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.

```cpp
#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include <string>

template<class T>
class BinaryTreeNode {
    public:
        BinaryTreeNode<T> () {
        }
};

template<class T>
class BinaryTree {
    private:
        /* You fill in private member data. */

        /* Recommended, but not necessary helper function. */
        void put(BinaryTreeNode<T> *rover, BinaryTreeNode<T> *newNode);
        /* Recommended, but not necessary helper function. */
        std::string inorderString(BinaryTreeNode<T> *node, std::string &ret);
    public:

        /* Creates an empty binary tree. */
        BinaryTree();

        /* Does a deep copy of the tree. */
        BinaryTree(const BinaryTree<T> &tree);

        /* Add a given value to the Binary Tree.
         * Must maintain ordering!
         */
        void put(const T &val);

        /* Returns the height of the binary tree. */
        int getHeight();

        /* Returns true if an item exists in the Binary Tree */
        bool contains(const T &val) const;

        /* Removes a specific val from the Binary Tree.
         * Returns true if the value exists (and was removed.)
         * Otherwise, returns false.
         */
        bool remove(const T &val);

        /* This method returns true iff there is a value in the tree
         * >= min and <= max. In other words, it returns true if there
         * is an item in the tree in the range [min, max]
         */
        bool existsInRange(T min, T max) const;

        /* This is similar but it returns the number of items in the range. */
        int countInRange(T min, T max) const;

        /* Returns a string representation of the binary Tree in order. */
        std::string inorderString();

        /* Returns a string representation of the binary Tree pre order. */
        std::string preorderString();

        /* Returns a string representation of the binary Tree pre order. */
        std::string postorderString();

        /* Does an inorder traversal of the Binary Search Tree calling
         * visit on each node.
         */
        void inorderTraversal(void (*visit) (T &item)) const;

        /* 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 30, 2019 2359

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