

Homework Assignment #2 – Binary Search Tree Implementation  
by Evan Olds  
Cpt S 223 – Advanced Data Structures

### **Submission Instructions:**

Submit source code (zipped) to Angel BEFORE the due date/time. If the Angel submission is not working, then submit to TA via email BEFORE the due date/time.

Optional: Include a readme.txt file in the zip with any relevant information that you want the grader to be aware of.

### **Assignment Instructions:**

**Read all the instructions *carefully* before you write any code.**

Download the zip file from Angel and open the Visual Studio 2012 project included within it. Complete the implementation of the binary search tree class functions.

1. Implement Add function in HW2BST class:

**bool** Add(**int** dataValue)

This function adds the specified value to the tree. If the tree is empty then the value becomes the root node. True is returned on success and false on failure. If the number already exists in the tree or memory cannot be allocated for a new node, then false should be returned.

2. Implement Remove function in HW2BST class:

**bool** Remove(**int** dataValue)

This function removes the specified value from the tree, if it exists. If it does exist and is removed, then true is returned. If it does not exist in the tree then false is returned and the tree is not modified.

3. Implement CountLevels function in HW2BST class:

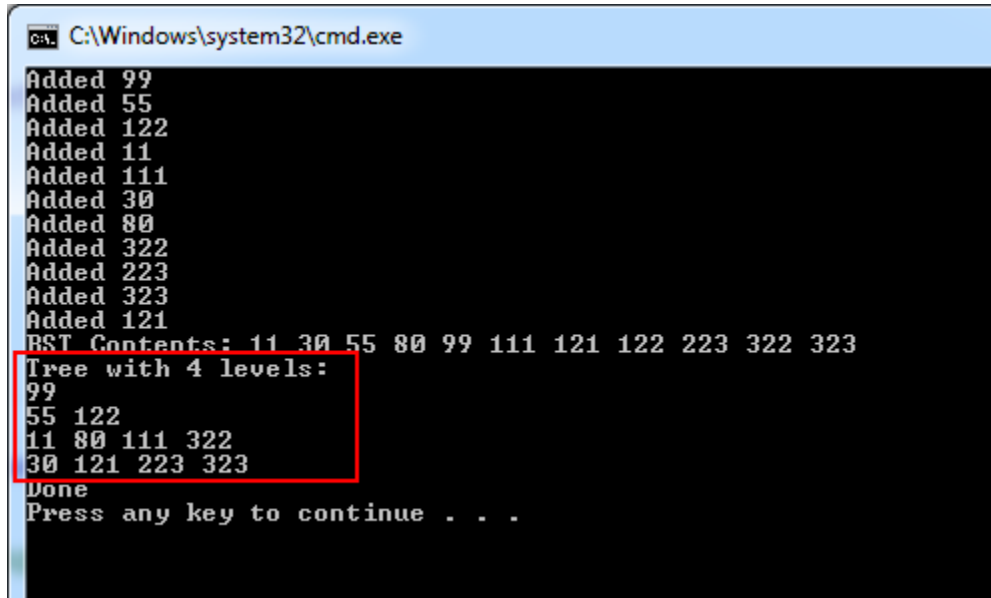
**int** CountLevels()

This function counts the number of levels in the tree and returns it. If you look at the tree visually, this is the number of rows of nodes. An empty tree (null root node) has 0 levels. A tree with a root node that has two null children has 1 level.

#### 4. Implement DisplayLevels function in HW2BST class:

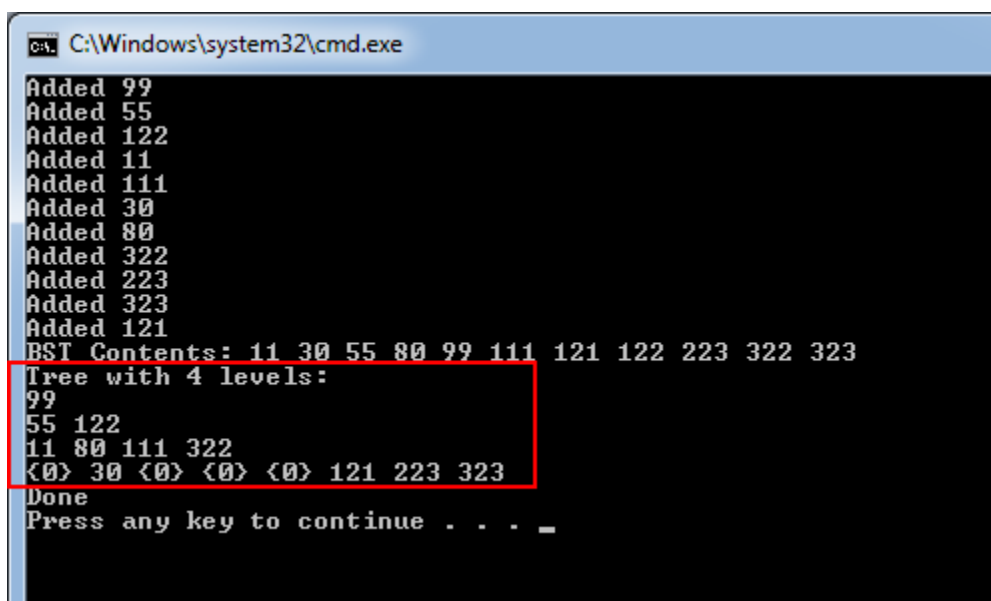
```
void DisplayLevels(std::ostream& outputStream)
```

This function displays each level of the tree on its own line. Each level/row of the tree must be displayed from left to right. You don't have to do really nice formatting on this if you don't want to, but you must correctly display all data values in each row in left to right order. At a minimum you can have something that looks like this:



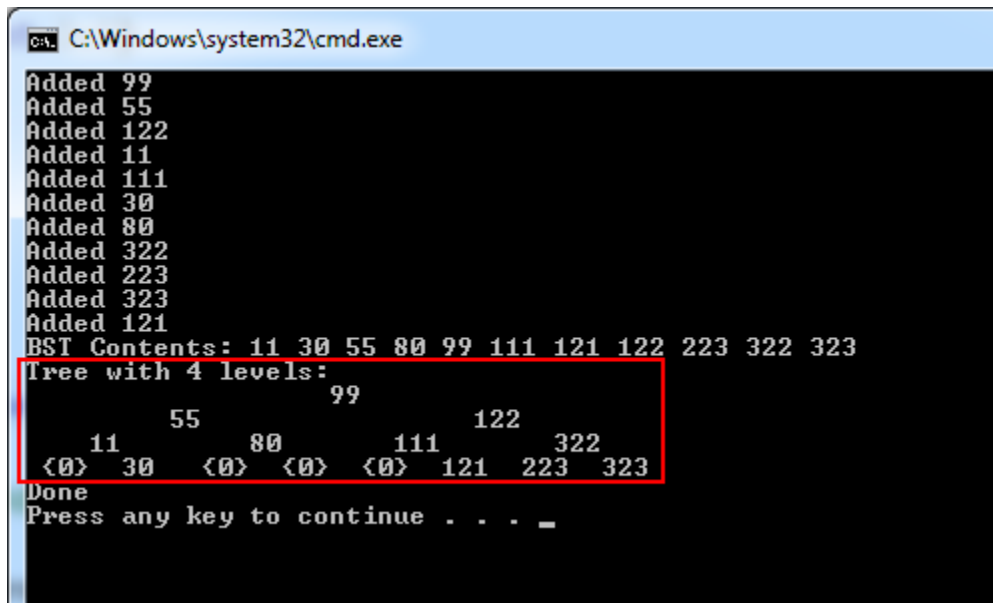
```
C:\Windows\system32\cmd.exe
Added 99
Added 55
Added 122
Added 11
Added 111
Added 30
Added 80
Added 322
Added 223
Added 323
Added 121
BST Contents: 11 30 55 80 99 111 121 122 223 322 323
Tree with 4 levels:
99
55 122
11 80 111 322
30 121 223 323
Done
Press any key to continue . . .
```

Slightly better would be if you also showed null nodes (with a special string like {0}) on each level, so it can be determined which nodes are children of their parents on the upper level. This would mean that you have always show  $2^n$  values on each level, where  $n$  is the zero-based level number:



```
C:\Windows\system32\cmd.exe
Added 99
Added 55
Added 122
Added 11
Added 111
Added 30
Added 80
Added 322
Added 223
Added 323
Added 121
BST Contents: 11 30 55 80 99 111 121 122 223 322 323
Tree with 4 levels:
99
55 122
11 80 111 322
{0} 30 {0} {0} {0} 121 223 323
Done
Press any key to continue . . .
```

The last option is to render the tree in a structured fashion. This is the most visually appealing and could potentially help with troubleshooting your tree functions, but is not recommended since there are no extra points for it and it can also be time consuming to implement. You should focus on getting the functions working correctly before tweaking any visual attributes of your application. Such an option might look something like:



```
C:\Windows\system32\cmd.exe
Added 99
Added 55
Added 122
Added 11
Added 111
Added 30
Added 80
Added 322
Added 223
Added 323
Added 121
BST Contents: 11 30 55 80 99 111 121 122 223 322 323
Tree with 4 levels:
      99
     /  \
   55    122
  /  \  /  \
11 30 80 111 322
{0} {0} {0} {0} {0} 121 223 323
Done
Press any key to continue . . . _
```

### Final notes:

There are sample input files and corresponding output files included in the homework .zip file. Make sure you use these for testing and verify that your functionality is correct.