Handing in Homeworks and Assignments

Applied Algorithms, Fall 2024

This document is to help with understanding how homeworks and 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.

Since I will be both pushing new homeworks and 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 homework/assignment.
- Don't edit the `test.c` file, and don't edit or create any file in the `feedback/` directory.

Repository

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

The repository has 6 homework directories, labelled `Homework1` ... `Homework6`. Each directory will be used for the corresponding homework. Please do not do anything in directories for homeworks that have not yet been handed out. (This instruction is to prevent conflicts.) Within each of these directories will be a `feedback/` directory, which is where the automated testing software will store the results of your submissions.

When a homework is released, I will push the files necessary to complete the homework to your repo, in the corresponding directory. This will generally include a `test.c` file which contains

a simple `main()` method to take in file input and call a function that will (after you are done) complete the homework task. It will also contain a header file for the homework, 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 on larger inputs.

The repository also has three directories for assignments, `Assignment1 ... Assignment3`. These directories work largely the same as the homework directories. But, there is no automatic testing for the assignments, so there is no `feedback/` directory.

How your code will be timed

(Note: this section discusses timing for simplicity; there may be some homeworks 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 Homework 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 Homework 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 possible 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. While we'll try to avoid this, it's conceivable that some of these testing times will need to be adjusted due to crashes or maintenance.

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 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. Submissions using these techniques would (of course) would not count for any credit.

Learning about errors

A summary of all runs of your program will be stored in a file called `feedback.html`. Double-clicking on this file in a file browser will generally open it in a web browser, allowing you to view this summary.

Specific runs will be stored in text files; these files will be named according to the date and time of the test. They can be opened directly, or accessed via links in `feedback.html`.

To keep the testing system somewhat secure, you may not receive feedback for your program's errors in some cases. If you are in that situation, try to replicate the issue on your local machine

if possible. I am also happy to help with debugging. (The full error log is stored, it's just not accessible to students, so in office hours you'll be able to get full information on what's going wrong.)

Hardware

All testing will be done on the computers in TCL 312. All computers in the lab are identical. Furthermore, all computers in TCL 301 have the same setup as well, and can also be used.

For the purposes of correctness and granular optimization, your computers at home will likely give accurate results. However, for very fine-grained optimization, you likely 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 following webpage has instructions on how to access the machines: https://csdoc.cs.williams.edu/wiki/index.php/Names_of_CS_linux_computers.

Be aware: these computers can only be accessed directly **from on campus** (see the linked webpage for more details). If you are off-campus, I would recommend using `ssh` to access `lohani`, and then using `ssh` from within `lohani` to access the lab computers.

Note that VSCode (and some other editors) has excellent tools for remotely editing files and running code. You may find that with some setup, by using VSCode you can avoid using command-line `ssh` entirely.

External Sources and Libraries

You are allowed to use external sources for your code, including other students, generative AI, and online resources like stackoverflow. However, you must cite any source that 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.

Homework/Assignment Questions

Each homework and assignment will come with questions to test your understanding of the material. 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 like using Overleaf, you can upload the `tex` file, edit it, and then download it when you are done (go to Menu → Source); let me know if you have any questions. 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.

Tips

Some things to bear in mind while doing assignments:

- Premature optimization is the root of all evil. Never try to optimize your code until you have a working version!
- Don't forget to work on the assignment questions; they are a major part of the course (and of the points for each assignment).
- Continuously reevaluate if the time you're spending on an idea is well-spent (don't get stuck in the weeds!)
  - External libraries (especially for crucial subroutines like sorting) have the potential to speed up your code, and seem like they may work without much effort. But, they can be very finicky and difficult to get working.
  - Using a more involved algorithm may save on running time, but even moderate increases in complexity can result in many new bugs.
  - On the other hand, oftentimes you can get a very significant performance improvement by relatively simple code refactoring.
- Always be ready to ask for help!