Assignment 01

If any of this is unclear, please get in touch with me ASAP. There are definitely things in the course I want you to struggle with. I absolutely do NOT want you struggling with understanding my expectations.

In this assignment, you will implement basic functionality of the `ls` program. Specifically, your implementation must work the same as the real `ls` program under the following scenarios:

- when run with no arguments;
- and when run with any combination of the arguments `-l`, `-a`, any number of files, and any number of directories ("any number" within reason).

You will have to experiment and/or research how `ls` behaves in these cases so that you can duplicate it.

The logic itself is not terribly complicated; I expect and intend much of your time will be spent reading man pages and acclimating yourself to maneuvering around Linux and C.

Goals

The goals of this assignment are for you to

- begin developing proficiency with C;
- begin developing familiarity with man pages;
- design and implement a fairly basic program.

New Mechanics

You might find these useful in this assignment.

New functions

- `opendir(3)`
- `readdir(3)`
- `closedir(3)`
- `getopt(3)`
- `stat(2)`
- `strftime(3)`
- `getpwnam(3)`
- `localtime(3)`

Separately, the `stat(2)` manpage lists the fields of the `struct stat` populated by `stat(3p)`; the `inode(7)` manpage goes into further detail. Note that file permissions (eg, read, write, execute) are called the "mode".

Behavioral requirements

You will produce a source file, `myls.c`, that implements the required logic. When compiled and run with no arguments, it should list all non-hidden files in the current directory. When supplied with a list of arguments, it should behave as `ls` does: for all non-directories, list the files themselves; for all directories, list the contents; error if a file doesn't exist. Ordering of the output is at your discretion (you will notice that, in a listing with both files and directories, `ls` will show all files first and then all directories, regardless of the order they appeared on the command-line; you do *not* need to do this). Unless a single filename exceeds 80 characters, the output of your program should not exceed 80 columns.

Furthermore, your `ls` program must accept the `-a` and `-l` options. When provided with the `-a` option, your program should list *all* files in the directories given, including hidden files. When provided with the `-l` option, your program should provide detailed information about each file, one file per line, just as the standard `ls` program does.

When provided with the `-l` option, your implementation does *not* need to print the "total ##" line that the standard version of `ls` prints.

You must use `getopt(3)` to parse command-line arguments.

You do *not* need to match the spacing of the real `ls`.

You may assume that a path will never exceed `PATH_MAX` bytes in length, even though this isn't really true.

Deliverable requirements

- `myls.c`, which should contain your implementation of `ls`.

- `Makefile`, whose default target should build `myls` (out of `myls.c`), It may have as many other targets as you like, to make your life easier), but the default one *must* build that program. It must also have a `clean` target that removes all built files.
- `README`, which will include a list of authors, a list of known bugs, and a list of resources you consulted in your work (a list of URLs is totally acceptable—if I can see the kinds of references you're using, I can get a better idea of how to tailor the instruction to your needs—just keep the README file open as you work and paste them there when you find something useful).

Style requirements

Your code must exhibit:

- appropriate use of comments;
- appropriate use of stack, heap, global, and static variables;
- appropriate use of constants;
- efficient memory usage;
- no magic numbers;
- appropriate decomposition (a good rule of thumb is functions should be no more than 1 screen long);
- clear organization (eg, grouped header files, struct definitions, function prototypes);
- well-named variables, functions, and other identifiers;
- verification of command-line arguments, if applicable;
- checking and appropriate handling of all return values;
- consistent style (capitalization, indenting, etc).

Hard Mode [optional!]

Here are some *optional* features you might consider adding. You do *not* need to implement any of them to receive credit, and I strongly recommend you do not try unless and until you have finished the rest of the requirements and wish to explore further This list is provided solely as idea fodder if you want to go above and beyond.

- Sort the output by filename. You will likely find the `malloc(3)` and `qsort(3)` library functions helpful.
- Implement the `-t` option, as documented in `ls(1)`.
- Implement the `-h` option, as documented in `ls(1)`.
- Implement the `-R` option, as documented in `ls(1)`.

Submission Process

Your submission files should be in a git repository on basin in this directory:

```
/home/<username>/<secret>/assignment1
```

Email me when you want feedback, with a description of the type of feedback you would like, and the identifier of the revision on which you would like feedback.

Last modified: 09/29/2022 15:32:41