Assignment 2

Issue Date: Wednesday – August 31st, 2022
Submission Deadline: Friday – September 9th, 2022 (Till 5:00 pm) **Marks = 50**

Instructions!

1. You are required to do this assignment on your own. Absolutely **NO** collaboration is allowed, if you face any difficulty feel free to discuss it with me.
2. Cheating will result in a **ZERO** for the assignment. (Finding solutions online is cheating, Copying someone else's solution is cheating). Also, do not hand your work over to another student to read/copy. If you allow anyone to copy your work, in part or whole, you are liable as well.
3. You programs should be running properly.
4. Hard **DEADLINE** of this assignment is **Friday, September 9th, 2022**. No late submissions will be accepted after the due date and time so manage emergencies beforehand.

Task 01: [25 Marks]

Recall that an ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory-efficient version of Doubly Linked List can be created using only one space for the address field with every node. This memory-efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bitwise XOR operation to save space for one address.

An ordinary doubly linked list stores addresses of the previous and next list items in each node, requiring two address fields. An XOR linked list compresses the same information into one address field by storing the bitwise XOR of the address for the previous and the address for the next in one field. Thus, in the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.

```
// Node structure of a memory efficient doubly linked list
struct Node
{
      int data;
      Node* npx; /* XOR of next and previous node */
};
```

Here, we store the XOR of the next and previous nodes with every node and we call it npx, which is the only address member we have with every node. When we insert a new node at the beginning, npx of new node will always be XOR of NULL and the current head. And npx of the current head must be changed to XOR of new node and node next to the current head. For a better understanding, you may visit wikipedia. Your task is to implement the following functions for XOR linked list.

```
      void insertATHead(int val);
      void insertAtTail(int val);
      void insertAfter(int val);
      void deletetBefore(int val)
      int RemoveAt Head();
      int RemoveAtTail();
      void printList();
```

Task 02: [25 Marks]

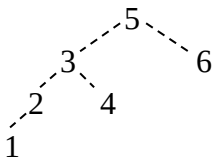
AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all the nodes. In this task, you have to modify your BST class to support AVL insertion and deletion. For AVL tree you can modify the node structure as follows:

```
template<class T>
class AVLNode
{
   T  data;
```

```
    int height;
    BSTNode<T>*  left;
    BSTNode<T>*  right;
      // Methods…
};
```

On insertion of a new node, you may need to modify the heights of all of the ancestors of the newly inserted node (Only ancestors). Also if the tree gets unbalanced on insertion you will need to balance it only once, deletion may require more than one-time balancing. Implement the following function:

1. **Constructor, destructor, Copy-constructor.**
2. **void insert (T value);**
3. **void deleteNode (BSTNode<T>* node)**
4. **AVLNode<T>* search (T val)**
5. **boolean isBST (BSTNode<T>* root);**
   Takes an Object of tree as a parameter and returns true if it is a BST, false otherwise. This function is to check that your tree is a valid BST after balancing.
6. **int getBalanceFactor (BSTNode<T>* node);**
7. **void printNodes (BSTNode<T>* root);** // Use level-order traversal to print the tree
8. **AVLNode* getParent(AVLNode* root);**
9. **void printTree ( );**
   This function should print tree in bellow form



10. Give a menu as part of main(), that allows the user to perform the following functionalities on your AVL.
    (a) Insert
    (b) Search
    (c) Delete
    (d) Print Tree

You may implement more utility functions for your ease such as getRoot, setRoot, getLeftChild, getRightChild, getHeight, getRotationType etc. Try to write modular code to avoid lengthy insert and delete functions.

Good Luck!