ASSIGNMENT 1

Specification

Outline

1. Introduction

This term, you will be using various tools in order to submit, build, and debug your code. This assignment is designed to help you get up to speed on some of these tools.

2. Setup

2.1. GitHub

Code submission for all assignments and labs in the class will be handled via GitHub so you will need a GitHub account. You will be provided with private repositories for all your homework and lab assignments. You must not use your own public repositories for storing your code. Following the course, please do not share your work in any form or with anyone.

2.2. Git Name and Email

Run these commands to set up your Name and Email that will be used for your Git commits. Make sure to replace "Your Name" and "your email@wou.edu" with your REAL name and REAL email.

```
$ git config –global user.name "Your Name"
$ git config –global user.email "your email@wou.edu"
```

2.3. Repos

You will have access to two private repositories in this course: a personal repository for assignments, and a personal repository for lab assignments. I may provide skeleton code for the assignments and labs, which you will need to download and add to your repository.

Clone the assignments repository from GitHub (one-time only).
https://classroom.github.com/a/dNyOw-Ky

3. Virtual Machine

I have prepared a virtual machine image that is pre-configured with all the tools necessary to run and test your code for this class.

Download and install the latest version of VirtualBox from:
https://www.virtualbox.org

Then, download the VM image from:
https://drive.google.com/drive/folders/14DqlEoU9LaBaw-r8jJ93D_zpYjiNpPbo?usp=sharing

4. Useful Tools

4.1. Git

Git is a version control program that helps keep track of your code. GitHub is only one of the many services that provide a place to host your code. You can use git on your own computer, without GitHub, but pushing your code to GitHub lets you easily share it and collaborate with others.

At this point, you have already used the basic features of git, when you set up your repos. But an understanding the inner workings of git will help you in this course.

If you have never used git or want a fresh start, I recommend you start here. If you sort of understand Git, this website will be useful in understanding the inner workings a bit more.

4.2. Make

make is a utility that automatically builds executable programs and libraries from source code by reading files called Makefiles, which specify how to derive the target program. How it does this is pretty cool: you list dependencies in your Makefile and make simply traverses the dependency graph to build everything. Unfortunately, make has very awkward syntax that is, at times, very confusing if you are not properly equipped to understand what is actually going on.

A few good tutorials are here and here. And of course the official GNU documentation (though it may be a bit dense) here.

For now we will use the simplest form of make: without a Makefile. (But you will want to learn how to build decent Makefiles before long!) You can compile and link wc.c by simply running:

```
$ make wc
```

This created an executable, which you can run. Try

```
$ ./wc wc.c
```

How is this different from the following? (Hint: run "which wc".)

```
$ wc wc.c
```

4.3. Man

man - the user manual pages - is really important. There is lots of stuff on the web, but man is definitive. The warm up to your first assignment is going to be to modify wc.c, so that it implements word count according to the specification of "man wc", except that it does not need to support any flags and only needs to support a single input file, (or STDIN if none is specified). Beware that wc in OS X behaves **differently** from wc in Ubuntu. I will expect you to follow the behavior of wc in Ubuntu, i.e., in your VM.

4.4. GDB

Debugging C programs is hard. Crashes don't give you nice exception messages or stack traces by default. Fortunately, there's gdb. If you compile your programs with a special flag -g then the output executable will have debug symbols, which allow gdb to do its magic. If your run

your C program inside gdb it will allow you to not only look get a stack trace, but inspect variables, change variables, pause code and much more!
Normal gdb has a very plain interface. So, we have installed cgdb for you to use on the virtual machine, which has syntax highlighting and few other nice features. In cgdb, you can use i and ESC to switch between the upper and lower panes.
gdb can start new processes and attach to existing processes (which will be useful when debugging your work.)

This is an excellent read on understanding how to use gdb.

Again, the official documentation is also good, but a bit verbose.
Take a moment to begin working on your wc. Provide the -g flag when you compile your program with gcc. Start the program under gdb. Set a break point at main. Run to there. Try out various commands. Figure out how to pass command line arguments. Add local variables and try probing their values. Learn about step, next, and break.

4.5. VIM

vim is a nice text editor to use in the terminal. It's well worth learning. Here is a good series to get better at vim. Others may prefer emacs. Whichever editor you choose, you will need to get proficient with an editor that is well suited for writing code.

If you want to use Sublime Text, Atom, CLion, or another GUI text editor, you're more than welcome to.

5. Assignment

5.1. Part I – Your own WC program

Your first task to write a clone of the tool wc, which counts the number of lines, words, and characters inside a text file. You can run the official wc in your VM to see what your output should look like and try to mimic its basic functionality in wc.c.

Just like the real version of wc your program should count the number of characters, words, and lines in file(s) provided as input. If no files are provided, it should read from STDIN. Additionally, your program should support flags, in particular -l -w -c .

While you are working on this take the time to get some experience with gdb. Use it to step through your code and examine variables. Be aware that wc in OS X behaves differently from wc in Ubuntu. I expect you to follow the behavior of wc in Ubuntu.

5.2. Part II – Fix It!

Being able to read code written by others and spotting the errors will be an important skill both in this class and in the real world.
Your second task is to look at fixit/uniq.c and fixit/sort.c. They are like the UNIX utilities uniq and sort, respectively.

An important difference is that uniq.c, by design, processes the input word-by-word instead of line-by-line. Each of these files is a standalone program and has a bug in it. For each program, write down the following in a text file called bugs.txt. Your answers should be short and concise.

- Briefly describe what the program is intended to do.
- Describe the bug(s).
- How would you reproduce the bug(s)?
- What are the symptoms of the bug(s)?
- How did you go about fixing the bug(s)?

There are relatively "clean" solutions to fixing all these programs. Please fix them.

6. Submission

1. Add the Assignment1 directory and all the contents to Git.
2. Commit your work with a useful commit message.
3. Create a tag/release for Assignment1.
4. Copy and paste that URL created in #3 into the assignment on the LMS.