Git/Project Instructions

I am going to assume that the bulk of development will be done in sessions. You will also maintain a development log as a markdown file. You should never delete anything from your log, even if you made mistakes and undid work. This log, together with your commit history, is how you show the invisible work you did.

Your code should not depend on your IDE or external libraries. Your code should work on the `cs1` and `cs2` machines.

Before Starting

1. Create a new local git repository. This repository will be used to contain your project and track its development.
2. Create a devlog.md in the repository's root directory.
3. Write your first log entry. The header for the entry should be the current date and time. Discuss what you know about the project, and what you are planning overall. Use this as a chance to get your thoughts out of your head.
4. Commit your devlog
   ```
   git add devlog.md
   git commit -m "Initial Commit"
   ```

Each Session

1. Add a new entry to the devlog. The current date and time should be used as the heading.
2. Answer the following.
   a. Give your thoughts so far. Have you had new thoughts about the project since the last session? It is alright for this part to be empty if there have been no new developments between sessions, but you should try to fill it in when you can.
   b. What do you plan to accomplish this session? Layout a plan, as detailed as possible. What do you plan to change? What do you plan to implement next? What is your goal for this session?
3. Commit the devlog
4. Work on your code. Feel free to add to the journal as you code when you encounter problems or learn something interesting. Feel free to commit your work multiple times as you code. By committing each time you implement a feature, it keeps a nice log of your work. This will better showcase your effort, and make it a bit easier to locate the source of bugs.
5. Once done for the session, add a new entry to your devlog. The heading should be the current date and time. Reflect on the session. Did you encounter any problems you have not already wrote about? Did you accomplish your goal for this session? If not, why? Did you accomplish more you have not written about? Record it now. What are you thinking about doing the next session?

Turning in Your Project

1. Make sure your working tree is clean. That is, the latest commit and the current code and devlog match.
2. Add a readme. This readme can be in markdown or plain text. It should describe the current files and their roles, how to compile/run the program from the command line, and anything you want to tell the TA before grading starts.
3. Zip the entire repository, and submit this zip file.

Tips

- Keep your devlog open while coding. This will make it easy for you to switch back to it to record a quick thought.
- If you want to make a quick change without sitting down for a whole session, Simply make the quick change and commit it. If you have not talked about the change before, add a quick entry to the devlog if you have time. If not, add it to the "thoughts so far" portion of the next session.
- Put effort into your devlog. The devlog and your commit history show all your unseen work, and is a major grading point.
- If you are using C/C++, consider using a Makefile.
- If you are using C/C++, consider compiling with the -g flag, and using gdb for debugging. It is great for when you have a segfault. The `backtrace` command will show the stack trace. The `info locals` command prints all local variables of the current frame. The `info args` command prints the arguments of the current frame. The `frame` command can change the current frame using the ids `backtrace` shows.
- If you want to experiment with an implementation, create a new git branch.
  ```
  git checkout -b experiment
  ```
  You can now work on your implementation. If you want to keep the changes, merge the experiment branch into the master branch. Be sure to commit all changes before merging.
  ```
  git checkout master
  git merge experiment
  ```
  If you decide you do not like it, switch back to the master branch. Make sure to commit any changes first. This will allow the TA to look at your experiment, a record of the work you put in.
  ```
  git checkout master
  ```
  Depending in the your configuration, the master branch may be named "main". You still want to keep any changes to your devlog.
  ```
  git checkout experiment devlog.md
  ```
  This will extract only devlog.md from the experiment branch to the working tree. You will still need to commit the changes to the current branch. You can name the branch anything you want. A more specific name would be better. Make sure to record about the branch in your devlog.
- If you find your self in a situation where you need to revert back to an old version of your code. For instance, perhaps you tried and failed to implement a feature and want to start over. There are several ways of accomplishing this in git, but we want to keep a record of all your work.

So, we will use `git revert`. First use `git log`, and identify the commit you want to revert to. Copy the hash of the one above it (`git revert` undoes specific commits)

```
git revert -n HASH..
```

The `HASH` is the hash you copied. The `..` tells git you want to undo all commits from the HASH to the most recent one. The `-n` tells git not to try and commit changes. This is necessary as git will ignore the `..` otherwise. You do not want to lose any changes to your devlog so lets extract that.

```
git checkout master devlog.md
```

Now you can commit the changes.

Grading Criteria

As an idea of how everything is weighted, here is a high-level breakdown.

Followed Specifications	30
Passes Tests (behaves correctly)	20
Devlog/Commit history	50