Data Structure and Algorithms Lab

LAB Program 2

1. Design and implement a menu-driven Java/C/C++ program to perform stack operations using a singly linked list. The program should allow the user to:
   Insert an element into the stack (Push). Remove the top element from the stack (Pop) with proper underflow handling. Retrieve the top element without removing it (Peek). Check whether the stack is empty (isEmpty). Display all the elements of the stack from top to bottom (Display).

2. Design and implement a menu-driven Java/C/C++ program to perform queue operations using a singly linked list.
   The program should allow the user to: Insert an element at the rear (Enqueue). Remove the element from the front (Dequeue) with proper underflow handling. Retrieve (peek) the front element without removing it (Front/Peek). Check whether the queue is empty (isEmpty). Display all the elements of the queue from front to rear (Display).

   Note: Maintain front and rear pointers for O(1) enqueue/dequeue.
   On first insert, set both front and rear to the new node; when the last node is dequeued, set both to NULL.

Additional Questions:

3. A software company is developing an Undo/Redo feature for a text editor, where each editing action such as typing, deleting, or formatting needs to be tracked so that the most recent change can be undone first following the Last-In-First-Out (LIFO) principle. To achieve this, a stack can be implemented using a linked list in which each node holds the description of a user action, for example “Typed A”, “Deleted word”, or “Bold Applied”. This dynamic structure will allow efficient insertion and removal of actions, checking whether the stack is empty, and displaying the sequence of actions in the order they can be undone, thereby simulating the core functionality required for an Undo/Redo system.

4. A ticket booking system is being developed for a cinema hall where customers are served in the same order in which they arrive. To model this, a queue needs to be implemented using a linked list. Each node of the queue will hold the details of a customer request, such as the customer’s name or ticket number. The structure should allow new requests to be added at the rear of the queue and existing requests to be served from the front, ensuring a proper First-In-First-Out (FIFO) sequence. The implementation must also provide a way to check if the queue is empty and to display all pending customer requests in the order they will be processed.