Stack

Stack is a linear list where both insertion and deletion operations are performed at one end of the list.

A stack is a linear data structure that follows the principle of Last In First Out (LIFO).

This means the last element inserted inside the stack is removed first.

Dr. Shiladitya Chowdhury
Ph.D(Computer Science & Engineering)(JU)
Assistant Professor, Dept. of MCA
Techno Main Saltlake, Kolkata 700091

Real world example of stack

We can think of the stack data structure as the pile of plates on top of another.

Fig 1: Stack representation similar to a pile of plate

Here, we can:

Put a new plate on top
Remove the top plate

If we want the plate at the bottom, you must first remove all the plates on top.

This is exactly how the stack data structure works.

LIFO Principle of Stack

In programming terms, putting an item on top of the stack is called ***push*** and removing an item is called ***pop***.

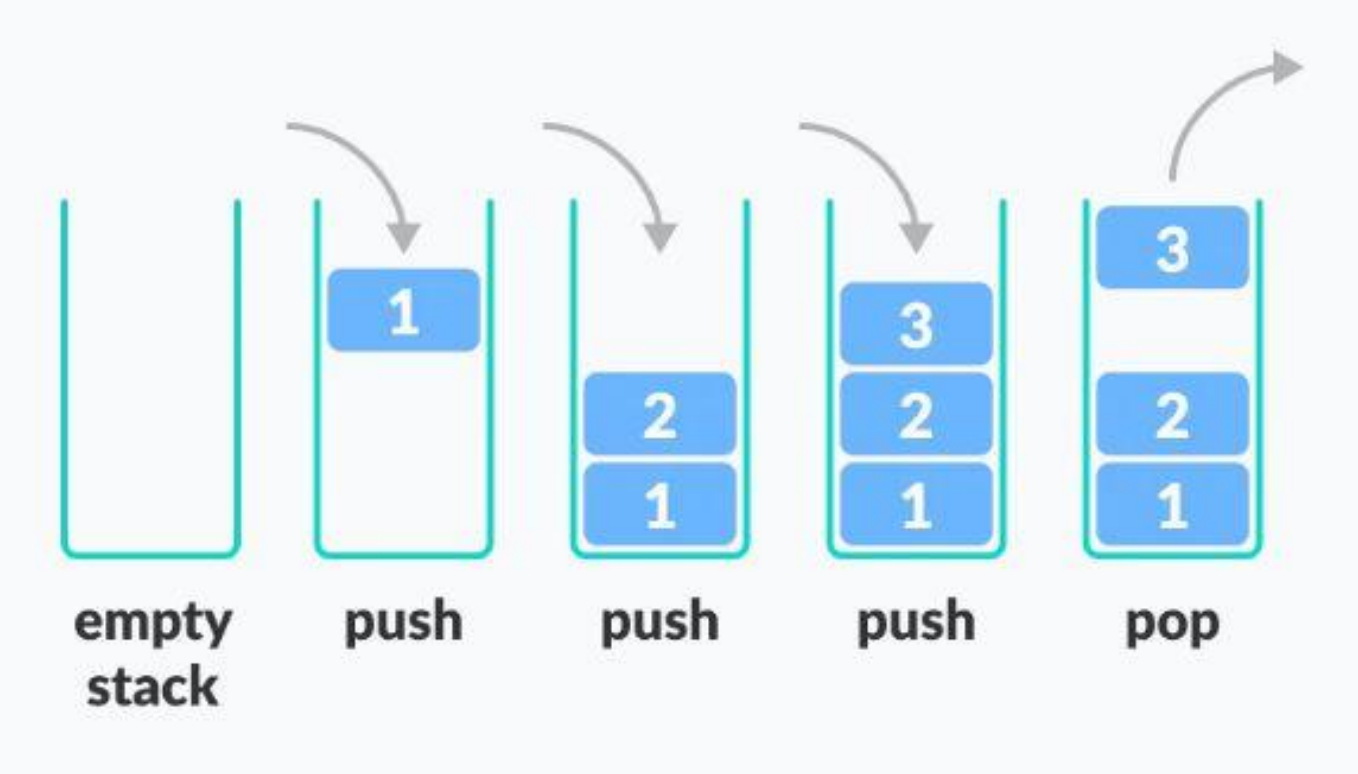

Fig 2: Stack Push and Pop Operations

Dr. Shiladitya Chowdhury
Ph.D(Computer Science & Engineering)(JU)
Assistant Professor, Dept. of MCA
Techno Main Saltlake, Kolkata 700091

*In Fig 2, although item **3** was kept last, it was removed first.*

This is exactly how the LIFO (Last In First Out) Principle works.

There are some basic operations that allow us to perform different actions on a stack.

Push: Add an element to the top of a stack

Pop: Remove an element from the top of a stack

IsEmpty: Check if the stack is empty

IsFull: Check if the stack is full

Working of Stack Data Structure

The operations work as follows:

1. A pointer called ***top*** is used to keep track of the top element in the stack.

2. When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing ***top* ==-1**

3. On pushing an element, we increase the value of ***top*** and place the new element in the position pointed to by ***top***.

4. On popping an element, we return the element pointed to by ***top*** and reduce its value.

5. Before pushing, we check if the stack is already full.

6. Before popping, we check if the stack is already empty.

Dr. Shiladitya Chowdhury
Ph.D(Computer Science & Engineering)(JU)
Assistant Professor, Dept. of MCA
Techno Main Saltlake, Kolkata 700091

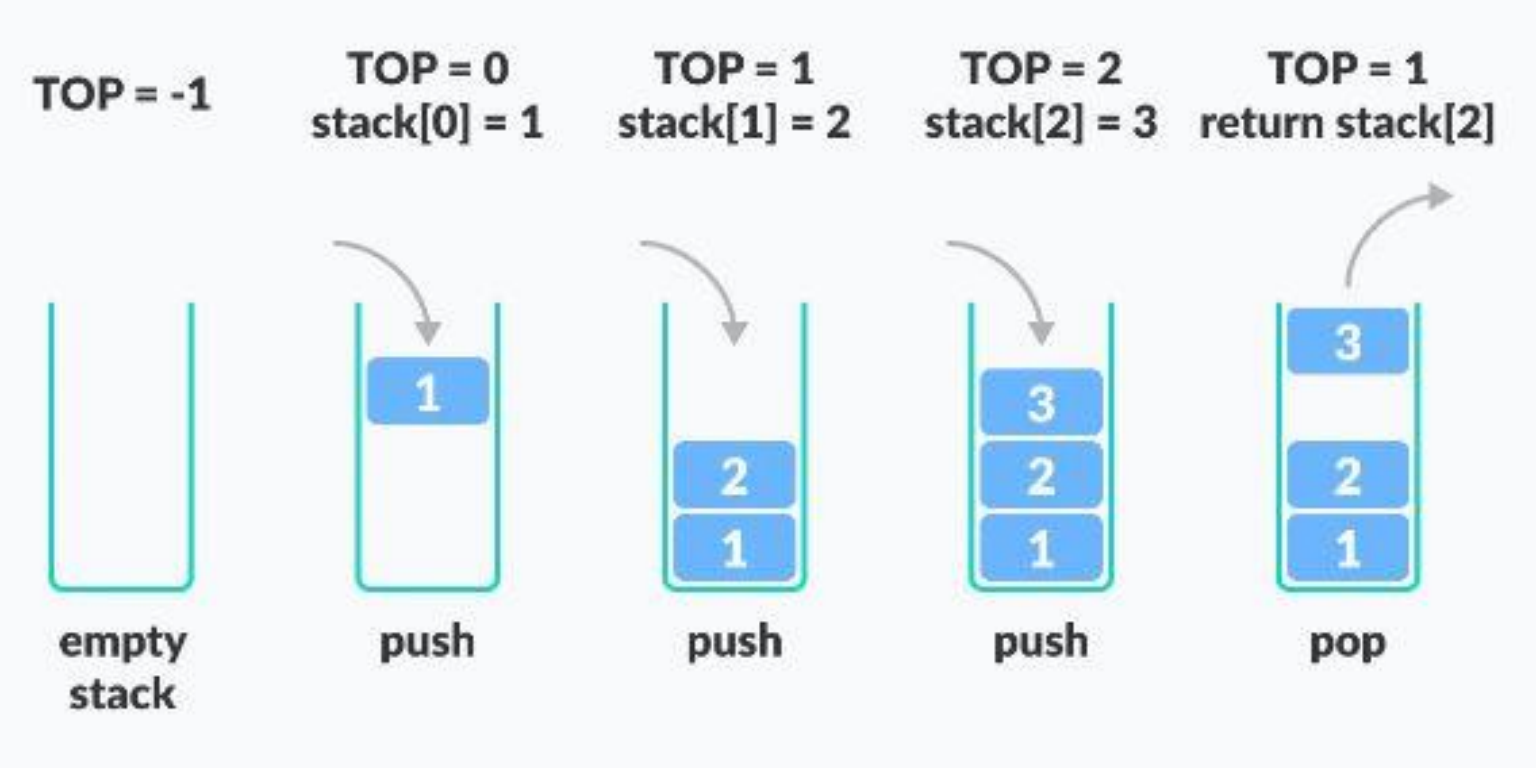

Fig 3: Working of Stack Data Structure

Applications of stack

Infix expression:

In infix expression, an operator is in-between every pair of operands.
Example: 10 + 5

Prefix expression:

In prefix expression, an operator is in before every pair of operands.
Example: + 10 5

Postfix expression:

In postfix expression, an operator is followed by every pair of operands.
Example: 10 5 +

Dr. Shiladitya Chowdhury
Ph.D(Computer Science & Engineering)(JU)
Assistant Professor, Dept. of MCA
Techno Main Saltlake, Kolkata 700091

Evaluation of Postfix Expression

The Postfix notation is used to represent algebraic expressions.

The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

Following is algorithm for evaluation postfix expressions.

1) Create a stack to store operands.

2) Scan the given expression and do following for every scanned element.

a) If the element is a number, push it into the stack.

b) If the element is a operator, pop two operands from stack.
Evaluate the operator and push the result back to the stack.

3) When the expression is ended, the number in the stack is the final answer.

Example:

Evaluate the following postfix expression:

3 5 6 * + 1 - $

Stack Content	Remaining Expression
	3 5 6 * + 1 - $
3	5 6 * + 1 - $
3 5	6 * + 1 - $
3 5 6	* + 1 - $
3 30	+ 1 - $
33	1 - $
33 1	- $
32	$

Infix to Postfix Conversion

To convert infix expression to postfix expression, we will use the stack data structure.

Algorithm

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else
    If the input priority of the scanned operator is greater than the in-stack priority of the topmost element of the stack OR the stack is empty OR the stack contains a '('
        Push the scanned operator.
    Else
        If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
        Else
            Pop all the operators from the stack whose in-stack priority is greater than or equal to the input priority of the scanned operator. After doing that Push the scanned operator to the stack.
4. Repeat steps 2-3 until infix expression is scanned.
5. Pop and output from the stack until it is not empty.

Operator	In-Stack Priority	Input Priority
+ -	1	1
* /	2	2
^	3	4
(	0	4
)	×	0

Example 1

Infix Expression: 3 ^ 2 ^ 3

Remaining Input Expression	Stack Content	Output
3 ^ 2 ^ 3		
^ 2 ^ 3		3
2 ^ 3	^	3
^ 3	^	3 2
3	^ ^	3 2
	^ ^	3 2 3
		3 2 3 ^ ^

Example 2

Infix Expression: ***3 + 5 * 6 - 1***

Remaining Input Expression	Stack Content	Output
3 + 5 * 6 – 1		
+ 5 * 6 - 1		3
5 * 6 - 1	+	3
* 6 - 1	+	3 5
6 - 1	+ *	3 5
- 1	+ *	3 5 6
1	-	3 5 6 * +
	-	3 5 6 * + 1
		3 5 6 * + 1 -