Lab Assignment: Stack Using Structure

Objective

To understand and implement stack data structure using structure-based encapsulation. This approach will reinforce the concept of data abstraction and promote modular code.

Functional Specification

You are required to implement a stack using the C programming language, where each stack is represented as a structure. The structure must encapsulate the elements of the stack (an array) and the top index.

Operations to Implement

1. **Initialize:** Set the stack's top index to -1.
2. **isFull:** Check if the top has reached MAX-1.
3. **isEmpty:** Check if the top is -1.
4. **Push:** Add an element to the top of the stack if it's not full.
5. **Pop:** Remove and return the top element if the stack is not empty.
6. **Peek:** Return the top element without removing it.

Function Signatures and Descriptions

1. `void initialize(Stack* s);`
   Initializes the stack by setting the `top` index to -1.
2. `int isFull(Stack* s);`
   Returns `1` if the stack is full (i.e., `top == MAX - 1`), otherwise `0`.
3. `int isEmpty(Stack* s);`
   Returns `1` if the stack is empty (i.e., `top == -1`), otherwise `0`.
4. `void push(Stack* s, int x);`
   Pushes an integer `x` onto the stack if it is not full. Displays appropriate messages for success or stack overflow.
5. `int pop(Stack* s);`
   Pops the top element from the stack if it is not empty and returns it. Displays a message for stack underflow if applicable.
6. `int peek(Stack* s);`
   Returns the top element of the stack without removing it. If the stack is empty, return a sentinel value and show an error message.

Time Complexities

Each of the stack operations should work in constant time O(1).

Advantages of Structure-Based Design

Using a structure to implement the stack provides the following advantages:

- **Clean and Modular Code**
    - Groups related data and logic, making the code easier to understand and maintain.
- **Multiple Instances**
    - You can declare multiple stack instances with their own state and storage.