

**CSC1310: LAB 8**

Binary Search Tree

# BST Class

You will have a header file (**BST.h**) that is a binary tree of integers. The class definition is provided for you, and you will need to implement all of the function prototypes using recursive algorithms. Each of the public member functions operate by calling one of the private helper functions (except the default constructor). The helper functions demonstrate the use of the three types of traversal through a binary tree: inorder, preorder, and postorder.

The copy function is used by the copy constructor, and is used to recursively make a copy of another tree. It takes in a node pointer that is passed by reference and a pointer to a node. The first parameter is used to add new nodes to the tree and keeps track of the traversal position in the recursive traversal. The second parameter is a reference to the tree being copied (sent as an argument to the copy constructor), it is used to traverse that tree and send the data to the new tree being created. Creating a tree must start with the root node and move top-down, so copy needs to be a preorder traversal algorithm. It should create a new node from the second parameter and call insert (**do not** use root as an argument to call insert here. Think about what other node you can pass for more efficiency), then do a left and right traversal for both parameters.

The insert function inserts a new node into the tree, and is called by the insertNode function *and* by the copy function. Because it’s being used in different ways by two different functions (remember, you’re not passing root from its call in the copy function, but you have no choice but to start at root from insertNode) it demonstrates the benefits of using encapsulation principles. Insert should either insert the new node at the current positions, do a left traversal, or do a right traversal.

The displayInOrder function displays all of the data in the tree in sorted order. To access data in sorted order it needs to travers the tree from left to right, so it does a recursive left traversal, then prints the node, then does a right traversal.

The destroySubTree function recursively deletes the node indicated by the parameter and every successor node under it in the tree. To avoid losing references to data the leaf nodes need to be deleted first, moving bottom-up through the tree, which means it needs to use a postorder traversal.

# DRIVER – Lab8.cpp

The driver program (**Lab8.cpp**) is provided for you. It initializes two trees, the first is populated by ten randomly generated numbers from 0-99 and the second is a copy of the first tree, made by calling the copy constructor. It prints both trees to show that they should match. Three random numbers are inserted into tree 2 and both trees are printed again.

# Sample Output

adding values: 60 89 85 45 94 99 40 28 76 37

display tree 1: 28 37 40 45 60 76 85 89 94 99

display tree 2: 28 37 40 45 60 76 85 89 94 99

adding to tree 2: 14 96 58

display tree 1 again: 28 37 40 45 60 76 85 89 94 99

display tree 2 again: 14 28 37 40 45 58 60 76 85 89 94 96 99

# What to Turn In

* BST.h
* Lab8.cpp