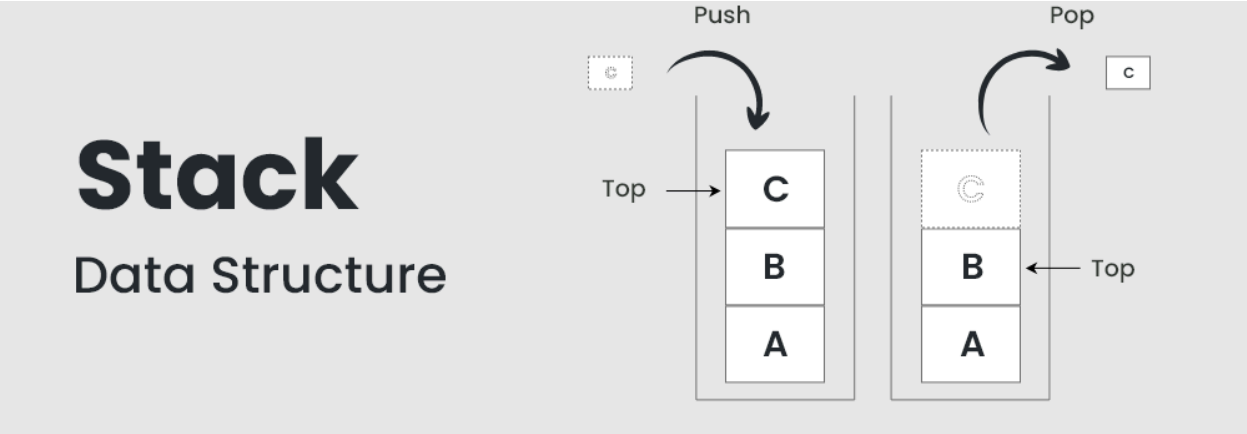
Stack:- A **Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO(Last In First Out)** or **FILO(First In Last Out)**. **LIFO** implies that the element that is inserted last, comes out first and **FILO** implies that the element that is inserted first, comes out last.



The basic operations we can do on a stack are:

- Push: Adds a new element on the stack.
- Pop: Removes and returns the top element from the stack.
- Peek: Returns the top element on the stack.
- isEmpty: Checks if the stack is empty.
- Size: Finds the number of elements in the stack.

Stacks can be implemented by using arrays or linked lists.

Stacks can be used to implement undo mechanisms, to revert to previous states, to create algorithms for depth-first search in graphs, or backtracking.

Stack Implementation using Arrays

Reasons to implement stacks using arrays:

- Memory Efficient: Array elements do not hold the next element's address as linked list nodes do.
- Easier to implement and understand: Using arrays to implement stacks requires less code than using linked lists, and for this reason, it is typically easier to understand as well.

A reason for not using arrays to implement stacks:

- Fixed-size: An array occupies a fixed part of the memory. This means that it could take up more memory than needed, or if the array fills up, it cannot hold more elements.

This is how a stack can be implemented using an Array.

```
class myStack{
    int [] arr = new int[10];
    int size = 0;

    public void push(int data){
        if(size == arr.length){
            int si = arr.length + arr.length/3;
            int [] arr1 = new int[si];
            for(int i=0; i<arr.length; i++){
                arr1[i] = arr[i];
            }
            arr = arr1;
        }
        arr[size++] = data;
    }
    public int peek(){
        if(size == 0){
            return -1;
        }
        return arr[size-1];
    }
    public int pop(){
        return arr[--size];
    }
```

Stack Implementation using Linked Lists

A reason for using linked lists to implement stacks:

- Dynamic size: The stack can grow and shrink dynamically, unlike with arrays.

Reasons for not using linked lists to implement stacks:

- Extra memory: Each stack element must contain the address to the next element (the next linked list node).
- Readability: The code might be harder to read and write for some because it is longer and more complex.

This is how a stack can be implemented using a linked list.

```
class Node{
    int data;
    Node next;

    Node(int data){
        this.data = data;
    }
}

class myStack{
    Node Peek = null;
    int size = 0;

    public void push(int data){
        if(Peek == null){
            Peek = new Node(data);
            size++;
        }else{
            Node node = new Node(data);
            node.next = Peek;
            Peek = node;
            size++;
        }
    }

    public int peek(){
        return Peek.data;
    }

    public int pop(){
        int val = Peek.data;
        Peek = Peek.next;
        size--;
        return val;
    }
```