**STACK Assignment**

• **Stack Data Structure (Using Array)**  
In this program, the stack is implemented using an array. A stack is a linear data structure that follows the **LIFO** (Last In, First Out) principle. This means the element that is inserted last will be removed first.

• **Dynamic Memory Allocation**  
The array for the stack is created dynamically using the new keyword. This allows the user to define the stack size at runtime.



• **Push Operation**  
When a new element is inserted into the stack, the program first checks whether the stack is full or not. If the stack is full, it prints a "Stack Overflow" message. Otherwise, it increments the top and stores the new value at that position.



• **Pop Operation**  
The pop operation removes the element from the top of the stack. If the stack is empty (i.e., top == -1), it prints a "Stack Underflow" message. Otherwise, it simply decreases the top to remove the element.



• **Peek Operation**  
The peek function is used to view the top element of the stack without removing it. If the stack is empty, it shows an appropriate message. Otherwise, it returns the top value.



• **isEmpty Function**  
This function checks whether the stack is empty or not. If top == -1, it means the stack has no elements.

• **Size Function (IsSize)**  
This function returns the current number of elements in the stack. It simply returns top + 1.

• **Display Function**  
The display function prints all elements of the stack from top to bottom (in LIFO order). If the stack is empty, it prints a message saying so.

• **Search Function**  
The search function is used to find a specific value in the stack. It performs a simple linear search from index 0 to top. If the value is found, it prints the index; otherwise, it shows a "not found" message.



• **Reverse Function**  
The reverse function does not actually reverse the stack but displays the elements in reverse order (i.e., similar to FIFO). It runs a loop from index 0 to top and prints the elements.



• **Object-Oriented Programming (OOP) Concepts**  
This code makes use of OOP concepts such as classes, constructors, encapsulation, and data hiding. All stack operations are written inside the class as member functions.



• **Modular Approach**  
Each operation (such as push, pop, peek, etc.) is written in a separate function, which makes the code more modular and easier to understand.

============================================================

**Real-Life Examples of Stack:**

**=>Stack of Plates in a Canteen**

Imagine you have a stack of plates in a canteen.

* You **add** plates one by one on top.
* When someone takes a plate, they always take the **top one first**.
* You can’t take the bottom plate without removing the ones above.

This is exactly how a stack works — **Last In = First Out**

**========================================================**