# EXP 7: Implementation of Stack Using Linked List

## Aim

To write and execute a C program to **implement a stack using a linked list**

**THEORY**

A **Stack** is a linear data structure that follows the **LIFO (Last In First Out)** principle.Stacks can be implemented using arrays or linked lists.

Stack implementation using arrays is efficient when only **one stack** is required. However, when **multiple stacks** are needed, array implementation becomes inefficient due to fixed memory size and wastage. To overcome this problem, a **Linked List** is used to implement a stack. This type of stack is known as a **Linked Stack** .In a linked stack, each element is stored in a **node**. Each node contains two parts:

- **Data part** – stores the stack element
- **Link part** – stores the address of the next node

The **head of the linked list** is used as the **top of the stack**. All **insertion (PUSH)** and **deletion (POP)** operations are performed at the head.

- **PUSH operation**:
  A new node is created and inserted at the beginning of the list.
- **POP operation**:
  The node at the beginning of the list is removed.

Linked stack allows **dynamic memory allocation**, so stack overflow occurs only when memory is exhausted.It also provides **easy insertion and deletion** of elements.Thus, stack implementation using linked list is flexible and efficient compared to array implementation.

```
#include <stdio.h>
#include <stdlib.h>
/* Structure of a node */
struct node {
   int data;
   struct node *next;
};

struct node *top = NULL;

/* Function declarations */
void push();
void pop();
void peek();
void display();

int main() {
    int choice;

    do {
        printf("\n\n--- STACK MENU ---");
        printf("\n1. Push");
        printf("\n2. Pop");
        printf("\n3. Peek");
        printf("\n4. Display");
        printf("\n5. Exit");

        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: peek(); break;
            case 4: display(); break;
            case 5: printf("Exiting program"); break;
            default: printf("Invalid choice");
        }
    } while (choice != 5);

    return 0;
}

/* Push operation */
void push() {
    struct node *newnode;
    int value;

    newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Stack Overflow (Memory not allocated)");
        return;
    }

    printf("Enter value to push: ");
```

```c
    scanf("%d", &value);

    newnode->data = value;
    newnode->next = top;
    top = newnode;

    printf("Element pushed successfully");
}

/* Pop operation */
void pop() {
    struct node *temp;

    if (top == NULL) {
        printf("Stack Underflow");
        return;
    }

    temp = top;
    printf("Popped element: %d", temp->data);
    top = top->next;
    free(temp);
}

/* Peek operation */
void peek() {
    if (top == NULL) {
        printf("Stack is empty");
    } else {
        printf("Top element: %d", top->data);
    }
}

/* Display operation */
void display() {
    struct node *temp;

    if (top == NULL) {
        printf("Stack is empty");
        return;
    }

    temp = top;
    printf("Stack elements:\n");
    while (temp != NULL) {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
```