Students:

This content is controlled by your instructor, and is not zyBooks content. Direct questions or concerns about this content to your instructor. If you have any technical issues with the zyLab submission system, use the **Trouble with lab** button at the bottom of the lab.

7.11 Programming Assignment 3 - Binary Search Tree (BST, Summer 21 B)

Due Tuesday August 17, 2021 at 11:59 PM

Collaboration Policy Reminder

You may not use code from any source (another student, a book, online, etc.) within your solution to this PROGRAM. In fact, you may not even look at another student's solution or partial solution to this PROGRAM to use as a guide or otherwise. You also may not allow another student to look at any part of your solution to this assignment. You should get help on this assignment by coming to the instructors' office hours or by posting questions on Slack. Do not post your code on Slack.

Problem Definition

You will be implementing a binary search tree. You must use a node-based implementation of a tree. Each node in the tree will be represented by a Node class and should have a left and right subtree pointer. Each node in your tree should hold a string (a string can be more than one word) and also contain an integer called count. Include the following functions for the tree. For each operation, I have included the function prototype that you must use so that your tree will interface with the main test file. This is by no means an exhaustive list of the functions that you will be writing.

Tree class: public interface

void insert (string) - Insert an item into the binary search tree. Be sure to keep the binary search tree properties. When an item is first inserted into the tree the count should be set to 1. When adding a duplicate string (case sensitive), rather than adding another node, the count variable should just be incremented.

bool search (string) - Search for a string in the binary search tree. It should return true if the string is in the tree, and false otherwise.

int height (string) - Compute and return the height of a particular string in the tree. The height of a leaf node is 0 (count the number of edges on the longest path). Return -1 if the string does not exist.

void remove (string) - Remove a specified string from the tree. Be sure to maintain all binary search tree properties. If removing a node with a count greater than 1, just decrement the count, otherwise, if the count is simply 1, remove the node. You MUST follow the remove algorithm discussed in class or else your program will not pass the test functions. When removing, if removing a leaf node, simply remove the leaf. Otherwise, if the node to remove has a left child, replace the node to remove with the largest string value that is smaller than the current string to remove (i.e. find the largest value in the left subtree of the node to remove). If the node has no left child, replace the node to remove with the smallest value larger than the current string to remove (i.e. find the smallest value in the right subtree of the node to remove).

Tree class: public interface (traversals)

When printing a node, print the string followed by the count in parentheses followed by a , and 1 space. Follow these guidelines exactly. For example: goodbye(1), Hello World(3),

void preOrder ( ) - Traverse and print the tree in preorder notation following the printing guidelines specified above.

void inOrder ( ) - Traverse and print the tree in inorder notation following the printing guidelines specified above.

void postOrder ( ) - Traverse and print the tree in postorder notation following the printing guidelines specified above.

Required (Recursion)

All of the above functions must be implemented recursively. You will lose points is you do not do these recursively. NOTE: inOrder, preOrder, postOrder, search, and remove functions may require you to overload 1 or more of the functions above. For instance, preOrder is called from main but is not passed any parameter (you should not be able to pass the root from main because it should be a private variable in your tree class). However, you can overload the preOrder function so that it will operate recursively.

For example, your preOrder function should look like this:

```
void Tree::preOrder( ) {


```

Tree.h

```
#ifndef __TREE_H
#define __TREE_H

#include "Node.h"

using namespace std;

class Tree {

private:
  Node *root;

private:
  void preOrder( Node * );

public:
  void insert( const string & );
  bool search( const string & );
  void inOrder( );
  void postOrder( );
  void preOrder( );
  string largest( );
  string smallest( );
  int height( const string & );
  void remove( const string & );

  // Add any additional variables/functions here

};
```

```
  string str;
  int count;

public:
  Node* left;
  Node* right;

public:
  Node ( string s ) { left = right = NULL; count = 1; str = s; };
  Node ( ) { left = right = NULL; count = 0; };
  ~Node ( );

  // Add any additional variables/functions here

};

#endif
```

Node.cpp

```
#include "Node.h"
#include <iostream>

using namespace std;

Node::~Node ( ) {
```

Initial Testing

There are two sample main.cpp files below (main_6_tests.cpp and main_8_tests.cpp) with corresponding output (main_6_tests.out and main_8_tests.out). Use them to get your program up and running. Further tests (published later on zyBooks) will be based on and extend these. Final tests will test your program more exhaustively.

main_6_tests.cpp

```
#include <iostream>
#include "Tree.h"

using namespace std;

void printOrders( Tree *tree ) {
  cout << "Preorder = ";
  tree->preOrder( );
  cout << "Inorder = ";
  tree->inOrder( );
  cout << "Postorder = ";
  tree->postOrder( );
}
```

```
  cout << tree.largest( ) << endl;
  cout <<
"-------------------------------------------------------------------"
       << endl;
  cout << "Test 5: Testing subtree heights" << endl;
  cout << "Height of subtree rooted at \"neo\" = "
       << tree.height( "neo" ) << endl;
  cout <<
"-------------------------------------------------------------------"
       << endl;
  cout << "Test 6: Remove testing" << endl;
  cout << "Removing \"neo\"" << endl;
  tree.remove( "neo" );
  printOrders( &tree );
  cout <<
"-------------------------------------------------------------------"
```

```
#include <iostream>
#include "Tree.h"

using namespace std;

void printOrders( Tree *tree ) {
  cout << "Preorder = ";
  tree->preOrder( );
  cout << "Inorder = ";
  tree->inOrder( );
```

```
  cout << "Test 4: Smallest value in the tree is..." << endl;
  cout << tree.smallest( ) << endl;
  cout <<
"--------------------------------------------------------------------"
       << endl;
  cout << "Test 5: Largest value in the tree is..." << endl;
```