Overview

In this assignment, you will write a small version of the `ls` program to list the files in a directory.

This assignment is divided into several parts.

Requirements

Your program should follow general Unix programming principles:

- exit with 0 on success, nonzero on error.
- use `getopt` to process the command-line options

Basic Functionality (60 points)

Your program should take options followed by zero or more directory or file names. Use `getopt` to parse the options.

It must recognize the following options:

- `-a` to enable listing all files (including hidden files, those starting with a `.`).
- `-d` to treat directories as files.

What each of these means is discussed in more detail below.

If your program was passed the `-d` option, then it should iterate through the provided paths and print out each one to standard output if it exists. An error should be printed to standard error for each path that does not exist.

If the `-d` option was *not* passed, then the program's behavior should follow two steps.

1. Iterate through the paths and print out the ones that are not directories (or symbolic links to directories).
2. For each directory, print its *contents*: the names of the files, directories, etc. that it contains. Do not include entries whose names begin with a period, unless the `-a` option was activated.

   If the directory is the only argument, just print its contents. If it is one of many arguments, including files or other directories, then before printing its contents, print the name of the directory followed by a colon.

If the program is not given any path name arguments — either it receives no arguments, or only option flags — then it must act as though it was given a single path name '.' (the current directory).

The program should produce its output with one file name per line; it should not try to replicate the common `ls` multi-columned output.

Long Listings (15 points)

Your program should also support the `-l` option. This option should enable *long mode*; in long mode, instead of just printing each file's name, the program should print the following:

- the file's type (`-` for regular file, `d` for directory, `l` for symbolic link, `c` for character special, `b` for block special, `s` for socket, `f` for FIFO)
- the file's *mode* (permissions) (`st_mode` field of `stat`); this can be printed as a 4-digit octal number (`%04o` in `printf`).
- the number of hard links (`st_nlink` field of `stat`)
- the file's owner and group (`st_uid` and `st_gid`); these can be printed as numbers.
- the file's size (`st_size`)
- the file's last modification time (`st_mtime.tv_sec`)
- the file's last modification time (`st_mtime.tv_sec`)
- the file's name
- if the file is a symbolic link: its target (so it appears ‘name -> target’)

The format should look like `ls -l`:

```
-0644   1 503  20    320 Jan 21 14:52 Makefile
d0755  23 503  20    782 Feb 17 12:40 assignments
d0755  16 503  20    544 Feb 12 16:41 demo
d0755  11 503  20    374 Feb 12 12:06 exams
d0755  24 503  20    816 Feb 12 12:18 lectures
d0755  10 503  20    340 Feb 13 15:43 machines
-0644   1 503  20     41 Feb 10 12:51 settings.gradle
-0644   1 503  20  18584 Jan 20 15:05 syllabus.dvi
-0644   1 503  20   3035 Jan  9 15:00 syllabus.html
-0644   1 503  20  11729 Jan 15 17:16 syllabus.man
-0644   1 503  20  37248 Jan 20 15:05 syllabus.pdf
d0755   3 503  20    102 Jan 26 10:44 tools
d0755   7 503  20    238 Feb  6 14:03 videos
```

Sorted Listings (10 points)

Your program should sort its listings. If no sorting options are provided, then:

- files/paths from the command line should be processed in the order in which they appear.
- the contents of directories should be printed in lexicographical order by file name, using `strcoll` to compare file names.

Your program should also recognize the following 2 options for sorting:

- `-t` will sort files by modification time (`st_mtime`), with the most recent (largest time value) appearing first.
- `-S` will sort files by size, with the largest files appearing first.

These options apply *both* to files from the command line, and to the contents of directories. Unless `-d` is specified, the directories should still be handled separately from non-directory paths on the command line.

I recommend loading the file names to sort, along with their `stat` structures, into an array or linked list and sorting that prior to printing. You can implement the sort yourself, or use the `qsort` function found in the C standard library.

Recursive Listings (15 points)

Your program should support another option, `-R`, that enables *recursive* listings.

In recursive listing mode, if a directory entry is a directory, then in addition to being listed with its siblings as the contents of its parent directory, its contents should also be listed as if the directory were also provided on the command line. This only takes effect if `-d` is not also supplied (if `-d` is supplied, then `-R` does nothing).

Human-Readable Long Listings (10 points extra credit)

Augment your long listing support to print human-readable information:

- user and group names (get these with `getpwuid` and `getgrgid`)
- human-readable file modes (e.g. rwxr-xr-x instead of 0755)

Submitting

Submit your program as a single `.c` file on TRACS.

Useful Functions

- `stat(2)`
- `lstat(2)`
- `opendir`, `readdir`, `closedir`
- `qsort`