# DSA - Lab 6

# Linked List and iterators

**Q1.** Implement a class named LinkedList which has the following functionalities.

**1. IsEmpty()**; Return true if FRONT/TAIL is pointing to NULL otherwise false.

**2. InsertAtFront (val)**; Creates a new node with data = val and next = front and sets front of the list to the newly created node and checks if tail is pointing to NULL then sets it to front. (5 points)

**3. InsertAtTail(val);** Creates a new node with data = val and next = NULL. It checks if the list is empty. If it is then it sets front = tail = newly created node else tail->next = newly created node. (10 points)

**4. InsertAtMid(val)**; Creates a new node and inserts it in the middle. It sets the next pointers of the new node, node before middle node, and node after middle node accordingly.

**5. It has two parts: Let's say you have the following LinkedList**

**1 => 5 => 6 => 9 => -50 => 75**

- **InsertAfter(val, key):** It should enter the new Node with the value **key** after the value **val. e.g** for the above linked list if **InsertAtAfter(9, 35)** is a called then the new list should become:

  **1 => 5 => 6 => 9 => 35 => -50 => 75**
- **InsertBefore(val, key):** It should enter the new Node with value **key**, before the value **val. e.g** for the above linked list if **InsertBefore (9, 35)** is a called then the new list should become:

  **1 => 5 => 6 => 35 => 9 => -50 => 75**

**6. T GetFront();** returns the value pointed by FRONT. Note: ***This function must be used only if the IsEmpty() returns false.***

**7. T GetTail();** returns the value pointed by TAIL. Note: ***This function must be used only if the IsEmpty() returns false.***

**8. Node* Search(key);** returns the pointer to the element **key** if the list contains it. Note: ***if the key is not found it should return NULL.***

**9. A) RemoveFront**(); Stores front of the list in a temp variable, points **FRONT** of list to the next element in the list and deletes temp.

- Also make sure if the list is about to become empty it MUST reset FRONT and TAIL to NULL.
- Make sure you put the Check if the list is already empty then it shouldn't delete anything (rather return immediately).

**B) RemoveTail**(): Move Tail to the 2nd last element of the array. And delete the last element in the list. Make sure you put the checks of the list being empty(as stated in the above cases).

**C) RemoveMiddle():** Remove the middle node of the linked list.

## Q2. Now implement stack and Queue using LinkedList data structure:

```
template<typename T>
class Stack{
        LinkedList<T> S; // no other memory than this.
public:

        Stack();
        void Push(T V);
        T Pop();
        bool IsEmpty();
};
```

```
template<typename T>
class Queue{
        LinkedList<T> S; // no other memory than this.
public:

        Queue();
        void Enqueue(T V);
        T Dequeue();
        bool IsEmpty();
};
```

## Q3. Do the following tasks:

- **Recursively Printing the linked list - In Linked list order and then reverse linked list order**
- **Finding the middle element in the linked list.**
- **Finding out whether the linked list is circular or not.**
- **Reversing the linked list.**