**LAB SESSION 4: LINKED LIST IMPLEMENTATION OF STACKS AND QUEUES**

**AIM**: To implement stack and Queue using a Linked list.

**PROBLEM DEFINITION:**

1. 1. Develop a C program to implement Queues using Linked Lists.
2. 2. Develop a C program to implement Stacks using Linked lists and use the same to convert a postfix expression to its equivalent infix expression.

**THEORY:** In **linked list implementation of stack,** the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack. Stack is said to be overflown if the space left in the memory heap is not enough to create a node. The top most node in the stack always contains null in its address field.

## Adding a node to the stack (Push operation)

Adding a node to the stack is referred to as **push** operation. Pushing an element to a stack in linked list implementation is different from that of an array implementation. In order to push an element onto the stack, the following steps are involved.

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.
3. If there are some nodes in the list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.

Deleting a node from the stack (POP operation)  
Deleting a node from the top of stack is referred to as a pop operation. Deleting a node from the linked list implementation of stack is different from that in the array implementation. In order to pop an element from the stack, we need to follow the following steps :

**Check for the underflow condition:** The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.

**Adjust the head pointer accordingly:** In stack, the elements are popped only from one end, therefore, the value stored in the head pointer must be deleted and the node must be freed. The next node of the head node now becomes the head node.

**Linked List implementation of Queue:** Each node of a linked queue consists of two fields: data and next(storing address of next node). The data field of each node contains the assigned value, and the next points to the node containing the next item in the queue. A linked queue consists of two pointers, i.e., the front pointer and the rear pointer. The front pointer stores the address of the first element of the queue, and the rear pointer stores the address of the last element of the queue. Insertion is performed at the rear end, whereas deletion is performed at the front end of the queue. If front and rear both points to NULL, it signifies that the queue is empty.

Implementing a queue using a linked list allows us to grow the queue as per the requirements, i.e., memory can be allocated dynamically. A queue implemented using a linked list will not change its behavior and will continue to work according to the FIFO principle.

Steps for implementing queue using linked list:

1. Enqueue Function

Enqueue function adds an element to the end of the queue. It takes O(1) time. The last element can be tracked using the rear pointer.

* First, build a new node with given data.
* Check if the queue is empty or not.
* If a queue is empty then, a new node is assigned to the front and rear.
* Else make next of rear as new node and rear as a new node.

2. Dequeue Function

The dequeue function always removes the first element of the queue. It takes O(1) time. For dequeue, the queue must contain at least one element, else underflow conditions will occur.

* Check if queue is empty or not.
* If the queue is empty, then dequeue is not possible.
* Else store front in temp
* And make next of front as the front.
* Delete temp, i.e, free(temp).

3. Print

Print function is used to display the content of the queue. Since we need to iterate over each element of the queue to print it, the time complexity of the print function is O(n), where n = number of nodes in a queue.

* Check if queue contains at least one element or not.
* If the queue is empty print “No elements in the queue.”
* Else, define a node pointer and initialize it with the front.
* Display data of node pointer until the next node pointer becomes NULL.