# Lab 07

## Resources

- Discussion slides (./lab07slides.pdf) - Copy of the presentation given by the TA
- Nemo's Discussion on Data Structures (http://www.cise.ufl.edu/~nemo/cop3503/slides/Lecture+20_data_struct.pptx) some information on Binary Search Trees, which you will be implementing
- main.cpp (./main.cpp) - I will be using this to grade your program today
- Sample output (./out.txt) - This is a sample run of a correctly working BST for reference

## Idea

This lab should help you get familiar with pointers and recursion. We will do this thro the binary search tree (BST) data structure.

## In-lab assignment

### Description

You will implement a BST class. Along with this, you will need to implement a Node c represent each element of the tree. Your payload or data carried in each node is to be integer. You tree will support the following operations: insert (an element) into the tr printing the elements in order, finding out if a given element is in the tree, and printi tree rooted at a given element. All of these functions will need to be recursive (most lik the Node class).

### Requirements/Deliverables

You will implement a standard BST. The tree will be made up of many Nodes, which y will also need to implement. Make the tree function as closely to the way described in you can. You will need to break up your implementation into .h and .cpp files as usual

*!!!!!! IT IS VERY IMPORTANT THAT YOU NAME THESE CLASSES AND FUNCTION EXACTLY AS THEY APPEAR HERE !!!!!!*

The tree class should be called BST and needs to implement the following functions (t are prototypes):

1. ```
   BST()
   ```

   This is the default constructor. It should initialize the root node to be nullptr.

2. ```
   void insert(int)
   ```

   This function inserts a given value into the tree. YOU MUST KEEP THE ORDER THE TREE. You keep tree order by placing nodes less than the root to the left and nodes greater than the root to the right. If you are given a duplicate node, you can simply ignore it (don't add it to the tree). I will not give you duplicate nodes in my testing.
   - For reference, the tree below was built by inserting nodes in this order: 2 -> 3 -> 5 -> 4

3. ```
   void print_inorder()
   ```

   This is an inorder traversal of your tree. It should print the nodes of the tree in so order. The format should be each value separated by a single space on one line. Th should be a new line at the end. Below is an example
   ```

```
  2
 / \
1   3
     \
      5
     /
    4
```

The result of the print function is:

```
1 2 3 4 5 \n
```

If the tree is empty, just print "Tree empty".

4. `bool find(int)`

   This function returns true if the given value is in the tree, false otherwise.

5. `void print_from_value(int)`

   This function searches for the value in the tree and prints the subtree rooted at th node using an inorder traversal. Taking the tree from above, calling this function passing 5 should print

   ```
   4 5 \n
   ```

   If the node is not in the tree, print a message saying "Node [whatever number wa passed] not found".

You may want to add in additional functions to accomplish intermediate taks. These functions are helper functions and should be *private*.

The class for the node should be called Node. I will not specify the functions that shou in the node class. *hint hint, the node functions should be the ones that are recursive, you will probably need functions very similar to those found in the BST class*. For instance, the BST::print_inorder function can be written as

```
void print_inorder(){
    if(root == nullptr){
        cout << "Tree empty" << endl;
        return;
    }
    root->print_inorder();
}
```

if you write the Node::print_inorder function correctly.

## Submission

You will only need to submit a BST.h and a BST.cpp file containing your implementat
the assignment Lab 07 area of elearning. Just submit them as two separate attachme
no need to zip them up. *There should not be a main() in either of these files!!!* I will w
my own main.cpp to test your BST class using the functions described. The test file I v
use to grade can be found here (./main.cpp). If you finish during lab time, I will come
and check you off on this test program. Otherwise, you have until Sunday night to tur
the program (extended from the usual Friday because of 4th festivities).

## Hints

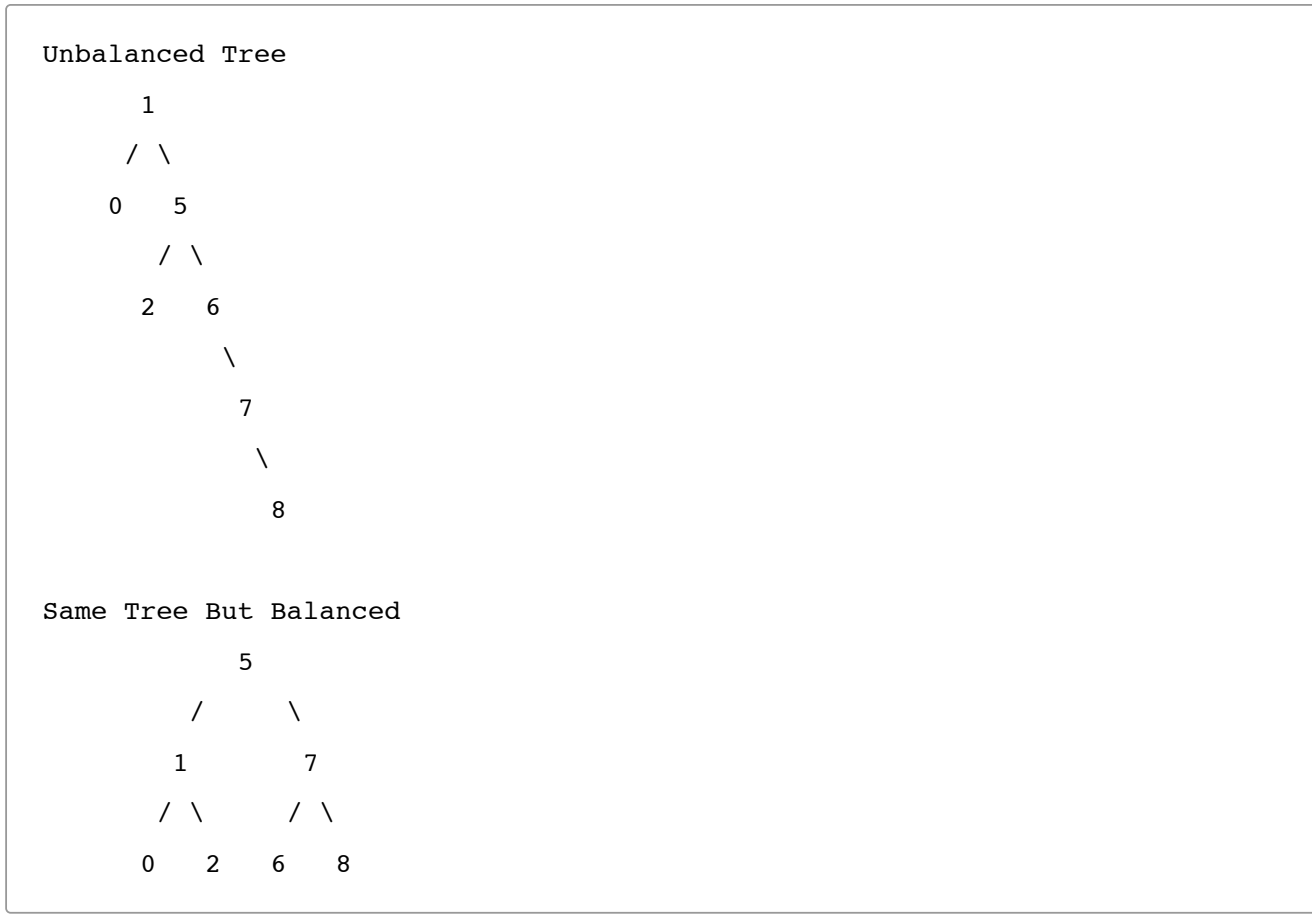
- You will probably want to write your own test harness main to test your program
  you go instead of trying to go for the one used for grading directly.
- Make sure you are initializing your pointers correctly (to nullptr or otherwise).
- You will need to make use of the new keyword for this lab. Be wary of memory lea
- Don't forget about the debugger lesson from last (last last?) week. Use it early and
  it often and you will get more and more profiicient at it.

## Grading Distribution

- 5 points for passing the tests in the given main
- 5 points for the BST class
- 5 points for the Node class. Note that most of the recursion should be in this class
- 5 points for good style

# Optional Enhancements

- This structure requires use of dynamic memory (each node needs to be created). ' leads to a lot of memory leaks when the tree is deleted. Write a destructor to avoi these memory leaks.
- Write a pre and post order traversal of the tree. The one you have already written in order traversal. It should be *very* easy to change this logic to make a pre or pos order traversal. How do these traversals differ?
- Add in functionality to delete a node from the tree. This requires a decent amoun thought. How do you delete a leaf node? How do you delete a node that has only o child? How do you delete a node that has 2 children? Each of these cases require different handling.
- There is a concept often applied to trees called *balancing*. A balanced tree is one which nodes in each level have the maximum amount of children (so ideally, each has 2 children). This avoids the tree degenerating into a linked list. First of all, wh this a bad thing? Secondly, how can you balance a tree? There are many ways. So keep the tree balanced as you add nodes, and some take an unbalanced tree and r it balanced. You can look up red/black trees or AVL trees for the former and the l Stout-Warren algorithm is a good place to get started for the latter.


Unbalanced Tree
1
0 5
2 6
7
8
Same Tree But Balanced
5
1 7
0 2 6 8