is encouraged for small subroutines. However, you may not use libraries that solve the assignment in its entirety. For example, if the assignment asks you to write code to multiply two matrices, you may not use a matrix multiplication library. You may, however, use a library that helps improve cache performance. If you are in a situation where it is not clear that library use is permitted, please ask about the specific situation.

Assignment Questions

Each assignment will come with questions to test your understanding of the material (to prepare for midterms and the final). It is expected that this work will be done individually unless the assignment specifies that it may be done in groups.

The questions will be given to you in the form of a Latex (`.tex`) document. Please answer your questions within the document and push it to the repo to submit it. If you are uncomfortable with Latex, it is likely sufficient to simply answer the questions via text in the space provided. If you have questions please contact me.