

Assignment A4

Due Date: November 24 @ NOON

Purpose

The purpose of this newest, delectable project is to implement one of the often-used tree-balancing algorithms, as well as to build upon legacy code (already working code) while using many C++ features.¹

Problem

We have seen that balancing a binary search tree (BST) is key to achieving the theoretical searching performance of $O(\log_2 N)$. On the course Web page is code in C++ that implements a binary tree as well as a binary search tree. However, the implementation does not include a nice way of displaying a tree, nor does it maintain a balanced tree. You will add to the given code to add this additional functionality.

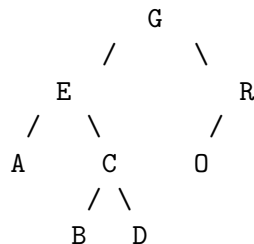
In addition, you should think of yourself as the creator of a tree library to be used by an application programmer. Thus, you are writing an API - an Application Programming Interface for various trees. The `main()` given on the course Web page is a sample of what an application programmer might do with the library. I will test your library with my own `main()`. Thus, you should **not** submit your own `main` function, but rather just the header and implementation files. Be sure to use the naming conventions described below.

Input

There is no input per se for this program. The input depends on what the application programmer does. However, you may assume that the input **type** will be in the set `{int, char, float}`.

Output

Much of the output is dependent on the application programmer. However, you will write a new method to display a nicely formatted binary tree, similar to the tree below:



For this output (only!), you need the alignment to work correctly only for input of type `char`. The tree should be nicely aligned up to a height of at least 4. Note that it is acceptable to have widely-spaced nodes at upper-levels of the tree, even if the height is only one or two.

There is some additional output for some new methods (or at least returned values), described below.

¹Yay! Back to C++!

Specifics

You must implement the following. Be sure to follow naming conventions **precisely**, or my testing of your code will fail. You should also implement these items in the following order, which is both easier and will be better for your grade.

- In `treesClass.h`, add a new method called `treeHeight()` to class `binaryTree`. This method should have no parameters and return an integer representing the height (as defined in our text and in class) of the current tree. Implement this and other methods below in `treesClass.cpp`.
- Also in class `binaryTree`, define a method called `treeDisplay()`. This method has no parameters. It should display the tree in a nicely aligned fashion as described in the Output section above. Note that the tree may not be balanced. **THIS IS NOT TRIVIAL.**
- In `treesClass.h`, define a new class called `balancedBST`. This should inherit from class `binarySearchTree`.
- In the new class above, add a method called `balanceFactors()`. This is a void method that takes no arguments. It should display the AVL balance factors of all the nodes in the tree in preorder fashion. All the values should be displayed on one line.
- Also in class `balancedBST`, implement a method called `insertItem (key)` that a) inserts *key* into its proper place in the BST and b) balances the tree. You should follow the AVL tree balancing method, which requires single- and double-rotations. **THIS IS A LOT OF WORK!**
- Before turning in, delete the `main()` function in `treesClass.cpp`. I will include your files in my own testing function(s).

Notes

- First be sure you understand the starting code. Then add in the methods/class in the order above.
- Feel free to pilfer code you find on the 'net or in texts. We don't need to reinvent the wheel, and it is also instructive to look at others' code. However, be sure to follow the requirements stated above.
- Note that you may need to implement "helper" methods to support those above. Any helper methods should be private in the class to which they belong.
- You may add **one** data item to `struct nodeType`, if necessary.
- When you get to the balancing method, first implement single rotations. If these work properly, you will get good partial credit. Adding the double rotations will get you full credit. However, be sure the single rotations work at any/all levels before going on to the double rotations. It is better to get the single rotations working **at all levels of a tree** than to get both single and double rotations working at only **some** levels.

- The name of your implementation file **must** be `treesClass.cpp` and your header file `treesClass.h`. Bundle these together in a tar and/or zip file under your last name and project number, as usual.
- Turn in the project via email to `mgousie@wheatonma.edu` before **noon** on the due date. If possible, turn in hard copy to me or under my door sometime on the 24th.
- Write/print and sign the Wheaton Honor Code Pledge on what you turn in: “I have abided by the Wheaton College Honor Code in this work.”
- Remember to save all of your work until your project is returned.

An Algorithm for the Organization of Information.

- The vague title of the paper by Adelson-Velskii and Landis describing AVL trees.