

CS& 131, Autumn 2020

Programming Assignment #8: Terminal Simulator (20 points)

Due Friday, December 4, 2020, 11:00 PM

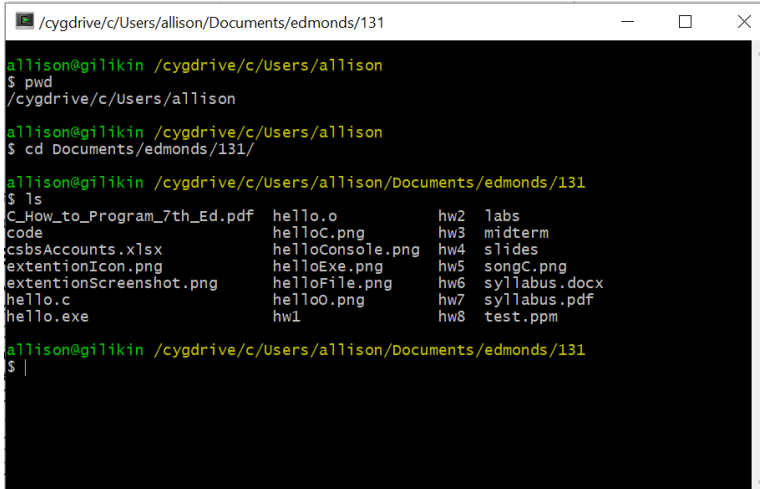
Thanks to Marty Stepp for parts of this assignment

For this assignment you will write a terminal simulator. This assignment focuses on **recursion and bit manipulation**. Turn in a file named `terminal.c`.

Most of us are used to getting around our computers with a **graphical user interface (GUI)** these days. We double click on files and programs to open them. Originally you could only get around your computer by typing instructions into a terminal to change directories, open files, run programs, etc. You can still use a terminal to get around your computer and they are really, really commonly used by programmers because they allow you to have **more direct and quick control**.

There are many different types of terminals. For this assignment we will be implementing a couple commands that are available under different names in almost all terminals.

You can see a screenshot of a real terminal at left. It shows the **`pwd` command that prints the current directory of the terminal**, the **`cd` command that moves to a different directory** and the **`ls` command that lists the files in the current directory**.



```
/cygdrive/c/Users/allison/Documents/edmonds/131
allison@gilikin /cygdrive/c/Users/allison
$ pwd
/cygdrive/c/Users/allison
allison@gilikin /cygdrive/c/Users/allison
$ cd Documents/edmonds/131/
allison@gilikin /cygdrive/c/Users/allison/Documents/edmonds/131
$ ls
C_How_to_Program_7th_Ed.pdf  hello.o          hw2  labs
code                        helloC.png       hw3  midterm
csbsAccounts.xlsx          helloConsole.png hw4  slides
extentionIcon.png          helloExe.png     hw5  songC.png
extentionScreenshot.png    helloFile.png    hw6  syllabus.docx
hello.c                    helloO.png       hw7  syllabus.pdf
hello.exe                  hw1              hw8  test.ppm
allison@gilikin /cygdrive/c/Users/allison/Documents/edmonds/131
$ |
```

Program Description:

In this assignment you will be **writing a console program that prompts the user with a `$` and waits for them to type in a command until the user types in `logout`**. Your program should be able to respond to the following four commands:

- **`pwd`** : prints the current directory

You can find the **current directory** by calling the `getcwd(string_variable, max_size)` . To use this you will need to `#include <unistd.h>`.

Example: `char directory[200];`
`getcwd(directory, 199);` // directory now stores the current directory

- **`cd path`** : **sets the current directory to the path**

The path the **user types in might be an absolute path or a relative path**. Assume that **absolute paths start with `/` or a single letter followed by `:`**.

Example absolute paths: `/cygdrive/c/users`
`C:\Users\allison\Documents\edmonds\131\code`

Example relative paths: `../../`
`cs131/slides/`

- **`ls path options`** : **lists the files**

This can be called with **no path or options**. If so, just list the names of **all** the files and directories in the current directory.

If it is called with an **absolute path** then just **list the files and directories in the directory at that path**.

If it is called with a **relative path** then list the files in the directory **at the current directory + the relative path**. For example, if the user types the relative path `../../` and the current directory is `/documents/cs/131/slides` your program should list all the files and folders in `/documents/cs/131/slides/../../`

Options

Options will always appear after a `-`. They will have **no spaces** in between but may be listed in any order.

- **a** : list all files. Unless the a flag is included files that have names that start with . should not be printed.
- **l** : list files with all extra information. You should include the file permissions, file size and then the file name. You can find this file information by using the `stat (full_file_path, stat_struct_pointer)` function. To use this you will need to `#include <sys/stat.h>`. You can then access `st_size` in the struct to get the size and `st_mode` to get the permissions.

Example:

```
struct stat path_stat;
stat(path, &path_stat);
printf("%d %o", path_stat.st_mode, path_stat.st_mode);
```

`st_mode` is an integer. The 9 rightmost bits represent the permissions with separate permissions for each type of user. See which bits map to which permission in the table below:

User			Group			Other		
read	write	execute	read	write	execute	read	write	execute
1	1	1	1	1	1	1	1	1

You should output these permissions as a string of r, w, x and - characters. If the bit is set to 1 output a r, w or x (depending on what it is a permission for). If the bit is 0 output a dash. For example if `st_mode` was set to 10000000 00000000 10000001 11100100 you should output `rwxr--r--`. This shows that the user has read, write and execute permissions on the file and the group and everyone else just have read privileges.

To receive full credit for this portion you must use bit operations to extract whether each permission flag is set on the number.

- **r** : list files recursively. This means that you should list all files and folders in the directory. If the directory you list contains other directories list all files and folders in them. If they contain directories list all files all files and folders in them. Keep doing this until there are no more levels of folders left. You should print out the files and folders as you go. To help make the output more readable you should indent 4 spaces more every time you go down another directory level.

For full credit this option must be implemented recursively.

When your function starts, get all files and folders in the current directory one by one. If you get a file just print out its name. If you get a folder, print out its name, increase your indentation level and then do the same thing.

You can get the contents of a directory by using the `opendir (directory_path)` and `read_dir (directory)` functions. To use these you must `#include <dirent.h>`.

Example:

```
DIR *dir = opendir("/users/allison/documents");
struct dirent *entry = readdir(dir);
printf("The file/folder name is %s\n", entry->d_name);
```

Note that you will have to call `readdir (dir)` repeatedly until it equals `NULL` to get all entries in the directory. The above code just gets the entry for the first file or folder in the directory.

Once you have the entry you will need to check if it is a file or a folder. You can do this using the `S_ISDIR (mode)` function. This function takes the `st_mode` from the `stat` struct, described earlier in this document, as a parameter. It will return `true` if the entry is for a directory and `false` otherwise.

Example:

```
struct stat path_stat;
stat("/users/allison/documents", &path_stat);
return S_ISDIR(path_stat.st_mode);
```

- **logout** : stops the program

Program Behavior:

Your program begins by **printing out the full path of the current working directory**. Then your program **prints a \$ and waits for the user to input** a command. After each command it should output a **\$** and wait for the **user's next command**. Note that your output will not exactly match these logs as you have different files and folders on your computer.

Log of execution 1 (user input underlined):

```
Current working dir: C:\Users\allison\Documents\edmonds\131\code
$ ls
adding_game.c
bits.c
bmr.c
bmr.exe
cities.txt
$ ls -l
rw-rw-rw- 1444 adding_game.c
rw-rw-rw- 1474 bits.c
rw-rw-rw- 2375 bmr.c
rwxrwxrwx 55589 bmr.exe
rw-rw-rw- 46 cities.txt
$ ls -a
.
..
.test
adding_game.c
bits.c
bmr.c
bmr.exe
cities.txt
$ ls -la
rw-rw-rw- 1000 .
rw-rw-rw- 1423 ..
rw-rw-rw- 14 .test
rw-rw-rw- 1444 adding_game.c
rw-rw-rw- 1474 bits.c
rw-rw-rw- 2375 bmr.c
rwxrwxrwx 55589 bmr.exe
rw-rw-rw- 46 cities.txt
$ logout
```

Log of execution 2 (user input underlined):

```
Current working dir: C:\Users\allison\Documents\edmonds\131\code
$ cd ..
$ pwd
C:\Users\allison\Documents\edmonds\131
$ ls
code
slides
website
syllabus.doc
syllabus.pdf
$ ls -r
code
    adding_game.c
    bits.c
    bmr.c
    bmr.exe
    cities.txt
slides
    week1
        print.ppt
        print.pdf
    week2
        old
```

```
        drawing.ppt
        drawing.pdf
        drawingFiles
            DrawingPanel.java
website
    test.html
syllabus.doc
syllabus.pdf
$ ls C:\Users\allison\pictures\cats
merlin.jpeg
percy.jpeg
$ logout
```

Style Guidelines:

For full credit you must use **recursion** to print out the contents of directories inside directories. It is just fine to output the contents of a single directory level with a loop. You also must use **bit operations** to figure out what permissions string to print.

We will grade your function structure on this assignment as we have on others. Use **at least four nontrivial functions** besides `main`. These functions should use **parameters and returns, including arrays and structs, as appropriate**. The functions should be well-structured and avoid redundancy. No one function should do too large a share of the overall task.

Your `main` function should be a concise summary of the overall program. It is **okay** for `main` to contain some code such as **`printf` statements and your main program loop**. But the `main` function should not perform too large a share of the overall work itself, **such as getting directory entries and checking if they are files or folders**.

We will also check for redundancy on this assignment. If you have a very **similar** piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a function, by using `for` loops over the elements of arrays, and/or by factoring `if/else` code.

Follow past style guidelines such as indentation, names, variables, types, line lengths, and **comments** (at the beginning of your program, on each method, and on complex sections of code).