STACKS LEARNING MODULE

Table of Contents

1. Basics of Stacks
2. Real-life Analogies of Stacks
3. Stack Implementation
    - Using Arrays
    - Using Linked Lists
4. Expression Conversion and Evaluation
    - Infix to Postfix Conversion
    - Postfix Evaluation
    - Infix to Prefix Conversion
    - Prefix Evaluation
5. Next Greater and Smaller Element Problems
6. Min and Max Stacks
7. Balanced Parentheses Problem
8. Advanced Stack Problems and Challenges
9. Practice Problems and Further Reading

BASICS OF STACKS

What is a Stack?

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

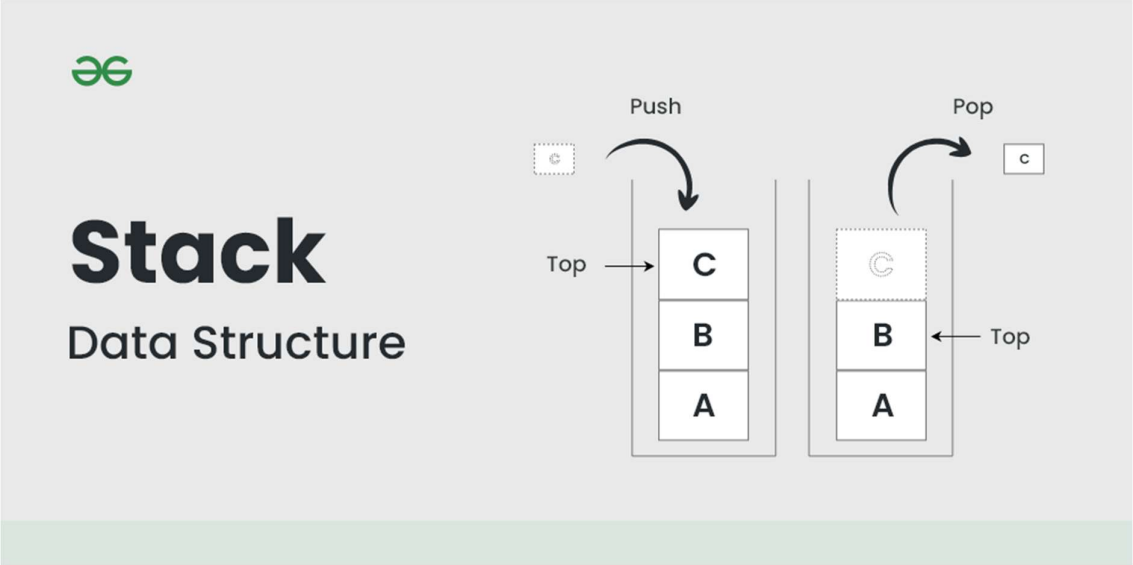
- **Last In First Out (LIFO):** The last element inserted is the first one to be removed.

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.

It behaves like a stack of plates, where the last plate added is the first one to be removed. **Think of it this way:**

- Pushing an element onto the stack is like adding a new plate on top.

- Popping an element removes the top plate from the stack.



Basics of Stack Data Structure

- Introduction to Stack
- Stack Array Implementation
- Stack Linked List Implementation
- Stack Implementation using Deque
- Applications of Stack

Implementations of Stack in Different Languages

- Stack in C++ STL
- Stack in Java
- Stack in Python
- Stack in C#
- Stack in JavaScript

Easy Problems on Stack Data Structures

- The Celebrity Problem
- Implement Queue using Stacks
- Implement two stacks in an array
- Implement Stack using Queues

- Stack using priority queue or heap
- Stack using single queue
- Infix to Postfix
- Prefix to Infix
- Prefix to Postfix
- Postfix to Prefix
- Postfix to Infix
- Infix To Prefix
- Check for balanced parentheses
- Arithmetic Expression Evalution
- Evaluation of Postfix Expression
- Reverse a stack using recursion
- Reverse Words
- Reverse a string using stack
- Reversing a Queue
- Reversing the first K of a Queue
- A Data Structure with O(1) Operations

Medium Problems on Stack Data Structures

- k Stacks in an Array
- Mergable Stack
- Previous Smaller Element
- Next Greater Element
- Stock Span Problem
- Buildings Facing Sun
- Next Smaller of next Greater in an array
- Next Greater Frequency Element
- Max product of indexes of greater on left and right
- Iterative Tower of Hanoi
- Sort a stack using a temporary stack

- Reverse a stack without using extra space in O(n)
- Delete middle of a stack
- Check if a queue can be sorted into another queue
- Check if an array is stack sortable
- Iterative Postorder Traversal | Set 1 (Using Two Stacks)
- Index of closing bracket for a given opening bracket
- Max Diff between nearest left and right smaller elements
- Delete consecutive same words in a sequence

Hard Problems on Stack Data Structures

- Largest Rectangular Area in a Histogram
- Sum of Max of all Subarrays
- Max of Mins of every window size
- Design a stack that supports getMin()
- Design a stack with max frequency operations
- Print next greater number of Q queries
- Length of the longest valid substring
- Iterative Postorder Traversal | Set 2 (Using One Stack)
- Print ancestors of a given binary tree node without recursion
- Expression contains redundant bracket or not
- Find if an expression has duplicate parenthesis
- Iterative method to find ancestors in a binary tree
- Stack Permutations
- Remove brackets from an algebraic string containing + and – operators
- Range Queries for Longest Correct Bracket Subsequence

Quick Links :

- 'Practice Problems' on Stack
- 'Videos' on Stack
- 'Quizzes' on Stack

Recommended:

- **Learn Data Structure and Algorithms | DSA Tutorial**
- Stack in Scala

BASIC OPERATIONS ON A STACK

Operation	Description
push(x)	Insert element x onto stack
pop()	Remove top element
peek()	Get top element without removing
isEmpty()	Check if stack is empty
size()	Returns number of elements

Visualization

Imagine a stack of plates; you add plates on top and remove plates from the top only.

---

2. REAL-LIFE ANALOGIES OF STACKS

- **Stack of plates:** Plates stacked on top of each other, you always take the top plate first.
- **Undo operation in text editors:** Last action done is the first to be undone.
- **Browser back button:** The last page you visited is the first to go back to.
- **Function call stack:** Functions call others, and return in reverse order.

---

3. STACK IMPLEMENTATION

3.1 Stack Using Arrays

- Fixed size stack (can be dynamically resized but often fixed)
- Operations involve index tracking

Java Code:

```
class StackUsingArray {
    private int[] stack;
    private int top;
    private int capacity;

    public StackUsingArray(int size) {
        stack = new int[size];
        capacity = size;
        top = -1;
    }

    public void push(int x) {
        if (top == capacity - 1) {
            System.out.println("Stack Overflow");
            return;
        }
        stack[++top] = x;
    }

    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow");
            return -1;
        }
        return stack[top--];
    }

    public int peek() {
```

```
        if (top == -1) {
            System.out.println("Stack is empty");
            return -1;
        }
        return stack[top];
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public int size() {
        return top + 1;
    }
}
```

3.2 STACK USING LINKED LISTS

- Dynamic size
- Nodes point to next element, head acts as top

Java Code:

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

        Node(int d) {
            data = d;
            next = null;
        }
```

```
}

private Node top;

public StackUsingLinkedList() {
    top = null;
}

public void push(int x) {
    Node newNode = new Node(x);
    newNode.next = top;
    top = newNode;
}

public int pop() {
    if (top == null) {
        System.out.println("Stack Underflow");
        return -1;
    }
    int value = top.data;
    top = top.next;
    return value;
}

public int peek() {
    if (top == null) {
        System.out.println("Stack is empty");
        return -1;
    }
    return top.data;
```

```
    }

    public boolean isEmpty() {
        return top == null;
    }
}
```

---

4. EXPRESSION CONVERSION AND EVALUATION

4.1 Infix to Postfix Conversion

Infix expressions are natural but harder for computers to evaluate due to operator precedence and parentheses. Postfix (Reverse Polish notation) makes evaluation straightforward.

Algorithm:

- Scan the infix expression left to right.
- Use a stack to hold operators.
- If operand, add to output.
- If operator, pop from stack while top has higher or equal precedence, then push current operator.
- If '(', push it; if ')', pop till '('.

Operator Precedence

Operator	Precedence
*, /	2
+, -	1
(	0

Java Code:

```
import java.util.Stack;

public class InfixToPostfix {
```

```
public static int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
    }
    return -1;
}

public static String infixToPostfix(String exp) {
    StringBuilder result = new StringBuilder();
    Stack<Character> stack = new Stack<>();

    for (int i = 0; i < exp.length(); i++) {
        char c = exp.charAt(i);

        if (Character.isLetterOrDigit(c)) {
            result.append(c);
        } else if (c == '(') {
            stack.push(c);
        } else if (c == ')') {
            while (!stack.isEmpty() && stack.peek() != '('
                result.append(stack.pop());
            stack.pop(); // Remove '('
        } else { // operator
            while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
```

```
                result.append(stack.pop());
            }
            stack.push(c);
        }
    }

    while (!stack.isEmpty()) {
        result.append(stack.pop());
    }
    return result.toString();
  }

  public static void main(String[] args) {
    String infix = "a+b*(c^d-e)^(f+g*h)-i";
    System.out.println("Postfix: " + infixToPostfix(infix));
  }
}
```

4.2 Postfix Evaluation

- Use a stack to store operands.
- When operator encountered, pop two operands, evaluate, push result back.

Java Code:

```
import java.util.Stack;
public class PostfixEvaluation {

  public static int evaluatePostfix(String exp) {
    Stack<Integer> stack = new Stack<>();
```

```
    for (int i = 0; i < exp.length(); i++) {
        char c = exp.charAt(i);

        if (Character.isDigit(c))
            stack.push(c - '0');
        else {
            int val2 = stack.pop();
            int val1 = stack.pop();

            switch (c) {
                case '+':
                    stack.push(val1 + val2);
                    break;
                case '-':
                    stack.push(val1 - val2);
                    break;
                case '*':
                    stack.push(val1 * val2);
                    break;
                case '/':
                    stack.push(val1 / val2);
                    break;
            }
        }
    }
    return stack.pop();
}

public static void main(String[] args) {
    String postfix = "231*+9-";
```

```
        System.out.println("Evaluation: " + evaluatePostfix(postfix)); // Output: -4
    }
}
```

4.3 Infix to Prefix Conversion

- Reverse the infix expression.
- Swap '(' with ')'.
- Convert to postfix.
- Reverse the postfix expression to get prefix.

4.4 Prefix Evaluation

- Scan expression from right to left.
- If operand, push.
- If operator, pop two operands, apply operator, push result.

---

5. NEXT GREATER AND SMALLER ELEMENT

Problem Statement

Given an array, find the next greater element (NGE) for every element in the array.

Approach

- Use stack to track candidates.
- Iterate from right to left.
- For each element, pop stack until you find a greater element.
- If none, answer is -1.

Java Code (Next Greater Element):

```
import java.util.Stack;
```

```
public class NextGreaterElement {

    public static int[] nextGreater(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr = {4, 5, 2, 25};
        int[] res = nextGreater(arr);
        for (int val : res) {
            System.out.print(val + " ");
        }
    }
}
```

Next Smaller Element

- Same as above but conditions reversed (find smaller elements).

---

6. MIN AND MAX STACK

Problem Statement

Design a stack that supports push, pop, and retrieving the minimum or maximum element in constant time.

Approach

- Maintain an auxiliary stack to keep track of min/max values.
- When pushing, update auxiliary stack with min/max so far.
- Pop both stacks simultaneously.

Java Code (Min Stack):

```
import java.util.Stack;

public class MinStack {

    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> minStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()) {
            minStack.push(x);
        }
    }

    public int pop() {
        int val = stack.pop();
        if (val == minStack.peek()) {
            minStack.pop();
        }
```

```
        return val;
    }

    public int getMin() {
        return minStack.peek();
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
}
```

---

7. BALANCED PARENTHESES PROBLEM

Problem Statement

Check if a string of parentheses, brackets, and braces is balanced.

Approach

- Use a stack.
- Push opening brackets.
- For closing brackets, pop and check if matching.

Java Code:

```
import java.util.Stack;

public class BalancedParentheses {

    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();
```

```
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if (!isMatching(top, c)) return false;
            }
        }
        return stack.isEmpty();
    }

    private static boolean isMatching(char open, char close) {
        return (open == '(' && close == ')') ||
               (open == '[' && close == ']') ||
               (open == '{' && close == '}');
    }

    public static void main(String[] args) {
        String expr = "{()}[]";
        System.out.println("Is balanced? " + isBalanced(expr));
    }
}
```

---

8. <u>ADVANCED STACK PROBLEMS AND CHALLENGES</u>

- Implement **Largest Rectangle in Histogram** using stack.
- Evaluate **Stock Span Problem** using stack.
- **Trapping Rain Water Problem** using stack.

- Implement **Design Browser History** using stacks.
- Solve **Decode String** problem (e.g., "3[a2[c]]" → "accaccacc").
- **Maximal Rectangle** in binary matrix (stack + DP).
- Implement **Undo-Redo functionality** in text editor.
- Design **Double-ended Stack (Deque)**.

---

9. PRACTICE PROBLEMS AND FURTHER READING

- LeetCode: 20 (Valid Parentheses), 84 (Largest Rectangle in Histogram), 150 (Evaluate Reverse Polish Notation), 155 (Min Stack)
- GeeksforGeeks: Stack Data Structure tutorials and problems.
- HackerRank: Balanced Brackets, Expression Evaluation problems.
- Books: *"Data Structures and Algorithms Made Easy"* by Narasimha Karumanchi, *"Introduction to Algorithms"* by Cormen et al.

RECOMMENDED YOUTUBE LINKS

1. https://youtu.be/7m1DMYAbdiY?si=8gLJGg-ua3bSa9WA
2. https://youtu.be/bxRVz8zklWM?si=jPkAijDm1uQcAYot
3. https://youtu.be/1byexQBGhas?si=FFTZEG8JGdVxWII_
4. https://youtu.be/rHQI4mrJ3cg?si=TNF2D6EXNiYoHG4Z
5. https://youtu.be/B5RbUqdPK80?si=WZA5QViiMN28r_bx
6. https://youtu.be/tqQ5fTamIN4?si=-Z_X3O-ganlTFBEw
7. https://youtu.be/Evn_JL40go4?si=U8MUVfU_zQ-Jp1jI
8. https://youtu.be/S9LUYztYLu4?si=3zcOIWrV9-fjm4JQ