EECS 560 Lab 6: Binary Search Tree

Due date:
10/11/2020, 11:59 pm -Sunday

Objective
Get familiar with Binary Search tree implementation with C++.Work on functions on Binary search tree

Specification of the ADT

Refer to section 4.3 in Textbook. Implement the code for methods in figure 4.16 of Binary search tree class skeleton, which has Type declaration of functions of Binary search tree. The implementation for some of these methods available as mentioned below.

1. Figure 4.17 : Public member functions calling private recursive member function of Binary search tree **Contains, Insert and remove.**
2. Figure 4.18 : **Contains** private member function for binary search trees.
3. Figure 4.20 Private member function. Recursive implementation of **findMin** for binary search trees
4. Figure 4.21 Private member function. Non recursive implementation of **findMax** for binary search trees
5. Figure 4.23 **Insert** private member functions into a binary search tree
6. Figure 4.26 **Remove** private member function. Deletion routine for binary search trees
7. Figure 4.27 **Destructor** of Binary search tree and recursive **makeEmpty** private member function.
8. Figure 4.28 **Copy constructor** of Binary search tree and recursive **clone** private member function
9. In Binary search tree class Skelton, there is a method called **Print tree** and its corresponding Private member function. **Please refer to "Print In Order, Post Order, Pre Order" section** for modified print methods to be implemented and ignore print tree function.

Requirements to completed from text from template:

10. Rename the Binary search tree class name into "**myBinarySearchTree",** instead of "BinarySearchTree" as indicated in the textbook.

11. Implement **myBinarySearchTree( )**defined from textbook template. It was not implemented in textbook. Constructor with default behavior.

12. Implement **myBinarySearchTree(myBinarySearchTree && rhs)** copy constructor method defined from textbook template. It was not implemented in textbook.

13. Implement public member functions for calling private recursive member function **const Comparable & findMin( ) const** and **const Comparable & findMax( ) const** defined from textbook template

14. Implement public member functions **bool isEmpty( ) const** defined from textbook template. This method is to verify if the tree is empty or not.

15. Implement public member function for calling private recursive member function **void makeEmpty( )** defined from textbook template.

16. Implement public member function for calling private recursive member function **void void insert(Comparable && x)** defined from textbook template.

17. Implement operator functions **myBinarySearchTree & operator=(const myBinarySearchTree & rhs)** and **myBinarySearchTree & operator=(myBinarySearchTree && rhs)** defined from textbook template.

Print In Order, Post Order, Pre Order:

18. Implement a public member function **printTreeInorder** with interface **void printTreeInorder(ostream & out = cout)** to call recursive private member function **printTreeInorder** . Implement private member function **printTreeInorder** with interface **void printTreeInorder(BinaryNode *t, ostream & out) const** This should write Binary search tree nodes in **InOrder** to output file.

19. Implement a public member function **printTreePreorder** with interface **void printTreePreorder(ostream & out = cout) const** to call recursive private member function **printTreeInorder** . Implement private member function **printTreePreorder** with interface **void printTreePreorder(BinaryNode *t, ostream & out) const.** This should write Binary search tree nodes in **PreOrder** to output file.

20. Implement a public member function **printTreePostorder** with interface **void printTreePostorder(ostream & out = cout) const** to call recursive private member function **printTreePostorder** . Implement private member function **printTreePostorder** with interface **void printTreePostorder(BinaryNode *t, ostream & out) const**This should write Binary search tree nodes in **PostOrder** to output file.

Testing and Grading
We will test your implementation using a tester main function, on a number of instances that are randomly generated. We will release the tester main function, several instances (will be different from the grading instances but will have the same format), and expected output together with the lab instruction via Blackboard. You code will be compiled under Ubuntu 20.04 LTS using g++ version 9.3.0 (default) with C++11 standard.

The command line we are going to use for compiling your code is:

"g++ -std=c++11 main.cpp" (note that main.cpp will try to include the .hpp file you submit, and your .hpp file needs to be property implemented to compile successfully).

Your final score will be the percentage your program passes the grading instances. Note that if your code does not compile (together with our tester main function), you will receive 0. Therefore, it is very important that you ensure your implementation can be successfully compiled before submission.

Submission and Deadline

Please submit your implementation as two one .hpp file, with file name "myBinarySearchTree _[YourKUID].hpp". For example, if my KU ID is c123z456, my submission will be files named "myBinarySearchTree _c124z456.hpp". Submissions that do not comply with the naming specification will not be graded. Please submit through Blackboard.