Coding 05: Binary Search Tree

Description: In this assignment you will create a Binary Search Tree class and object that will work with a similar struct Data type from the previous assignments (int for id and string data).

Requirements: Create a Binary Search Tree class as discussed in class and in your text. The class will contain all the data and methods to have a complete working and proper Binary Search Tree object.

- You are given ***main.cpp*** and ***main.h***. **Do not modify these**. If you feel you need to modify them, ask first.
- You are given ***functions.cpp*** and ***functions.h***. You may modify these as needed.
- You will write ***bintree.cpp*** and ***bintree.h***. This will be your binary search tree class.
- Your BST must be completely self contained and fully functional.
- The BST object may not print to the console except for the display functions.
- Your BST will be a collection of node pointers, and left and right pointers. i.e. The tree is an ordered "list" of pointers to structs.
- Your tree has to be capable of growing to any size.
- You must create and delete your nodes inside the tree (as in previous structures).
- Testing has been written for you in main(). Review this code carefully to understand the proper level of testing required.
- Your only class variables are ***DataNode *root*** and ***int count***. **Do not** make any other class wide variables.
- The root pointer may not be accessed outside the class. You may not pass it in or out of the class. In other words, the address of root is private and completely inaccessible outside the class.
- Your Tree must have the following functionality (public methods). Ones in green do not have private overloads. Ones in blue will probably need private overloads (unless you figure out a better way to do it). This will be explained in class.
  - ***BinTree();*** This is the constructor. Set count=0; and root=NULL; here. Do not initialize those in your .h file.
  - ***~BinTree();*** This is the destructor, call clear() from inside this function to destroy the tree as the object is deleted.
  - ***bool isEmpty();*** Return true or false depending on ***count == 0*** (i.e. ***!coun***t)
  - ***int getCount();*** Return a count of the nodes. The variable count should be updated as you go. Do not calculate it each time getCount() is called but rather increment and decrement based on the add and remove functions.
  - ***bool getRootData(Data*);*** If the root exists, fill the Data struct with the root data and return true. If the root doesn't exist, fill the Data struct with -1 and an empty string and return false.
  - ***void clear();*** Empties the tree. Make sure to delete all allocated data. This essentially re-sets the tree.
  - ***bool addNode(int, string);*** Add a node in-order based on id. Pass in an id and string, allocate the DataNode, and put it in the tree in order.
  - ***bool removeNode(int);*** Remove a node and re-order the tree. This method is the most complex. You should do it last.
  - ***bool getNode(Data*, int);*** Pass in an id. Fill the data to the Data struct if the node exists and return true. If it does not exist, place -1 and an empty string in the Data struct and return false.
  - ***bool contains(int);*** Return true or false if a node exists or not.
  - ***int getHeight();*** Return the height of the tree. You have to calculate this on the fly for each call to getHeight().

- ○ ***void displayPreOrder();*** Print out the ids and strings pre-order.
- ○ ***void displayPostOrder();*** Print out the ids and strings post-order.
- ○ ***void displayInOrder();*** Print out the ids and strings in-order.
- ○ ***void displayTree();*** Show the tree as shown in the example. You **must** call ***isEmpty()***, ***getHeight(), getCount(), displayPreOrder(), displayPreOrder(), displayPreOrder()*** from this function.

- You will have several private methods. Probably at least one for each item in blue above, and probably some helper functions like max() and minValueNode(). This will be explained in class.
- All good programming practices, proper architecture, and submission guidelines apply.

Example Output

Binary Search Tree created

DISPLAY TREE
==
Tree is empty
Height 0
Node count: 0

Pre-Order Traversal

In-Order Traversal

Post-Order Traversal
==

Testing removeNode() on empty tree
==
removing 10... failed

Testing getRootData() on empty tree
==
NOT retrieved -1

Testing contains() and getNode() on empty tree
==
dose NOT contain 61
NOT found: 61

Filling Tree
==
adding 60...added
the height of the tree is 1

adding 20...added
the height of the tree is 2

adding 70...added
the height of the tree is 2

adding 40...added
the height of the tree is 3

adding 10...added
the height of the tree is 3

adding 50...added
the height of the tree is 4

adding 30...added
the height of the tree is 4

DISPLAY TREE
===
Tree is NOT empty
Height 4
Node count: 7

Pre-Order Traversal
60 sixty
20 twenty
10 ten
40 forty
30 thirty
50 fifty
70 seventy

In-Order Traversal
10 ten
20 twenty
30 thirty
40 forty
50 fifty
60 sixty
70 seventy

Post-Order Traversal
10 ten
30 thirty
50 fifty
40 forty
20 twenty
70 seventy
60 sixty

==

Testing getRootData() on non-empty tree
==
retrieved 60 sixty

Testing contains() randomly
==
contains 20
contains 70
contains 20
contains 60
contains 70
contains 40
contains 60
dose NOT contain 5
dose NOT contain 32
dose NOT contain 5219
==

Testing getNode() randomly
==
retrieved: 40 forty
retrieved: 60 sixty
retrieved: 70 seventy
NOT found: 1
NOT found: 1000

Testing removeNode() randomly
==
removing 40... removed
removing root 60... removed
removing 30... removed
removing 35... failed

DISPLAY TREE
==
Tree is NOT empty
Height 3
Node count: 4

Pre-Order Traversal
70 sixty
20 twenty
10 ten
50 forty

In-Order Traversal

10 ten
20 twenty
50 forty
70 sixty

Post-Order Traversal
10 ten
50 forty
20 twenty
70 sixty
==

adding 35... added

DISPLAY TREE
==
Tree is NOT empty
Height 4
Node count: 5

Pre-Order Traversal
70 sixty
20 twenty
10 ten
50 forty
35 thirty five

In-Order Traversal
10 ten
20 twenty
35 thirty five
50 forty
70 sixty

Post-Order Traversal
10 ten
35 thirty five
50 forty
20 twenty
70 sixty
==

Clearing tree... Cleared

DISPLAY TREE
==

Tree is empty
Height 0
Node count: 0

Pre-Order Traversal

In-Order Traversal

Post-Order Traversal
==

Filling tree with poorly chosen data
==
adding 5...added
the height of the tree is 1

adding 15...added
the height of the tree is 2

adding 25...added
the height of the tree is 3

adding 35...added
the height of the tree is 4

adding 45...added
the height of the tree is 5

adding 55...added
the height of the tree is 6

DISPLAY TREE
==
Tree is NOT empty
Height 6
Node count: 6

Pre-Order Traversal
5 five
15 fifteen
25 twenty five
35 thirty five
45 forty five
55 fifty five

In-Order Traversal

5 five
15 fifteen
25 twenty five
35 thirty five
45 forty five
55 fifty five

Post-Order Traversal
55 fifty five
45 forty five
35 thirty five
25 twenty five
15 fifteen
5 five
===