Introduction to stacks

- Stacks are simple data structures that allow us to store and retrieve data sequentially.
- A stack is a linear data structure like arrays and linked lists.
- It is an abstract data type **(ADT)**.
- In a stack, the order in which the data arrives is essential. It follows the LIFO order of data insertion/abstraction. LIFO stands for **Last In First Out.**
- Consider the example of a pile of books:

Here, unless the book at the topmost position is removed from the pile, we can't have access to the second book from the top and similarly, the books below the second one. When we apply the same technique to the data in our program then, this pile-type structure is said to be a stack.

Like deletion/removal, we can only insert the book at the top of the pile rather than at any other position. This means that the object/data that made its entry at the last would be one to come out first, hence known as **LIFO.**

Various operations on stacks

In a stack, insertion and deletion are done at one end, called **top**.

- **Insertion**: This is known as a **push** operation.
- **Deletion**: This is known as a **pop** operation.

Main stack operations

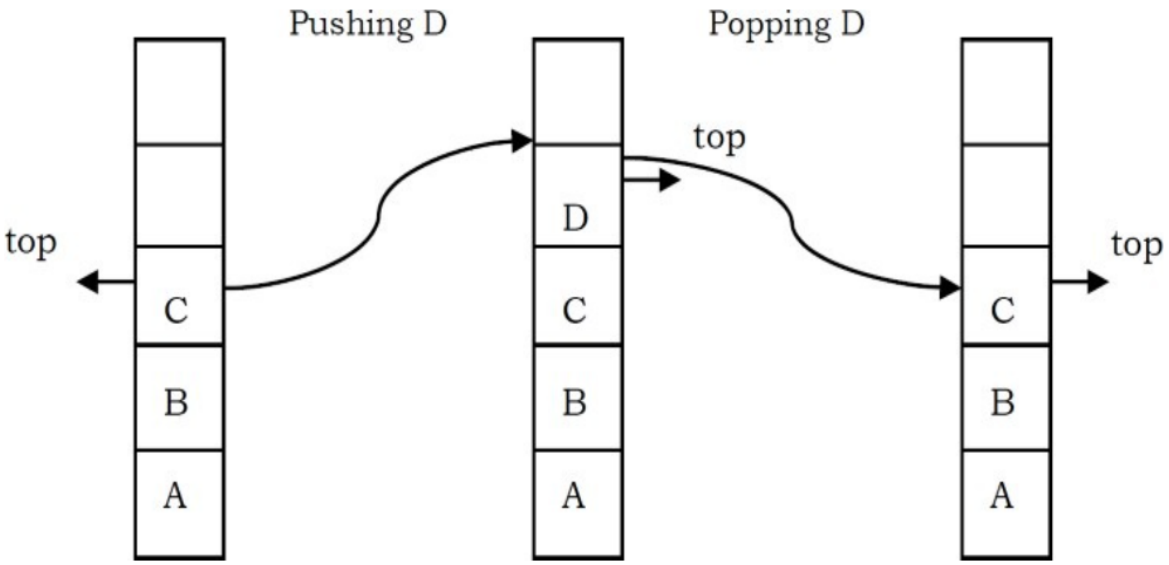
• **push (data)**: Insert data onto the stack.

• **pop()**: Removes and returns the last inserted element from the stack.

Auxiliary stack operations

• **top()**: Returns the last inserted element without removing it.

• **int size()**: Returns the number of elements stored in the stack.



• **boolean isEmpty()**: Indicates whether any elements are stored in the stack or not i.e. whether the stack is empty or not.

Implementation of Stacks

Stacks can be implemented using arrays, linked lists, or queues. The underlying algorithm for implementing operations of the stack remains the same.

- **Push operation**

```
function push(data, stack)
        //   data : the data to be inserted into the stack.
        if stack is full
                return null
        /*
          top : It refers to the position (index in arrays) of the last
          element into the stack
        */
        top = top + 1
        stack[top] = data
```

- **Pop operation**

```
function pop(stack)
        if stack is empty
```

```
            return null

        //  Retrieving data of the element to be popped.
        data = stack[top]
        //  Decrementing top
        top = top - 1
        return data
```

- **Top operation**

```
function top(stack)
        if stack is empty
                return null
        else
                return stack[top]
```

- **isEmpty operation**

```
function isEmpty(stack)
        if top is null
                return true
        else
                return false
```

Time Complexity of various operations

Let ‘n’ be the number of elements in the stack. The complexities of stack operations with this representation can be given as:

Operations	Time Complexity
Push(data)	O(1)
Pop()	O(1)
int top()	O(1)
boolean isEmpty()	O(1)
int size()	O(1)
boolean isFull()	O(1)

Exceptions

- Attempting the execution of an operation may sometimes cause an error condition, called an exception.
- Exceptions are said to be “thrown” by an operation that cannot be executed.
- Attempting the execution of pop() on an empty stack throws an exception called **Stack Underflow**.
- Trying to push an element in a full-stack throws an exception called **Stack Overflow**.

Applications of stacks

- Stacks are useful when we need **dynamic addition and deletion of elements in our data structure**. Since stacks require O(1) time complexity for all the operations, we can achieve our tasks very efficiently.
- It also has applications in various **popular algorithmic problems** like:-
    - Tower of Hanoi
    - Balancing Parenthesis
    - Infix to postfix
    - Backtracking problems
- It is useful in many **Graph Algorithms** like topological sorting and finding strongly connected components.
- Apart from all these it also has very practical applications like:
    - **Undo and Redo operations in editors**
    - **Forward and backward buttons in browsers**
    - **Allocation of memory by an operating system while executing a proces**