CSC 212 Homework # 5
Binary Trees & Binary Search Trees
Due date: 04/12/2014

27/11/2014

Guidelines:	This is an individual assignment. Fill in this page, **print it out and use it as a cover page** to your homework. The homework **must be handwritten**. Write your name, student ID number, section number and homework number on each page as a header. Staple your homework. The parts designated **online** must be submitted electronically to Web-CAT. The homework must be handed in hard copy to your tutorial instructor.
Name:	.....
Student ID:	.....
Section:	.....

Problem 1

1. As a user of the ADT Binary Tree, write the instructions necessary to transform the tree shown in Figure 1 into the one shown in Figure 2 (let the tree be called *bt*, a variable of type *BinaryTree <String >*).

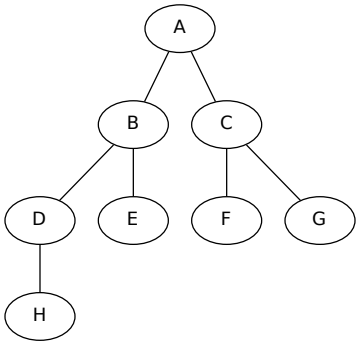

Figure 1: A binary tree (H is the left child of D).

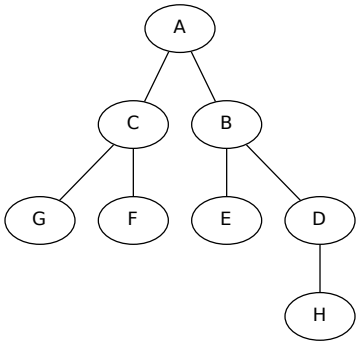

Figure 2: The mirror of the tree shown in Figure 1 (H is the right child of D).

2. Draw the tree shown in Figure 1 after calling the method *func* written below.

```
public class BT<T>{
        ...

        public void func(){

                recFunc(root,true);
        }

        private void recFunc(BTNode<T> t, boolean flag){

                if(t==null)
                        return;

                recFunc(t.left, flag);
                recFunc(t.right, !flag); // Notice the !
```

```
                if(flag){
                        BTNode<T> tmp= t.left;
                        t.left= t.right;
                        t.right= tmp;
                }
        }
}
```

. . . . . .

Problem 2

1. Write the **iterative** method *collectInOrder*, member of the class BT (binary tree) that returns a list that contains all the data in the tree in the *InOrder* order.

   The method signature is: *public List*$<T>$*collectInOrder()*.

   Example 2.1. *For the tree shown in Figure 2, the output to* collectInOrder() *is the list:* $G \to C \to F \to A \to E \to B \to H \to D$.

2. Write the **recursive** method *find*, a private member method of the class BT (binary tree) that takes as input a node t and data e and returns true if e exists in the subtree rooted at t, false otherwise. The method signature is: *private boolean find(BTNode* $<T>$*t, T e)*.

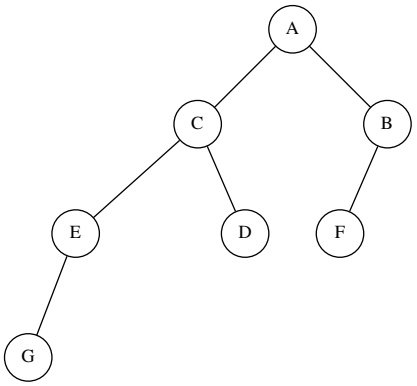

Figure 3: A binary tree.

   Example 2.2. *In the tree shown in Figure 3, the call to* find("E") *with the node containing data C as parameter returns true, whereas the call with the node containing data B as parameter returns false.*

3. Write a **recursive** method *sizeBalanced*, member of the class BT (Binary Tree), that returns true if the tree is empty, or, at every node, the absolute value of the difference between the number of nodes in the two subtrees of the node is at most 1. The method signature is: *public int sizeBalanced()* (this method calls the private recursive method *recSizeBalanced*).

   Example 2.3. *The binary tree shown shown in Figure 3 is not size balanced. The size balance at* E *is -1, at* C *is -1, at* B *is -1, but at* A*, the size balance is 2-4=-2.*

. . . . . .

Problem 3

Consider the binary search tree shown in Figure 4.

1. Draw the tree after inserting the keys: 3, 9, 7 and 4.
2. Draw the tree (obtained in Step 1 after all 4 inserts) after deleting the keys: 1, 10, 8 and 9.

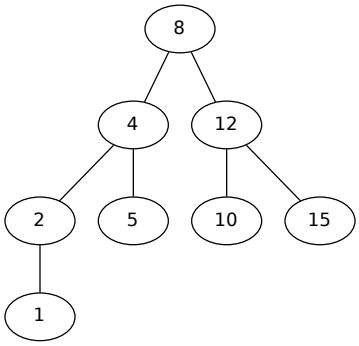

Figure 4: A binary search tree (1 is the left child of 2).

. . . . . .

Problem 4

1. Write a **recursive** version of the method *findKey*, member of the class *BST* (see the specification in the slides).
2. Write the method *removeLarger*, memebr of the class *BST* that takes as input a key k that exists in the tree and removes all the keys that are (strictly) larger than k. The method signature is *public void removeLarger(int k)*.
3. Write the method *nbLess*, member of the class *BST* that takes as input a key k and returns the number of keys in the tree that are smaller or equal k.

   The method signature is: *public int nbLess(int k)*.

   Example 4.1. *The call* nbLess(10) *on the BST shown in Figure 4 returns 6. The call* nbLess(6) *returns 4.*

. . . . . .

Problem 5 (Hard and Online)

Write the method *intervalSearch*, member of the class *BST*, that takes as input two keys k_1 and k_2 and returns a queue that contains the data of all nodes that have keys in the interval $[k_1, k_2]$. The order of the data in the queue must be the same as that of the keys.

The method signature is: *public LinkedQueue<T >intervalSearch(int k_1, int k_2)*.

Example 5.1. *For example, the call to the method* intervalSearch(5,14) *on the tree shown in Figure 5, returns the queue that contains:* $E \rightarrow A \rightarrow C \rightarrow F$.

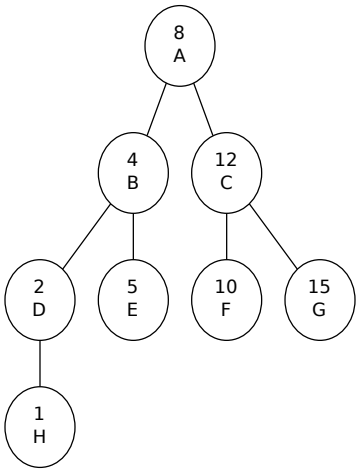

Figure 5: A binary search tree (H is the left child of D). The keys are the numbers, whereas the letters are the data.

. . . . . .