

**Lab Problem 1. Balanced Parentheses Checker (Any Four)**

**Concept Used:** Stack (LIFO principle)

**Problem:**

Write a program using Stack to check if an expression has balanced parentheses — including {}, [], and ().

**Example:**

Input: {[()()]} → Output: Balanced

Input: {[(])} → Output: Not Balanced

**Hint:** Push opening brackets to the stack. Pop when a closing bracket is encountered. Check matching types.

PROGRAM

import java.util.\*;

public class BalancedParenthesesChecker {

public static boolean isBalanced(String expr) {

Stack<Character> stack = new Stack<>();

for (char ch : expr.toCharArray()) {

if (ch == '(' || ch == '{' || ch == '[') {

stack.push(ch);

} else if (ch == ')' || ch == '}' || ch == ']') {

if (stack.isEmpty()) return false;

char top = stack.pop();

if ((ch == ')' && top != '(') ||

(ch == '}' && top != '{') ||

(ch == ']' && top != '[')) {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter expression: ");

String expr = sc.nextLine();

if (isBalanced(expr))

System.out.println("Balanced");

else

System.out.println("Not Balanced");

sc.close();

}

}

OUTPUT

Input: {[()()]}

Output: Balanced

Input: {[(])}

Output: Not Balanced

**Lab Problem 2. Reverse a String Using Stack**

**Concept Used:** Stack for element reversal

**Problem:**

Reverse a given string using stack operations (push, pop).

**Example:**

Input: HELLO → Output: OLLEH

**Hint:** Push each character into the stack, then pop them to form the reversed string.

PROGRAM

import java.util.\*;

public class ReverseStringUsingStack {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = sc.nextLine();

Stack<Character> stack = new Stack<>();

for (char ch : input.toCharArray()) {

stack.push(ch);

}

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

System.out.println("Reversed String: " + reversed);

sc.close();

}

}

OUTPUT

Input: HELLO

Output: OLLEH

**Lab Problem 3. Evaluate Postfix Expression**

**Concept Used:** Stack for expression evaluation

**Problem:**

Evaluate a postfix (Reverse Polish Notation) expression.

**Example:**

Input: 6 3 2 + \* → Output: 30

**Hint:** Scan from left to right; push operands and apply operators by popping top two elements.

PROGRAM

import java.util.\*;

public class EvaluatePostfixExpression {

public static int evaluate(String expr) {

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

String[] tokens = expr.split(" ");

for (String token : tokens) {

if (Character.isDigit(token.charAt(0))) {

stack.push(Integer.parseInt(token));

} else {

int b = stack.pop();

int a = stack.pop();

switch (token) {

case "+": stack.push(a + b); break;

case "-": stack.push(a - b); break;

case "\*": stack.push(a \* b); break;

case "/": stack.push(a / b); break;

default:

System.out.println("Invalid operator: " + token);

}

}

}

return stack.pop();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter postfix expression (space-separated): ");

String expr = sc.nextLine();

int result = evaluate(expr);

System.out.println("Result: " + result);

sc.close();

}

}

OUTPUT

Input: 6 3 2 + \*

Output: Result: 30

**Lab Problem 4. Implement Stack Using List**

**Concept Used:** Stack operations (push, pop, peek, isEmpty)

**Problem:**

Implement a Stack using Python’s list structure and demonstrate all basic operations. **Hint:** Use append() for push and pop() for pop operations.

PROGRAM

import java.util.\*;

class StackUsingList {

private List<Integer> stackList = new ArrayList<>();

public void push(int item) {

stackList.add(item);

System.out.println(item + " pushed to stack");

}

public int pop() {

if (isEmpty()) {

System.out.println("Stack Underflow!");

return -1;

}

return stackList.remove(stackList.size() - 1);

}

public int peek() {

if (isEmpty()) {

System.out.println("Stack is empty!");

return -1;

}

return stackList.get(stackList.size() - 1);

}

public boolean isEmpty() {

return stackList.isEmpty();

}

public void display() {

System.out.println("Stack: " + stackList);

}

public static void main(String[] args) {

StackUsingList stack = new StackUsingList();

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Enter command (PUSH <num>/POP/PEEK/DISPLAY/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("PUSH")) {

int num = sc.nextInt();

stack.push(num);

}

else if (cmd.equalsIgnoreCase("POP")) {

int popped = stack.pop();

if (popped != -1)

System.out.println("Popped: " + popped);

}

else if (cmd.equalsIgnoreCase("PEEK")) {

int top = stack.peek();

if (top != -1)

System.out.println("Top element: " + top);

}

else if (cmd.equalsIgnoreCase("DISPLAY")) {

stack.display();

}

else if (cmd.equalsIgnoreCase("EXIT")) {

break;

}

else {

System.out.println("Invalid command!");

}

}

sc.close();

}

}

OUTPUT

PUSH 10

PUSH 20

PEEK

POP

DISPLAY

Output:

10 pushed to stack

20 pushed to stack

Top element: 20

Popped: 20

Stack: [10]