**lab 14 Binary Trees part 1**

---

**Instructions:** This lab begins our construction of Binary Trees. It is very important to get these basics down before proceeding to further functionality of Trees. Please implement empty constructor, copy constructor, put, inorderString, lowest common ancestor (LCA), and a deconstructor.

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

```
        std::string inorderString();

        /* Return the lowest common ancestor (LCA) of two values.
         * The LCA is the most immediate parent of both values. For example:
         * 4
         * / \
         * 2 8
         * / \ / \
         * 1 3 6 10
         * LCA(1, 3) = 2
         * LCA(1, 2) = 2
         * LCA(1, 6) = 4
         */
        T& lca(T& a, T& b);

        /* Always free memory. */
        ~BinaryTree();
};

/* Since BinaryTree is templated, we include the .cpp.
 * Templated classes are not implemented until utilized (or explicitly
 * declared.)
 */
#include "binarytree.cpp"

#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 25, 2017 2359

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