Assignment 0
CSE 130: Principles of Computer Systems Design

Due: April 13, 2022 at 12:00 AM

Goals This assignment will provide you with a refresher for programming in the C language. It serves three purposes: (1) to give you a chance to setup an Ubuntu 20.04 programming environment, (2) to give you an introduction to the CSE 130 programming assignments, and (3) to help you make sure that you're ready for this course by giving you a straightforward assignment. If you find that this assignment is very challenging, or that you're learning new content (i.e., that the assignment is not a review), then you might be most successful reviewing the C programming language and taking CSE 130 in another term.

Assignment Details

For this assignment, you will write a program, `split`, that takes a `delimiter` character and a list of files as input. The program "splits" each file into a set of lines by replacing each instance of a `delimiter` character into a new line character and prints the lines to `stdout`. To execute `split`, you specify:

```
./split <delimiter> <file1> <file2> ...
```

For example, to split a file `foo` into lines at each instance of the character, `a`, you would specify:

```
./split a foo
```

If `foo` contains the character sequence, `baabab`, then the above command prints the following to `stdout`: `b\n\nb\nb` (your program should print the new line character, '`\n`', rather than the sequence of characters \ followed by n).

Submission details Your repository should create a binary, called `split`, when we run `make` in the `asgn0` directory. `split` should implement the functionality listed below, subject to the limitations listed below. Additionally, your repository should also include a `README.md` file, which we will use in place of a design doc. You should include the interesting design decisions that you make and outline the high-level functions, data-structures, and modules that you used.

You must submit a 40 character commit ID hash on Canvas in order for us to identify which commit you want us to grade. We will grade the last hash that you submit to the assignment on Canvas and will use the timestamp of your last upload to determine grace days and bonus points. For example, if you post a commit hash 36 hours after the deadline, we will subtract 2 grace days from your total. Your repository should include the following files:

- `split.c`
- `Makefile`
- `README.md`

Functionality You must implement the following behavior:

- If `-` is given as a filename, then `split` should use standard input as the next input. That is, `split a foo -` specifies that `split` first splits `foo` and then splits standard input. You can assume that `-` will only be used once in the list of input files.
- `split` should work for an arbitrary number of files
- `split` must work on binary input files.

- `split` should produce the error messages and return codes that are identical to those of the reference implementation; see the section below on errors.
- `split` should not have any memory leaks.
- `split` must support any single-character delimiter.
- `split` must be reasonably efficient.
- Your code should be formatted according to the clang-format provided in your repository and it should compile using `clang` with the `-Wall -Werror -Wextra -pedantic` compiler flags.

Limitations You must write `split` using the C programming language. Your program cannot call the following functions from the C `stdio.h` library: `fwrite`, `fread`, variants of `put` (i.e., `fptuc`, `putc`, `putc_unlocked`, `putchar`, `putchar_unlocked`, and `putw`), and `get` (i.e, `fgetc`, `getc`,`getc_unlocked`, `getchar`, `getchar_unlocked`, and `getw`). You cannot use functions, like `system(3)`, that allow you to execute external programs. **If your `split` does not meet these minimum requirements, then the maximum score that you can get is 5%.**

Errors Your version of `split` must produce the same error messages and return codes as the reference implementation. You can find the reference implementation (in binary format, of course!) on Git@UCSC. Part of the assignment is enumerating as many errors as you can think of and running them through the reference implementation to see what output you should produce!

Hints Here are some hints to help you get going:
- You will likely need to lookup how some system calls (e.g., `read`) and library functions (e.g., `warn`) work. You can always Google them, but you might also find the man pages useful (e.g., try typing `man 2 read` on a terminal).
- You might find the `warn` function useful for producing error messages.
- Think about how to make your program more efficient. Inefficient programs will lose points.
- You can check the return code of a program by printing the value of `$?` on a terminal after you execute a program. For example, to see the code returned by `split`, you would run `split a foo; echo $?`.
- To format a file, named `foo`, using the included `clang-format`, execute the command `clang-format -i -style=file foo`.

Grading Your submission will be graded using the following high-level rubric:
- Functionality tests: 70%
- README design doc: 10%
- Coding style: 20%