ENSC 251 – Summer 2023 – Course Project Part 3

Due: July 10, 2023, 11:55 pm

In this part of the project, you will be implementing a simple file system that is based upon a tree structure. Besides a vector of shared pointers to children nodes inherited from a TreeNode object, a Directory or File object contains a name (a File object currently inherits the vector, but it should remain empty). The directory-tree structure is a simple familiar example of a tree, and you will need the completed basic tree structure for Part 4 in order to build an Abstract Syntax Tree using the classes – Token class and classes derived from it – from Part 2.

Consider the following structure of directories and files:

```
test_directory
│   mainFile.txt
│
├───Books
│       textfile.txt
│
├───Lectures
│   │   bitmap.bmp
│   │   spreadsheet.xlsx
│   ├───Course1
│   └───Course2
│
├───Music
│       songs.zip
│
├───Notes
│   │   first.txt
│   └───one
│
└───Video
```

Fill in the missing TreeNode, Directory, and File (etc.) functionality. Please look for comments of the form "***** … *****" to know where to fill in missing functionality. For example you might see "***** Complete this member function *****".

This code introduces the notion of function pointers. For example, member function count_traverse in class TreeNode calls member function traverse_children_post_order and passes in a pointer to itself (count_traverse) and also a pointer to the count_action function. As shown in function print_traverse in class Directory, with a bit of effort it is also possible to pass to traverse_children_post_order pointers to member functions from a derived class.

Using indirect recursion, member function print_traverse traverses the nodes in a tree and, using member function print_action, prints out the information about the directories and files. As distributed, print_traverse traverses all nodes in post order fashion. Modify print_traverse so that it obeys the following rules:

a. If a given node has two children, then traverse at that point using an in-order traversal (i.e. given node processed in-between processing the children's subtrees)

b. If the node has more than two children, then traverse at that point using a pre-order traversal (i.e. given node processed before processing the children's subtrees)

c. If the node has less than two children, then use post-order traversal at that point

You can search on the web for pre-order, in-order and post-order traversal. One useful resource is https://en.wikipedia.org/wiki/Tree_traversal

For the tree shown above, the output of traversing the nodes all in post-order fashion should look as follows:

```
     textfile.txt      F
          Books        | 1
      bitmap.bmp       F
        Course1        | 0
        Course2        | 0
spreadsheet.xlsx  F
       Lectures        | 4
     mainFile.txt      F
       songs.zip       F
          Music        | 1
        first.txt      F
            One        | 0
          Notes        | 2
          Video        | 0
    testDirectory      | 6
```

Note: We are providing you the directory named “testDirectory” in the template project, which contains the directory structure explained above. You are free to copy those directories and expand on them in order to do further testing for yourself. Do keep a copy of “testDirectory” as-is, however, as we may mark your work using the provided “testDirectory” and therefore you might want to check that your code still works with that directory structure before you submit.

Instructions for submitting your code:

- Do not modify the names of the .hpp files given to you. For this part of the project, you will be submitting the files FileSystem.hpp and TreeNode.hpp. Also, do not modify the stream output to the stream given by macro OUT, as we will likely be using that for marking.