CS 3100 – *Data Structures and Algorithms*
Project #2 – Directory Tree

Learning Objectives

- Apply basic object-oriented programming concepts in C++
- Design, implement, and use a general tree data structure
- Analyze operations for time complexity

Overview

The directory (folder) structure on a computer can be represented as a general tree. Your task for project 2 is to simulate this. We discussed four different ways to implement a general tree during lecture — you may use any of these for your project.

Your program should not place any limits on the breadth or depth of the directory tree.

You should write a driver program creates a single directory called "root" and then continually waits for the user to type in a command and performs the requested action. Your program must support the commands listed below. The program should end if the user types "quit", and it should display the message "Unknown command" if the user enters anything that is not in this list (other than "quit"). The commands should be considered case-sensitive (i.e. if the user types in LS instead of ls, the program should output "Unknown command").

Your code should contain a comment explaining the worst-case asymptotic time complexity of each operation listed below.

Available Commands:

- **pwd** — this should display the name of the current directory

- **ls** —this should display the names of all children of the current directory; if the current directory has not subdirectories, display the message "This directory is empty" (note that the 'l' is a lowercase L)

- **exists <directory>** — displays the message "<directory> exists" if <directory> exists anywhere in the directory structure; otherwise displays the message "<directory> does not exist". Note that you may have more than one directory with the same name. This command should say the directory exists if there is at least one directory with the specified name.

- **cd <new_directory>** — this should change the current directory to <new_directory> but only if <new_directory> is a child of the current directory; otherwise it should display the message "<new_directory> not found".

- **cd ..** — the special command cd .. should change the current directory to the parent of the one you are in when the command is entered. If you are already in the root directory, this command should display the message "You are in the root directory"

- **mkdir <new_directory>** — this should create a new directory named <new_directory> whose parent is the current directory

- **rmdir <directory_to_remove>** — this should remove <directory_to_remove> and all of its subdirectories but only if <directory_to_remove> is a child of the current directory; otherwise it should display the message "<directory_to_remove> not found".

- **countdir** — should display the number of directories in the subtree rooted at the current directory (i.e. the number of descendants of the current directory + 1)

Extra Credit:

- **showtree** — this should display the current directory and all of its descendants using a pre-order traversal, with each directory name tabbed over based on its depth (i.e. the children of the current directory should be tabbed over one, the children of those directories tabbed over two, and so on)

Turn in and Grading

Please turn in all .h and .cpp files that you write for this project into the dropbox on Pilot. Please do not zip or compress your files.

Your project will be graded according to the following rubric. The number in [ ] is the number of points each item is worth. ***Projects that do not compile will receive a zero.***

[10] Driver program reads user commands and responds differently based on which command was entered. The program halts if the user enters 'quit'.

[10] You have implemented a workable general tree (at a minimum you can add items to it) and there is no limit on its breadth or depth, nor any memory leaks.

[80] Each of the eight commands listed under "Available Commands" is worth nine points for a correct implementation, plus one point for correctly stating its time complexity.

Extra Credit: Correct implementation of the showtree command is worth an additional 9 points of extra credit, and correctly stating its time complexity is worth one additional point of extra credit.