CSE 2421 Lab 3

**Learning pointers, structs,
and command line arguments**

Due: Monday 19 June 2023 11:59 PM
Early: 2 bonus points if submitted by Saturday 17 June 2023 11:59 PM
Late: Tuesday 20 June 11:59 PM

We'll be using Carmen's timestamps to determine lateness or not. If you are not enrolled, you may send me an email with the zip file. If you are unable to submit by the deadline, you need to send me a screenshot of your finished files with the clock visible and then *still* submit to Carmen when you can.

This lab contains three parts. Make sure you do all of them!

Something you should do before getting started on the lab is creating your readme file. Simply create a file called "readme" and include your name at the top. As you progress through the lab, you may see different parts (labeled things like **Part 1**). When you see this, write it down in your readme. While in a numbered part of a lab, you'll come across various questions, prepended with a bolded Q and a number (such as **Q5**). When you see this, include the Q and the number, along with you answer to the question(s) in the file. A brief example:

> Name: Rob LaTour
>
> I certify that I completed all of the work myself with no aid from anyone aside from the instructor or the undergraduate graders.
>
> Part 1:
>
> Q1: Don't make fun of me, but my favorite color is gray
>
> Q2: The definitive answer is that Han did shoot first. The original version is the one that captured the heart of the public, and thus the director lost creative control. In this essay I will…

Part 0: A quick introduction

For this lab, we'll be implementing a fake file system. The task is to create a program that allows us to create "files" and "directories," move them, and display information. First, however, we're going to make some assumptions about our file system so that coding this beast will be a little easier.

1. Each element of the file system may be either a directory or a file
2. Names of files and directories can be no more than 64 characters long
3. File contents will contain no more than 256 characters.
4. No directory will have more than 64 children (this includes files and subdirectories)

Part 1: File Systems

The first part of the lab requires you to implement a program that allows us to build a fake file system using linked data structures. This must be done within your program. This means you may not simply have your program run Linux commands! Your program should be able to support the following commands:

1. cd
2. ls
3. pwd
4. cat
5. cp
6. mv
7. rm
8. mkdir
9. rmdir (up to you if you want to require the directory to be empty or to use extra flags to make it more powerful)
10. file (this is not a Linux command, but to avoid having to write your own text editor, this command should create a "file" and ask the user to input the text contents of the file)
11. help (list these commands and what they do)

Q1: How do the elements of your file system structure differ from the nodes of the linked structures we discussed in class?

Q2: You may have to print that you are at the root directory with no children. How will you convey to the user that there is a directory here with no children, rather than simply printing nothing?

Q3: Do you think "terse" insertion and deletion functions (similar to what we discussed in class) are possible with this lab?

Part 2: Make it your own

For part 2, your task is to customize your file system in some way. You must add three different command line commands that do whatever you want, in addition to one final one: dirprint. The dirprint command should output in some sort of visually pleasing way (up to you to design this) the current directory's name, the siblings of the current directory, the parent of the current directory, and so on, up to and including the root.

Q1: What extra commands did you choose to implement? Why?

Q2: Did you use any other data structures to help with the extra commands? Why or why not?

Q3: What was the motivation for displaying the directory structure in the way that you did?

Part 3: Adding in command line arguments

In this final part of the lab, you are tasked with adding one extra piece of command-line-argument-goodness to your code, with the following results:

1. If your program is run without an extra command line argument, then it will run exactly as the above instructions state
2. If your program is run with a command line argument, your program will first treat the argument as a directory path and read the contents of that directory and create a file at the root directory level for each file in the directory. The contents of these files should be some dummy value.
   Example: If I have 3 files (file1, file2, file3) in the current directory and I run my program with a "." as the command line argument, then my program should create three files with the same names (file1, file2, file3) in your program.

Some notes:

1. You do not need to try to account for file names that are too long. Just assume that no file name in the actual Linux environment will be too long. It is your responsibility to handle directories that have too many items in them!
2. You can put whatever you want as the dummy contents of the file
3. Check out the dirent.h header file!
4. If you want to make life slightly easier, you also don't have to care about file type when doing this. Consider anything in the specified directory to be a "file" to be created in your program.

Q1: Provide a full set of instructions for how to run your program at the top of your readme (after your name).

Q2: How did you handle the case where the provided directory had more than 64 items in it?

Q3: Please include a set of Linux commands to create a test directory for the above functionality.

Q4: How did you handle the case where the command line argument provided wasn't a valid directory? Should such cases be your responsibility to handle? Why or why not?

When submitting this lab, please include all c files used to produce your executable, your makefile, and your readme.