CS14 Summer 2016
Programming Assignment 3

Binary Search Tree (BST)

Due Monday July 11 at 11:30 PM

BST

In this assignment, you will be implementing a binary search tree. You must use a node-based implementation of a tree. Each node in the tree will be represented by a Node class and should have a left and right subtree pointer. Each node in your tree should hold a string (note: a string can be more than one word) and also contain an integer called count. Include the following functions for the tree. For each operation, I have included the function prototype that you must use so that your tree will interface with the main test file. This is by no means an exhaustive list of the functions that you will be writing. **These functions must be public functions in your tree class**

- void insert (string) - Insert an item into the binary search tree. Be sure to keep the binary search tree properties. When an item is first inserted into the tree the count should be set to 1. When adding a duplicate string (case sensitive), rather than adding another node, the count variable should just be incremented.
- bool search (string) - Search for a string in the binary search tree. It should return true if the string is in the tree, and false otherwise.
- string largest ( ) - Find and return the largest value in the tree. Return an empty string if the tree is empty.
- string smallest ( ) - Find and return the smallest value in the tree. Return an empty string if the tree is empty.
- int height (string) - Compute and return the height of a particular string in the tree. The height of a leaf node is 0 (count the number of edges on the longest path). Return -1 if the string does not exist.
- void remove (string) - Remove a specified string from the tree. Be sure to maintain all binary search tree properties. If removing a node with a count greater than 1, just decrement the count, otherwise, if the count is simply 1, remove the node. You MUST follow the remove algorithm discussed in class or else your program will not pass the test functions. When removing, if removing a leaf node, simply remove the leaf. Otherwise, if the node to remove has a left child, replace the node to remove with the largest string value that is smaller than the current string to remove (i.e. find the largest value in the left subtree of the node to remove). If the node has no left child, replace the node to remove with the smallest value larger than the current string to remove (i.e. find the smallest value in the right subtree of the node to remove).
- Print the tree in the following manners. When printing a node, print the string followed by the count in parentheses followed by a , and 1 space. You must follow these guidelines exactly. For example: goodbye(1), Hello World(3),

- void preOrder ( ) - Traverse and print the tree in preorder notation following the printing guidelines specified above.
- void inOrder ( ) - Traverse and print the tree in inorder notation following the printing guidelines specified above.
- void postOrder ( ) - Traverse and print the tree in postorder notation following the printing guidelines specified above.

Note about recursion

Some of the above functions used to interface with the main test file are not conducive for recursive methodology. However, you must write the inOrder, preOrder, postOrder, search, and remove functions recursively **(you will lose points if you do not do these recursively)**. This may require you to overload 1 or more of the functions. For instance, preOrder is called from main but is not passed any parameter (you should not be able to pass the root from main because it should be a private variable of your tree class). However, you can overload the preOrder function so that it will operate recursively. For example, your preOrder function should look like this:

```
void Tree::preOrder( ) {
   preOrder( root );
}
```

and you will write the preOrder code within the recursive function that takes a node as a parameter. You may need to do something similar for the inOrder, postOrder, search, and/or remove functions so that you can write these functions recursively. Should these recursive helper functions be public or private?

Provided Skeleton Code

tree.h includes a node class that you will need to create.

Extra credit for as3

I think it is very important for you to write a function to print the tree out in this manner. For 5% extra credit, write a function to print the tree out sideways (it must LOOK exactly like my sample - we will visually check instead of doing a diff test). Here is the order the items were inserted to get the output. You must use the following function prototype: void printTreeSideways( ) and the function must be part of the tree class. The printTreeSideways function will print the tree to a file called "mytree.out." We will only test for this function if you include a readme.txt file that states that you did implement the function. If you do not follow these specifications exactly, you may not get the extra credit. You should write this function first to help you with debugging. Also, this is actually a very easy function to write if you think about it so don't pass up this easy extra credit.

Testing

We will be grading your program with a main file that looks something like this (of course we will be using a much more rigorous main to test your program). The output should match a diff test with this

For this assignment, you will need to learn how to **test and debug your own code** without the help of the test file we will be using. You will need to think about what cases to test and how to test them. To ensure that your code will compile with our main testing file, make sure that your code compiles with simple_main.cc. To ensure that you output will match the diff test, DO NOT print ANYTHING out in the tree/node class functions! You should still do all of the necessary error checking but DO NOT print out error messages.

Suggestions to follow while coding

- Follow a modular programming style. Write one function for your program and completely test it before moving to the next function. This will isolate your debugging because most of the time the bug will be in the newly created module. However, if you forget a test case in an old module and do not completely test it, a new module can reveal old bugs. Watch out for this.
- I highly suggest you write a print function to display your tree in a readable format (order traversals don't give you the actual tree structure but it does allow us to verify the functionality of your code). Do the extra credit and print the tree out sideways-->

What to turn in

You should **ALWAYS** turn in what you have thus far on our program at least **6** hours before the due date (by 5:30PM). Then continue to work on your program and turn in more current versions as you get them working.

You must turn in all .cc and .h files. In addition, you must follow any guidelines and include any information that is stated on the syllabus about at-home programming assignments. You must also turn in a readme file stating which test cases work and which do not. Your readme file can also include any notes or comments about your code that you would like to make to the grader.

DO NOT, I repeat, DO NOT turn in any files that are not required for the final compilation of your assignment. Do not turn in your temporary .cc files or your saved/old revisions of your .cc files. If you turn in files that aren't required for your assignment, it hinders the grading processes. You will be docked points if you turn in extraneous files.

A reminder about collaboration on home programming assignments

Please remember, at-home programming assignments are not lab assignments and you may not team-code with your lab partner or any other individual. Limited collaboration may be acceptable, but programs must represent YOUR OWN original work. Sharing code or team-coding are strictly not allowed. Copying code from ANY source (any book, current or past students, past solutions, INCLUDING your own, web, etc) is strictly not allowed even with citation. Collaboration may consist of discussing the general approach to solving the problem, but should not involve communicating in code or even pseudo-code. Students may help others find bugs. Your code MUST be unique -- the odds of randomly producing similar code is very low.

Computing, like surgery or driving a car or playing golf, can only be learned by doing it yourself!