

## lab 15 Binary Trees part 2

---

**Instructions:** This lab continues our construction of Binary Trees. For this lab extend your previous implementation of Binary Search Tree with contains, delete, remove, existsInRange, and countInRange.

```
1 #ifndef BINARY_TREE_H
2 #define BINARY_TREE_H
3
4 #include <string>
5
6 template<class T>
7 class BinaryTreeNode {
8     public:
9         BinaryTreeNode<T> () {
10             }
11 };
12
13 template<class T>
14 class BinaryTree {
15     private:
16         /* You fill in private member data. */
17
18         /* Recommended, but not necessary helper function. */
19         void put(BinaryTreeNode<T> *rover, BinaryTreeNode<T> *newNode);
20         /* Recommended, but not necessary helper function. */
21         std::string inorderString(BinaryTreeNode<T> *node, std::string &ret);
22     public:
23
24         /* Creates an empty binary tree. */
25         BinaryTree();
26
27         /* Does a deep copy of the tree. */
28         BinaryTree(const BinaryTree<T> &tree);
29
30         /* Add a given value to the Binary Tree.
31          * Must maintain ordering!
32          */
33         void put(const T &val);
34
35         /* Returns the height of the binary tree. */
36         int getHeight();
37
38         /* Returns true if an item exists in the Binary Tree */
39         bool contains(const T &val) const;
```

```

40
41     /* Removes a specific val from the Binary Tree.
42     * Returns true if the value exists (and was removed.)
43     * Otherwise, returns false.
44     */
45     bool remove(const T &val);
46
47     /* This method returns true iff there is a value in the tree
48     * >= min and <= max. In other words, it returns true if there
49     * is an item in the tree in the range [min, max]
50     */
51     bool existsInRange(T min, T max) const;
52
53     /* This is similar but it returns the number of items in the range. */
54     int countInRange(T min, T max) const;
55
56     /* Returns a string representation of the binary Tree in order. */
57     std::string inorderString();
58
59     /* Returns a string representation of the binary Tree pre order. */
60     std::string preorderString();
61
62     /* Returns a string representation of the binary Tree pre order. */
63     std::string postorderString();
64
65     /* Does an inorder traversal of the Binary Search Tree calling
66     * visit on each node.
67     */
68     void inorderTraversal(void (*visit) (T &item)) const;
69
70     /* Always free memory. */
71     ~BinaryTree();
72 };
73
74 /* Since BinaryTree is templated, we include the .cpp.
75 * Templated classes are not implemented until utilized (or explicitly
76 * declared.)
77 */
78 #include "binarytree.cpp"
79
80 #endif

```

### Write some test cases:

Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

### Memory Management:

Now that are using new, we must ensure that there is a corresponding delete to free the memory.

Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

**How to turn in:**

Turn in via GitHub. Ensure the file(s) are in your directory and then:

- \$ git add <files>
- \$ git commit
- \$ git push

**Due Date:** October 14, 2020 2359

**Teamwork:** No teamwork, your work must be your own.