AIM- to understand the concept of stack

THEORY- **Stack Data Structure in C**

A **stack** is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. Stacks can be implemented using arrays or linked lists.

Basic Operations of a Stack

1. **Push**: Add an element to the top of the stack.
2. **Pop**: Remove and return the top element of the stack.
3. **Peek** (or Top): Return the top element without removing it from the stack.
4. **isEmpty**: Check if the stack is empty.
5. **isFull**: Check if the stack is full (for stack implemented with an array).

Stack Implementation in C

Input-

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100  // Define the maximum size of the stack

typedef struct {
    int arr[MAX];
    int top;
} Stack;

// Initialize the stack
void initialize(Stack *s) {
    s->top = -1;
}

// Check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
}

// Check if the stack is full
int isFull(Stack *s) {
    return s->top == MAX - 1;
}
```

```
// Push an element onto the stack
void push(Stack *s, int value) {
    if (isFull(s)) {
        printf("Stack overflow! Cannot push %d onto the stack.\n", value);
        return;
    }
    s->arr[++(s->top)] = value;
    printf("Pushed %d to stack.\n", value);
}

// Pop an element from the stack
int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack underflow! Cannot pop from the stack.\n");
        return -1;  // Return a sentinel value to indicate error
    }
    return s->arr[(s->top)--];
}

// Peek at the top element of the stack
int peek(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty! Nothing to peek.\n");
        return -1;  // Return a sentinel value to indicate error
    }
    return s->arr[s->top];
}

// Display the elements of the stack
void display(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements are:\n");
    for (int i = s->top; i >= 0; i--) {
        printf("%d\n", s->arr[i]);
    }
}

// Main function to test stack operations
int main() {
    Stack s;
    initialize(&s);
```

```
    push(&s, 10);
    push(&s, 20);
    push(&s, 30);
    display(&s);

    printf("Top element is %d\n", peek(&s));

    printf("Popped %d from stack.\n", pop(&s));
    display(&s);

    printf("Popped %d from stack.\n", pop(&s));
    printf("Popped %d from stack.\n", pop(&s));
    printf("Popped %d from stack.\n", pop(&s));  // Stack underflow message

    return 0;
}
```

output-

```
Pushed 10 to stack.
Pushed 20 to stack.
Pushed 30 to stack.
Stack elements are:
30
20
10
Top element is 30
Popped 30 from stack.
Stack elements are:
20
10
Popped 20 from stack.
Popped 10 from stack.
Stack underflow! Cannot pop from the stack.
Popped -1 from stack.
```

Conclusion-

In this way we understood the concept of stacks in a detailed and understandable manner.