CSCI-442 : Project 1 - Tree Command

Due: Please see the assignment on Canvas for dates.

Introduction

For this project, you'll implement a program which prints a directory tree to the terminal. This program should be implemented in the C programming language, and run on the Linux operating system.

Learning Objectives

- Practice C programming with a simple introductory project.
- Parse a very simple command line argument.
- Refresh your knowledge of Linux directory structures from CSCI-274.

Project Requirements

Functionality

After running `make`, you should have an executable program named `tree` located in the root of your repository. This program should take one optional command line argument: the directory to tree, which may be either an absolute or relative path. If this argument is not specified, you should tree the current working directory.

If your program encounters a symlink, you should output the special line (indented to the correct level):

```
SYM filename -> link_path
```

Output Format

You should output each filename on it's own line. When outputting a directory, you should output the directory name, then tree the entries of the directory by one (further) tab character (ASCII value 9).

You should not output the `.` or `..` entries. Recall from CSCI-274 that these are used for the current and parent directory, respectively.

The order of entries within a directory may be sorted in any order.

> ***Warning***
>
> Please note that the C strings (i.e., char arrays) you will use to store and print the paths should be null terminated (https://www.tutorialspoint.com/cprogramming/c_strings.htm). Otherwise, although your output visually looks good, it will not pass our automated grading tests.

Examples

Suppose that you are located in a directory `/tmp/foo`, and this directory contains three entries, excluding `.` and `..`:

- A directory named `bar`, containing a directory named `baz`, and two files: `prog.c` and `fib.c`. `baz` contains a directory named `bam` and another directory named `bip`. `bam` contains a file named `a.out` and an empty directory named `b.files`. `bip` contains one file: `zip.txt`.
- A directory named `projects`, containing two files `TOP_SECRET`, and a symlink to `/tmp/foo/bar` named `teleportation`.
- A file named `a.tar.bz2`.

```
$ /path/to/your/tree
bar
        fib.c
        baz
                bip
                        zip.txt
                bam
                        b.files
                        a.out
        prog.c
a.tar.bz2
projects
        SYM teleportation -> /tmp/foo/bar
        TOP_SECRET
$ /path/to/your/tree bar/baz
bip
        zip.txt
bam
        b.files
        a.out
$ /path/to/your/tree ../foo/bar/baz
bip
        zip.txt
bam
        b.files
        a.out
```

General Requirements

- You should handle errors gracefully. For example, if you are not able to access a directory, print a relevant and descriptive error message to the `stderr` file stream (not `stdout`), and continue to traverse the other directories you still have permission to. Your program should have a non-zero exit status if any errors are encountered.
- Your program should have a zero exit status if no errors are encountered.
- Your project must be written in the C programming language, and execute on the ALAMODE lab machines.
- You should follow Linux Kernel coding style, a common style guide for open-source C projects. A small number of points will be deducted on projects which do not follow this style guide.
- Your project must not execute external programs or use network resources.
- Your project should be memory safe. For example, if your program is susceptible to buffer-overflow based on certain inputs, it is not memory safe. As a corollary to this, you should not use any of the following functions: `strcat`, `strcpy`, or `sprintf`.
- You should `free` any memory that you heap-allocate, and `close` (or `closedir`) any files that you open.

- To compile your code, the grader should be able to `cd` into the root directory of your repository and run `make` using the provided `Makefile`.

Getting Started

The starter code provides a `Makefile` and boilerplate for a main function, but does not dictate how you should structure the code for your implementation. This means you are going to need to create some of your own software design, e.g., make new functions and potentially even new files. In other words, this is not a "fill in the functions and you will have a working project" sort of starter code.

You are free to dispose of any parts of the starter code you don't want to use.

Resources

You may find the following manuals on your system of use:

- `opendir(3)`
- `readdir(3)`
- `closedir(3)`
- `readlink(2)`
- `nftw(3)`

You can open these using the `man` command. For example:

```
$ man 3 opendir
```

You cannot copy/paste/modify/derive code from an existing solution taken from any external resource (e.g. internet, student etc.)

Submitting Your Project

Submission of your project will be handled using Gradescope. Further instructions will be provided.

Collaboration Policy

Please see the syllabus for the course plagarism policies.

This is an **individual project**. Plagarism cases will be punished harshly according to school policies.

Please do keep any Git repos private, even after you finish this course. This will keep the project fun for future students!