CS 252: Computer Organization

Pre-Project

(see class website for due date)

Some handy YouTube videos:

- Intro to `ssh` and the Unix Command Line
  `https://youtu.be/5jqmmeyVnyQ`
- What is Version Control?
  `https://youtu.be/muY8jgnahHk`
- How to Accept an Assignment in GitHub Classroom
  `https://youtu.be/nLmdvqftvI0`
- Using Git on the Command Line
  `https://youtu.be/5j3QAHlOREk`
- Using Git to Synchronize Two Machines
  `https://youtu.be/HtI-7ejrDYc`
- C for Java Programmers
  `https://youtu.be/m5R0h1hr0w4`
  How to Compile a C Program
  `https://youtu.be/oPpYwafaxtE`
- Java for C Programmers
  `https://youtu.be/otDBNxBz4F0`

1 Purpose

This project exists simply to make sure that everybody figures out how to use the tools we'll be using for projects this semester. You'll write a tiny bit of code, but mostly what you'll be doing is setting up user IDs, downloading code that I've provided, and running it to test your environment.

2 GitHub Classroom

This semester, we'll be using GitHub Classroom to turn in our projects (both simulation and assembly). For each project, I will create a repository, which contains the base code (I'll release the same files on the web). You will "fork" that repository (that is, ask GitHub to make a private copy of it for you), and you will turn in your code by uploading files to that repository.

For the Pre-Project, you will create a GitHub user ID (if you don't already have one), fork the GitHub version of this project, do a few simple tests (to make sure you know how to compile and run code), and commit your changes to your fork of the repo.

2.1 GitHub User ID

If you already have a GitHub account, skip this step. (And yes, it's OK if your GitHub account doesn't use your University email address.)

If you don't have an account yet, go to `https://github.com` and create one. You should choose the **free** option - there's no need (for this class) to pay for the more expensive versions. (Those are mostly used for developers at software companies, who want to keep their code private.)

2.2 Fork the Repo

I've provided a link, on the class website, which you can use to fork the repo. Follow the link and click on "Accept this assignment;" GitHub will create a new repo, under your account, which contains a copy of my repo.

It should take you right there once it has been created, however, if you forget where it is, you can always go back to `https://github.com/` and you will see a list of your repositories on the left column.

2.3 Download the Code

Next, you need to access the files. There are several options. The simplest one - (but the one which will make things harder long-term) is to just download a `.zip` file, directly from the GitHub website.

A **much better long-term strategy** is to use `git` - the tool around which the entire GitHub website is based. See the video links at the top of this spec.

3 UserID Associator

GitHub Classroom doesn't (yet) yave any good way to directly integrate GitHub usernames with our university email addresses. So I wrote one of my own. Please go to
`https://userid-associator-dot-lecturer-russ.appspot.com/`

This website will first ask you to log in using your NetID. Once you've logged in, my website will ask the University what your NetID is; I don't try to perform any other action.

When you complete the NetID login process, it will then ask you to do the same for GitHub. Again, all this website will ask for is your username (not your password!). This is because I need to be able to link student repositories to the proper NetIDs, for grading.

When you complete this process, my website will record the two usernames, but will not save any other information about you. You only have to do this once; my website will remember this information for the entire semester.

4 Write a Simple Java Program

A few students may have experience with C, but not with Java. That's fine, we won't be expecting anything very hard! Watch the videos linked at the top of this document; if you need more help, come to Office Hours. Or, you could just follow a simple Java tutorial from the web.

Some of the Simulation projects will require that you write and test some Java code. We expect that all students who are taking 252 have done some Java programming before, so for this step, you must write a simple program in Java. (Make sure to test it!) Your program must:

- Print out a welcome message to the user
- Read a single string (one line, or one word) from the keyboard
- Print out something which includes what was typed

Commit **exactly one** `.java` file to your repo (name it whatever you'd like). We'll be building and running it automatically, so please make it simple - and make sure that you've tested it!

5 Compile Your Choice of C Programs

In this class, we don't expect that you have written C before. We'll be using C for a couple of the Simulation projects, but **don't worry** - we'll only be using a very tiny subset of the language! (And I'll show you everything you need to know.)

In the repo, I've posted several fun programs from the International Obfuscated C Code Contest. These programs are quite silly - they are **intentionally designed** to be as hard to read as possible. So you are free to look inside them, but don't expect to be able to understand them - even I can't (unless I took a **lot** of time)!

You will need to compile one of these programs and run it. If you have a C compiler on your own machine, that's great - you are welcome to run it there. However, I expect that most of you will be using Lectura to compile and run your C programs - see the video links at the top of the spec.

Once you have compiled and run the program, save the output from the program, and commit it to your repo. You should name the file `<program>.txt` (replace `<program>` with the name of the program you chose). You only have to do this for one of the programs - although you're allowed to do it for more than one if you enjoy it.

6 Run the MARS Simluator

One of the files in the repo is `Mars4.5la.jar`, which is the MARS simulator, which we'll be using for assembly projects. It's a Java application; you can

probably run it simply by double-clicking on it. Run it, and take a screenshot of the window that pops up; commit that screenshot to your repo. You can name the file anything you like, but it must be one of the following file types: `PNG, GIF, JPEG, BMP, PDF`. (For the sake of the TAs' sanity, we will not accept other file types.)

7 Run the Grading Script

Log on to Lectura (if you haven't already) and check out the repo there (see my Git video for instructions how to do this). Run the grading script `grade_preProject`. For the rest of the projects, you will be using grading scripts like this to check to see if your code works correctly. Of course, in this case, we don't have code to auto-grade; instead, I've provided a trivial little assembly program (`asm_preProj.s`) and a testcase for it (`test_01.s, test_01.out`). If your environment is working correctly, the grading script will report that you passed the testcase. Save the output from the grading script to a text file, and commit it to your repo. Again, name it whatever you want (so long as it's obvious what it is).

8 Turning In Your Code

As you commit changes to your copy of the repository, `git` saves them to a little database, stored on your computer[1]; it keeps track of what files changed, and when. While it doesn't track every single change you made, it **does** keep track of every **commit** you performed, and it's possible to go back in time to see old files. Very cool!

But if you only make changes to your local repository, then it's not on the GitHub website. In order to post changes there, you must "push" changes to the "origin" (which, in this case, means GitHub itself). Make sure that you do this after you've committed changes - if you don't push your commits, I will never see what you did.

However, once you've pushed your commits to GitHub, you're done. Good job!

[1] When you commit on Lectura, it's the same thing - `git` simply sees Lectura as "another computer with a local copy."