**1. Write a program that implements stack (its operations) using i) Arrays(Pointers).**

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
int *stack;
int top = -1;
void push(int item) {
    if (top >= MAX_SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        stack[++top] = item;
    }
}
int pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}
int peek() {
    if (top < 0) {
        printf("Stack is empty\n");
```

```
        return -1;
    } else {
        return stack[top];
    }
}
int main() {
    stack = (int *)malloc(MAX_SIZE * sizeof(int));
    push(1);
    push(2);
    push(3);
    printf("pop is %d\n", pop());
    printf("peek is %d\n", peek());
    free(stack);
    return 0;
}
```

**Output :** pop is 3
peek is 2

**2. Write a program that implements stack (its operations) using i)Linked list(Pointers).**

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
```

```
    struct Node* next;
};
struct Node* top = NULL;
void push(int item) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = item;
    new_node->next = top;
    top = new_node;
}
int pop() {
    if (top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        struct Node* temp = top;
        int popped = temp->data;
        top = top->next;
        free(temp);
        return popped;
    }
}
int peek() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return -1;
    } else {
        return top->data;
```

```
    }
}
int main() {
    push(5);
    push(6);
    push(7);
    printf("%d\n", pop()); // Output: 3
    printf("%d\n", peek()); // Output: 2
    return 0;
}
```

**Output :** pop is 7

peek is 6