1 Introduction

In this homework, we replicate the functionality of the standard Unix command line utility `wc`. This program counts the number of lines, words and/or characters of one or more files, depending on parameters passed to it.

2 Prerequisites

You may use the class server `words` to do this homework. It is accessed as follows:

`ssh -p 8190 YOUR_NETID_HERE@bits-head.cs.uic.edu` with your UIC password. Add this to your .ssh/config so that you can just type `ssh pages` on the terminal to log in. Also, set up passwordless ssh login, so that you don't have to type in your password every time. If this sounds unnecessary, **do it anyway.** Thank me later. Ask your classmates or google how if you need help with this part, but don't skip it.

For working on `words`, I recommend that you use Visual Studio Code on your own machine, with the *Remote - SSH extension* `https://code.visualstudio.com/docs/remote/ssh` to edit files on the server, rather than command line editors like vi or nano. It's really slick.

You may also want to set up your own environment, so that you still have the tools available to you when class is over. You will need the following:

- Optionally create a virtual machine with sufficient storage for Ubuntu 20.04 and some additional programs. 5 GB should be ample.
- Install Ubuntu 20.04 Desktop[1]
- Install rust following the instructions on rust-lang.org.

Keep in mind that our video conferences are likely to be resource hungry, and your machine may not support running both that and a virtual machine at the same time. A local Linux environment (VM or otherwise) is not needed to complete the class, but it can be very helpful if you need to do assignments offline.

3 The `wc` command line utility

Your assignment is to exactly replicate the behavior of the Ubuntu Linux 20.04 `wc` command line utility, with a Rust program. To get started, create a new project called hw1 using the cargo utility: `cargo new hw1`. This creates a project folder, including `hw1/src/main.rs` which is the file you will be editing.

[1] Ubuntu Desktop is a great choice, with lots of good free software for it. However if you, like me, prefer to work on a Mac desktop, you can install Ubuntu Server instead, and ssh to the VM from your mac. I can't help you with this part if you're a Windows user.

To run your program, `cd` into the hw1 folder, then run `cargo run`. This should produce the output `Hello world!`.

After running `cargo run` or `cargo build`, the binary executable created is `target/debug/hw1`. This is the program we will be testing.

3.1 Grading criteria

Grading will be done by comparing the output of your program, to the output of the stock `wc` program, for a number of different inputs. Specifically, we will test for:

- No parameters. In this case, your program should read from stdin.
- Combinations of -w, -c, and -l parameters in different orders, with and without files passed as parameters.
- One or multiple files as parameters.

In all cases, program should produce output that matching stock `wc`'s output exactly, byte by byte. You may use bash process substitution together with the `diff` utility to verify that the outputs of the two programs are identical, as follows:

```
diff -u <(./hw1 test.txt) <(wc test.txt)
```

The example provided compares the output of the commands `./hw1 test.txt` to that of the command `wc test.txt`. If the two commands produce identical output, diff produces no output, and your program passed the test. If it differs by as much as one byte, it fails the test. Some other examples (not exhaustive) that we will be testing:

```
diff -u <(./hw1 test.txt test2.txt) <(wc test.txt test2.txt)
diff -u <(./hw1 -c -l test.txt test2.txt) <(wc -c -l test.txt test2.txt)
diff -u <(./hw1 -l -c test.txt test2.txt) <(wc -l -c test.txt test2.txt)
diff -u <(cat test.txt | ./hw1 -w) <(cat test.txt | wc -w)
```

3.2 Double-check your submission

Before turning in your homework, verify that your program builds and runs correctly after downloading the submitted version, and that it passes all example test cases provided. Test cases will differ for grading purposes, but as long as your program is implemented with the intent of being general (i.e. no hard-coding of values), this should not present a problem.

3.3 turn-in instructions

To be determined, will be announced at a later date.