Lab 7 - Deleting a BST

Due Date: 5:00 p.m., April 6, 2015
All function interfaces are suggested naming and parameter guidelines. If you feel there is a better way, you are free to alter names, functions interfaces, etc, as long as you follow the lab and style guidelines. Output should match exactly unless otherwise stated.

The goal of Lab 7 is to delete and traverse a BST.

Part A: Creating a BST Class

- Using your BST from lab 6, we are going to extend the BST with remove and traversal. Your BST must have the following public interface:
    - BSTree()
    - **BSTree(const BSTree &old_tree)**
        - Performs a deep copy of a BSTree object using preorder traversal
    - **~BSTree()**
        - Removes all allocated memory in a BSTree using postorder traversal
    - bool empty()
        - true if the tree is empty
        - false if it is not
    - bool insert(int val)
        - Returns true if the value was inserted
        - false if the value was already in the tree
    - bool find(int val)
        - true if the value is in the tree
        - false if the value is not in the tree
    - **void sortedArray(vector<int> &list)**
        - Takes a vector reference, and fills the vector with the tree values in sorted order
    - **bool remove(int num)**
        - Takes a value and removes that value from the tree if found
        - Returns true is the value was removed, false if the value is not in the tree.
- You must complete your header file and empty implementations of your public methods in lab. Make sure everything compiles before moving on.
    - Example:

```
bool removeLeaf(Node *){
        return true;
}
```

--END OF IN LAB REQUIRED WORK--

Part B: Implementing remove and traversal

- You will need to implement the following algorithms. You can implement them as separate private methods or in your public methods.
    - InOrder Traversal for sortedArray
    - PostOrder Traversal for tree deletion
    - PreOrder Traversal for tree deep copy
- You should not have any additional public methods than those defined in the public interface in Part A.

Part C : Code Organization and Submission

- Required code organization:
    - lab7.cpp
    - BSTree.cpp/.h
    - makefile
        - executable should be called: lab7
            - *do not add a .exe extension*
- While inside your lab 7 folder, create a zip archive with the following command
    - zip -r lab7 *
        - This creates an archive of all file and folders in the current directory called lab7.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under ‘Lab 7’

Expected Interface and Test Output

Driver Test Commands

- Use the following driver code to test your BST
    - lab7.cpp

Grading Guidelines

- **Part A (2 points)**
    - Contains all (and only) public methods as defined by the interface in part A.
- **Part B (10 points)**
    - 1 point for each test

- **Part C (1 point)**
    - Follows formatting guidelines, requested project structure and naming conventions, contains written Readme, and submission does not include .o files or binary

Formatting Guidelines

- Stores all values in a named variable.
    - No Magic Numbers.
- Uses indentation to identify code blocks.
    - Every Code block should be indented from it's parent block to identify scope.
- No single letter or non-descriptive variable names
    - The only exception to this rule is 'i' in a for loop
- Separates code blocks and logical sections with whitespace
    - Optimize your code for the reader, not the writer
- Output is formatted with an explanation of the output values
    - Format your output so that someone who does not know what the program is supposed to do would know what the output meant
- Each method is preceded by a comment explaining what the method does
- Each significant code block is preceded by a comment explaining what the code block does.
    - A significant code block is more than 3 lines performing a single logical operation
- CONSTANTS are in all caps
- Only data types start with a capital letter
    - Classes, Enums, Structs, etc.
- Do not use the 'using namespace' declaration in a header (.h) file
- In general we will follow the Google C++ style guidelines. If you want more info, you can view them here: https://google.github.io/styleguide/cppguide.html