STACK DATA STRUCTURE

What is a Stack?

- A stack is a **linear data structure**
- it **follows the Last-In-First-Out (LIFO) principle**
- **last element** added is the **first to be removed.**
- It operates on a single end called the **"top,"**
- **elements are added and removed** from this **end.**

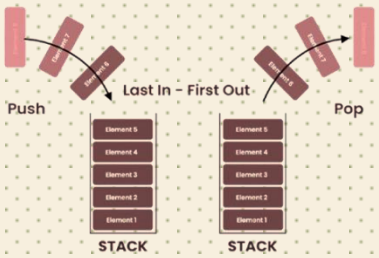

Figure 1 Stack Diagram

Key Points Related to Stack

- Stacks follow the **LIFO** principle (Last-In-First-Out).
- **Operations** occur **at one end** (the top).
- Can be **implemented using arrays or linked lists**.
- The stack supports operations like **push, pop, peek, isEmpty, isFull**, etc.
- It has a pointer (top) that keeps track of the current top element.

Working of Stack

1. **Push Operation**:
    - Check if the stack is full (top == size - 1).
    - If full, return an error (stack overflow).
    - If not full, increment the 'top' pointer and insert the new element at the new top index.

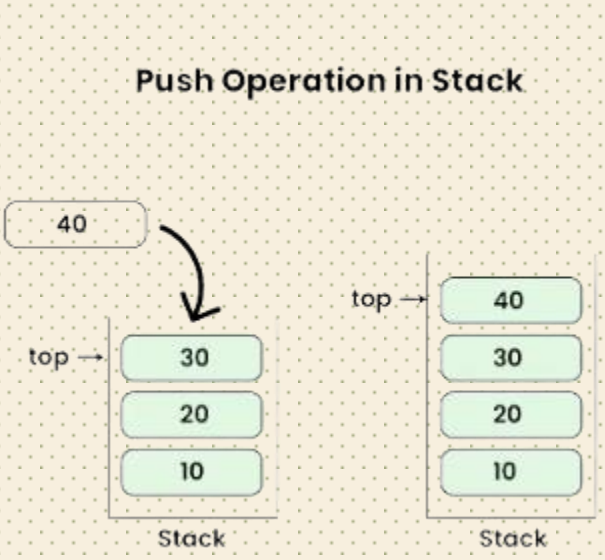

Figure 2 Push Operation Diagram

2. **Pop Operation**:
    - Check if the stack is empty (top == -1).
    - If empty, return an error (stack underflow).
    - If not empty, retrieve the element at the 'top' index and decrement the 'top' pointer.

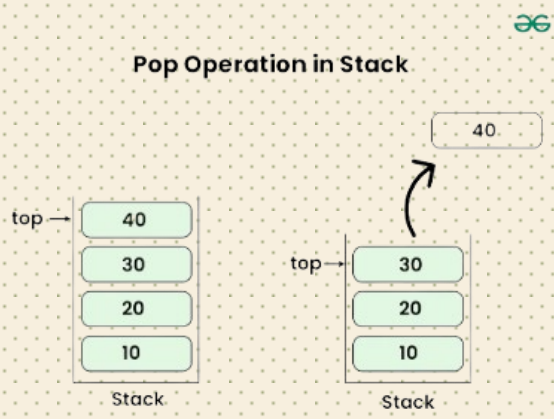

Figure 3 Pop Operation Diagram

3. **Peek Operation**:
    - Check if the stack is empty (top == -1).
    - If empty, return an error.
    - If not empty, return the element at the 'top' index without removing it.
4. **isEmpty Operation**:
    - Check if 'top' equals -1.
    - If yes, return true (stack is empty).
    - If no, return false (stack is not empty).
5. **isFull Operation**:
    - Check if 'top' equals size - 1.
    - If yes, return true (stack is full).
    - If no, return false (stack is not full).

Algorithm for Stack Implementation

- **Push(item)**: Add a new element to the top of the stack.
- **Pop()**: Remove the topmost element from the stack.
- **Peek()**: View the topmost element without removing it.
- **isEmpty()**: Check if the stack is empty.
- **isFull()**: Check if the stack is full.
- **size()**: Return the number of elements in the stack.
- **top()**: Access the topmost element (same as peek()).

- **display()**: Display all elements in the stack.
- **clear()**: Remove all elements from the stack.
- **reverse()**: Reverse the order of elements in the stack.
- **contains(item)**: Check if an item exists in the stack.

Advantages of Using Stacks

- **Maintaining Order**: Stacks naturally maintain the order of elements using the LIFO principle, which is useful for backtracking tasks.
- **Efficient Operations**: Operations like push, pop, and peek are performed in constant time (O(1)).
- **Memory Management Support**: Stacks help manage function calls in recursion by allocating and deallocating memory for local variables.
- **Reverse Functionality**: Stacks can reverse strings, expressions, or sequences by popping elements in the reverse order they were added.
- **Expression Conversion Aid**: Useful for converting expressions between infix, prefix, and postfix notations, often used in compilers.
- **Undo/Redo Capability**: Stacks are ideal for implementing undo/redo functionality in applications due to their LIFO nature.

Disadvantages Associated with Stacks

- **Size Limitation**: Array-based stacks have a fixed size, which can lead to stack overflow if the size is underestimated.
- **Limited Access Scope**: Elements can only be accessed from the top, making it inefficient to search for elements deeper in the stack.
- **One-Way Operations**: Stacks restrict operations to one end (top), limiting their flexibility compared to structures like queues or doubly linked lists.
- **Extra Memory Usage**: In linked-list implementations, each node requires extra memory for storing the pointer, which can lead to space inefficiencies, especially for large stacks.

Standard Stack Operations

- **Push(item)**: Insert an element at the top.
- **Pop()**: Remove and return the topmost element.
- **Peek()**: Retrieve the topmost element without removing it.
- **isEmpty()**: Check if the stack is empty.
- **isFull()**: Check if the stack is full.
- **size()**: Return the number of elements in the stack.

- **top()**: Return the topmost element (same as peek()).
- **display()**: Show all elements in the stack.
- **clear()**: Clear the stack.
- **reverse()**: Reverse the order of elements in the stack.
- **contains(item)**: Check if the item exists in the stack.