ALERT!

1. The objective of this lab is understanding binary search tree data structure.
2. This is an individual lab, you are strictly **NOT** allowed to discuss your solution with fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can **ONLY** discuss with your TAs or with me.
3. Beware of memory leaks and dangling pointers.
4. Pay attention to **GOOD coding conventions** e.g.
   - Proper indentation.
   - Meaning variable and function names.
   - Use camelCase naming convention
   - Use meaningful prompt lines/labels for all input/output

Task 01: [65 Marks]

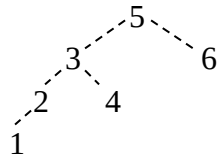
Implement the binary tree using linked implementation as we discussed in class.

```
// forward declaration of template class BTree
template<class T>
class BST;

template<class T>
struct BSTNode
{
   T  data;
   BSTNode<T>*  left;
   BSTNode<T>*  right;
     // Methods…
};

template<class T>
class BST
{
    BSTNode<T>* root;
     // Methods…
};
```

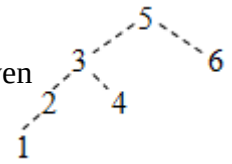
Implement following functions for BST class:

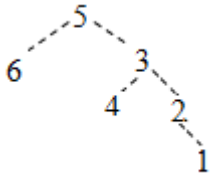
1. **Constructor, destructor, Copy-constructor.** **[2+2+2]**
2. **void setRoot (T value);** **[1]**
3. **BST Node<T>* getRoot ( );** **[1]**
4. **void insert (T value);** **[2]**
5. **BSTNode<T>* getLeftChild (BSTNode<T>* node);** **[1]**
6. **BSTNode<T>* getRightChild (BSTNode<T>* node);** **[1]**
7. **DNode<T>* search (T val)** **[2]**
8. **void deleteNode (BSTNode<T>* node)** **[4]**
9. **void printPostOrder (BSTNode<T>* root);** **[4]**
   This function should print the tree nodes in post order iteratively.
10. **boolean isBST (BSTNode<T>* root);** **[4]**
    Takes an Object of tree as a parameter and returns true if it is a BST, false otherwise.
11. **int getNodeDegree(T node);** **[2]**
    This function should return the degree of node.
12. **int getTreeDegree();/// returns the tree degree.** **[3]**
13. **void printTree ( );** **[4]**

This function should print tree in bellow form



14. **boolean isEqual (BSTNode<T>* r1, BSTNote<T>* r2);** **[4]**
    Takes roots of two trees and as input parameter and returns true if they are equal.
15. **boolean isInternalNode (BSTNote<T>* node);** **[2]**
    Returns true if given node is an internal node. Where, internal Node is one which has degree greater than zero.
16. **boolean isExternalNode (BSTNote<T>* node);** **[2]**
    Returns true if given node is an external node. External Node is one which has degree equal to zero.
17. **int getHeight ( );** **[3]**
    Returns the height of tree.
    If u want to write recursive function for height calculation then call your recursive function in this function and make it wrapper/driver of recursive function.
18. **void displayDescedents (T val);** **[3]**
    Display decedents of the node containing given value.
19. **void displayAncestors (T val);** **[3]**
    Display Ancestors of the node containing given value.
20. **BST<T> getMirrorImage ( );** **[4]**
    Returns the mirror image of *this tree.

For Example, given 

you should return 

21. **int getNodeCount (BST<T>* tree);** **[3]**
    Return the number of nodes in a given tree.
22. **T findMinMax (int flag);** **[4]**
    This function takes a single parameter i.e a flag with only 2 possible values, 0 and 1. If the input value is 1 then return the Max value stored in your BST if the value is 0 then return minimum value.
23. **T printSorted ( );** **[2]**
    This function should display tree data in sorted order.