Project 1

For this assignment you will be implementing a primitive file-system shell. Your program will need to be able to navigate paths and traverse a directory structure. For the directory structure you will want to create an n-ary tree where the number of children is not limited. There are two ways to achieve this. One way is to change nodes from having 2 fixed children to having a list of children. Another way is to change the meaning from 'left' and 'right' pointers to 'nextsibling' and 'firstchild' pointers. This last pattern simulates an n-ary tree through a binary tree. A parent node only points to the first child out of its children and has to walk the nextsibling pointer from the first child to get to the second child, walk the nextsibling pointer from the second child to get to the third child, and so on.

Commands given from commandline:

```
ls
ls <path>
pwd
mkdir <path>
touch <path>
cd <path>
rm <path>
rm -f <path>
find <local name>
```

Core Concepts:

1. Path - navigation of the directory structures
2. cwd - Current Working Directory (for executing commands)
3. pwd - Path from root directory to cwd
4. ls - list contents of cwd or specified directory.
5. mkdir - create a directory at the specified path.
6. touch - create a file in the proper directory

7. cd - change the cwd to the directory specified by the path.
8. rm - remove specified file or directory (error if directory not empty)
9. rm -f - force remove of specified directory, deleting all sub-directories and their content.
10. find - list the paths to all directories and files with the a matching local-name.

ORDER OF CHILDREN IN DIRECTORY

- The contents of a directory are saved on a list in the following order:
  1. Directories are listed before files.
  2. Within directories and files regions, contents are sorted in the following lexicographic ordering (semi dictionary order)
     - '.' < '-' < '_' < (0-9) < (A-Z) < (a-z)
- You may need to use a custom compare function to compare character by character to guide your inserting new files and directories in proper order.

Examples In Order

```
file
file.txt
file-name.txt
file_name.txt
file1name.txt
fileName.txt
filename.txt
```

1 Paths

Names of files and directories are at most a sequence of 20 characters from A-Z, a-z, 0-9 and '.', '-', '_'

For any given directory, the names of all immediate children (files and directories) must be unique.

The name '.' refers to the current working directory (CWD)

The name '..' refers to the parent directory of cwd (or the root when cwd = root).

A Path is a sequence of names separated by '/' symbols.

The max PATH length is 1000 characters.

Paths which start with the '/' character are called Absolute Paths

The absolute path '/' refers to the root directory

The absolute path '/A/B/C/D' refers to the file or directory D inside directory C, C is inside directory B, B is inside directory A, and A is a directory inside the root directory.

Paths without a leading '/' character are called relative paths, relative to the CWD.

The relative path 'A/B/C' is the directory or file C inside the directory B, B is inside the A directory and A is inside the current working directory.

Paths are recursive constructs. The pattern "PATH/." where PATH is an arbitrary path refers to the directory indicated by "PATH". The pattern "PATH/.." where PATH is an arbitrary path refers to the parent directory of the directory indicated by "PATH". If "PATH" is the root directory, then "PATH/.." is the root directory.

The last name on the path may be the name of a file or directory, but all other preceding names on the path must be names of directories in order for the path to be usable in the file system.

1.1 Common Path Usage and Errors

These path operations should get their own helper functions

1. GetPWD(NODE* n): PATH
   (a) returns the absolute path to the node n.
   (b) walk the parent pointers back to the root node to figure out the path (in reverse).
2. Follow(NODE * n, PATH P): NODE *
   - If P is absolute, set n to point to the root
   - Walk the path from n and return the pointer to the directory or file indicated by the path
   - If an error occurs, return NULL;
   - Error Case: a directory along prefix of path does not exist.

- Error Output: `Path Error: directory '<Absolute PATH To Missing Directory>' does not`

3. SafeToCreate(NODE* n, PATH P): boolean
   - Starting from n for relative paths, checks to see that the prefix of PATH P that already exists is a directory and that it is safe to create the missing remainder of the path.
   - Error Case A: a name along the path is to a file and not a directory.
   - Error Output A: `Path Error: Cannot create sub-directory content, '<Absolute PATH To`
   - Error Case B: the path given indicates a file or directory that already exists.
   - Error Output B: `Path Error: '<Absolute PATH>' already exists and cannot be created.`

2 LS - (list directory contents)

The 'ls' command without a following path is short for 'ls .', requesting a listing of the current working directory's content.

This command does not change the actual current working directory.

Steps for 'ls PATH':

1. Follow the path to the indicated directory
2. If a path error occurred, print list error and return (Path error already printed)
   - Error Output: `List Error: Cannot perform list operation.`
3. If the successful following of a path indicates a file, print error and return
   - Error Output: `List Error: Cannot perform list operation. '<Absolute PATH>' is a fil`
4. Print `Listing For '<Absolute Path>':` on its own line then print the names of the directories and files in order of their occurrence on the child list (list should be sorted already). (See Example Below)

PWD is '/dir1'

Command: 'ls dir2'

```
Listing For '/dir1/dir2':
D dir3
D dir4
D dir5
F file1
F file2
```

3 MKDIR - Make Directory

Steps For 'mkdir PATH'

- Check that it is safe to create the directory indicated by the path, otherwise print error message and return
  - Error Output: `Make Dir Error: Cannot create directory.`
- Create all missing directories along the given path.
  - be sure that each directory is inserted into the proper order within it's parent's child list.

4 TOUCH - Make File

Steps For 'touch PATH'

- Check that it is safe to create the file indicated by the path, otherwise print error message and return
  - Error Output: `Touch Error: Cannot create file.`
- Create all missing directories along the given path then create the final file.
  - be sure that each directory and file is inserted into the proper order within it's parent's child list.

5 CD - Change Working Directory

Steps for 'cd PATH'

- Make sure you can follow the path, if not print out an error message and return
  - Error Output: `Change Dir Error: Cannot change working directory.`
- Set the current working directory to the directory indicated by following the path.

6 RM - Remove File or Directory

Steps for 'rm PATH'

- Make sure you can follow the path, if not print out an error message and return
  - Error Output: `Remove Error: Cannot remove file or directory.`
- remove the file or empty directory (no children).
- If the directory is not empty, give an error
  - Error Output: `Remove Error: directory '<Absolute Path TO Directory>' is not empty.`

Steps for 'rm -f PATH'

- Make sure you can follow the path, if not print out an error message and return
  - Error Output: `Remove Error: Cannot remove file or directory.`
- remove the file or directory (recursively deleting the sub-tree under the directory).

7 FIND - Search file system for file or directory

Print in lexicographical order (pre-order visitation of directories) the absolute paths to all nodes which contain the name being searched for. Print D or F preceding the absolute path to indicate if it is a directory of file.

Example: 'find taxes'

```
Searching For 'taxes':
D /taxes
F /taxes/1995/taxes.pdf
F /taxes/1996/taxes.pdf
D /taxes/2006taxes
F /taxes/2006taxes/2006taxes.pdf
```