This lab (Lab 21.1) involves binary trees. It includes the **`IntBinaryTree`** class from the textbook, with several enhancements to show the internal data, and a few new member functions. The "starter code" program works, but the '**D**' and '**H**' commands are incomplete.

The primary purpose of this lab is to become familiar with the operation of a binary tree, and its representation in memory.

Due Date:

You must demonstrate the solution to this lab exercise to the instructor by **Wednesday, December 11**, in order to receive full credit for this work.

Lab 21.1 Starter Code

Download the ZIP file from Moodle.

Extract the files and add the **.cpp** and **.h** files to your Visual C++ Project.

This is a typical command-loop lab program, which creates an **IntBinaryTree** object. It also allows the user to add nodes, display the tree, and remove nodes by means of the following interactive commands:

- the '**C**' (upper-case 'C') removes ALL nodes from the tree ("Clobber").
- the '**d**' (lower-case 'd') command displays the binary tree nodes by INorder traversal
- the '**D**' (upper-case 'D') command implements an enhanced version of the tree display (lab exercise)
- the '**h**' command prints "help" text.
- the '**H**' (upper-case 'H') command displays the tree height (lab exercise).
- the '**i**' command allows the user to insert a node into the tree, with a user-specified value
- the '**n**' command displays the number of nodes in the tree.
- the '**post**' command displays the binary tree nodes by POSTorder traversal.
- the '**pre**' command displays the binary tree nodes by PREorder traversal.
- the '**q**' command exits the program.
- the '**r**' command first deletes any tree which is already loaded, and then opens an input (text) file and reads the contents into the binary tree.
- the '**R**' (upper-case 'R') command removes a selected node from the tree.
- the '**s**' command searches the tree for a user-specified value
- the '**v**' command sets or clears VERBOSE mode.

Lab Exercise

1. Compile and run the program. Verify that you can create a binary tree, by using the '**r**' command. (Several sample text files are provided: `tree01.txt`, `tree02.txt`, and `tree03.txt`.)
2. Use the'**d**' command to display the contents of the tree. Visually trace your way through the tree by following the node addresses, beginning with the value of root, and following the left and right link values. (Draw the tree on paper, if that is helpful to see its organization.)

3. Also try the "**pre**" and "**post**" commands, to observe the difference between the different traversal methods.
4. Try adding one or more nodes to the tree, using the '**i**' command.
5. Use the '**n**' command to display the number of nodes present.
6. Use the '**R**' command to remove nodes from the tree, and verify the result with the '**d**' command. Be sure to try removing some "leaf" nodes as well as nodes with one and two "child" nodes.

Programming Exercise

7. Complete the implementation of the '**H**' command (Tree Height) by filling in the missing code in the **`IntBinaryTree::calculateSubTreeHeight`** function. The '**H**' command should return the maximum tree height in the same way that the '**n**' command returns the number of nodes in the tree. (Recall the definition of *Tree Height*, which we discussed in class.)
   HINT: use the **`IntBinaryTree::calculateNumberOfNodes`** function as a guide.

8. Implement the '**D**' command by completing the **`IntBinaryTree::enhancedTreeDisplay`** function. This command should display the tree contents like the '**d**', '**pre**', and '**post**' commands do, but adding the current node depth (within the tree) of each node to the output. The display order ("INorder", "PREorder", or "POSTorder") is your choice. An example of the possible output is shown in the table below.

   HINT: You may want to implement a new version of the **`displayNode`** function. This new, overloaded function could have a function prototype similar to the example below:

```
IntBinaryTree::displayNode(TreeNode *nodePtr, int depth) const;
```

Enhanced INorder display of binary tree (includes depth of each node at the left)

```
Command: r
Enter name of input file: tree01.txt
5 lines read from input file.

Command: D
root=0062B7F0
( 2) node=0062B880 value=   7 left=null     right=null
( 1) node=0062B838 value=  19 left=0062B880 right=null
( 0) node=0062B7F0 value=  31 left=0062B838 right=0062B8C8
( 2) node=0062B910 value=  43 left=null     right=null
( 1) node=0062B8C8 value=  59 left=0062B910 right=null

Command:
```