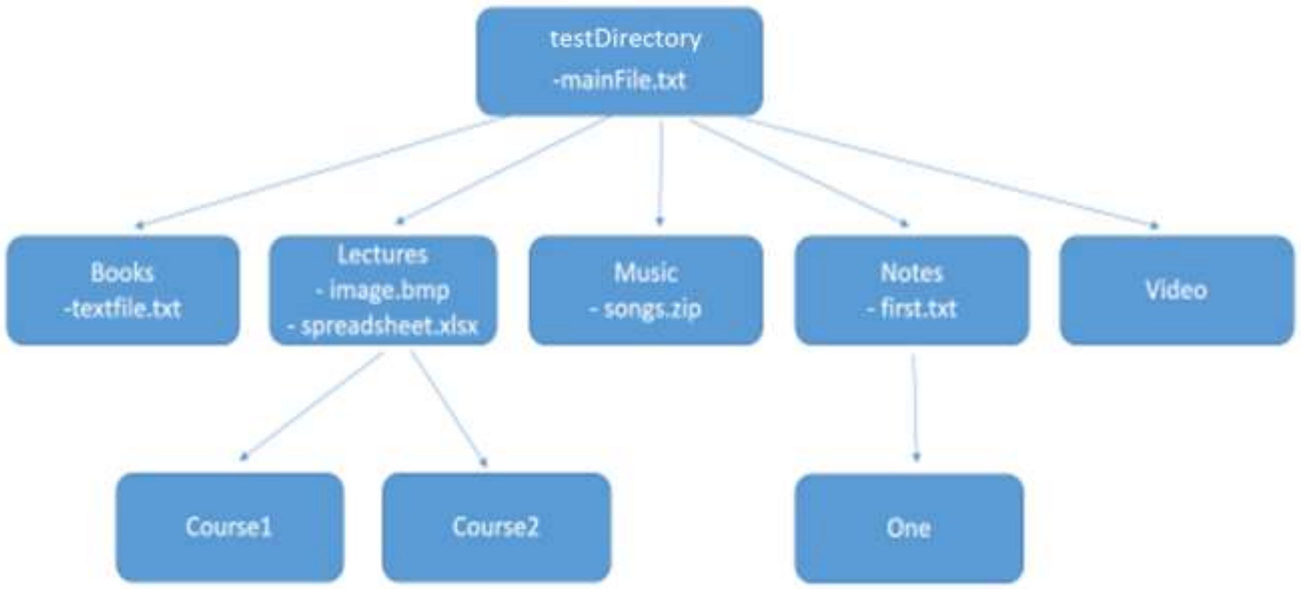
ENSC 251 – Summer 2017 – Course Project Part 3

In this part of the project, you will be implementing a basic directory-tree structure that is derived from a base tree structure. Besides a vector of children nodes, a directory-tree node contains a directory name along with names of files in the directory. The directory-tree structure is a simple familiar example of a tree, and you will need the completed base 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.

If we have following structure of directories and files:

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

Then the generated tree for that will be …



where each directory is treated as a node of the tree and the files of that directory can be stored in a vector of strings in that node.

Fill in the missing tree 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 the tree and, using member function print_action, prints out the information about the directories. 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 the children using an in-order traversal (i.e. given node processed in-between the children's subtrees)

b. If the node has more than two children, then traverse the children using a post-order traversal
c. If the node has less than two children, then use pre-order traversal at that point (i.e. children processed after the given node)

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:

```
          Books  | 0    -textfile.txt
        Course1  | 0
        Course2  | 0
       Lectures  | 2    -bitmap.bmp  -spreadsheet.xlsx
          Music  | 0    -songs.zip
            One  | 0
          Notes  | 1    -first.txt
          Video  | 0
 testDirectory   | 5    -mainFile.txt
```

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 .cpp and .hpp files given to you. For this part of the project, you will be submitting the files Directory.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.