**Practical 5**

**Name:**Shantanu Sethi

**Roll No.** 163

**Aim: Evaluation of postfix expression and balancing of parenthesis**

To implement applications of stack:

1. Evaluation of postfix expressions

2. Balancing of parenthesis

**Objectives:**

1. Learn how to apply stack logic to evaluate postfix expression and checking parenthesis are balanced in given expression

**Theory:** Stack using Array Implementation

**1. Postfix Expressions Evaluation**

The expression of the form “a b operator” (ab+) i.e., when a pair of operands is followed by an operator. Iterate the expression from left to right and keep on storing the operands into a stack. Once an operator is received, pop the two topmost elements and evaluate them and push the result in the stack again.

1. Create an empty stack that will contain operands.

2. Take one by one token from the left to right.

1. If a token is an operand, push it onto the stack.

1. If token is an operator op

2. Pop the top item from the stack as operand2.

3. Pop again the top item from the stack as operand1.

4. Perform operation operand1 op operand2.

5. Push the result back to stack.

3. When all tokens in input expression are processed stack should contain a single item, which is

the value of expression

**2. Balancing Parenthesis**

The idea is to put all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature. If this holds then pop the stack and continue the iteration. In the end if the stack is empty, it means all brackets are balanced or well-formed. Otherwise, they are not balanced.

1. Declare a character stack S.

2. Now traverse the expression string exp.

1. If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.

2. If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if

the popped character is the matching starting bracket then fine else brackets are not

balanced.

3. After complete traversal, if there is some starting bracket left in the stack then “not balanced”.

**Program:**

package Stack1;

import java.util.Stack;

public class Postfix {

public static void main(String[] args) {

System.out.println("184-ABHINAV SINGH");

String expression = "5 3 + 4 \*"; // Example: (5 + 3) \* 4

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

// Split the expression into tokens based on space

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

// Iterate through each token

for (String token : tokens) {

// Check if the token is an operand

if (token.matches("-?\\d+")) {

// Push the operand onto the stack

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

} else {

// The token is an operator

int operand2 = stack.pop(); // Pop the topmost item

int operand1 = stack.pop(); // Pop the next item

// Perform the operation based on the operator

int result = 0;

switch (token) {

case "+":

result = operand1 + operand2;

break;

case "-":

result = operand1 - operand2;

break;

case "\*":

result = operand1 \* operand2;

break;

case "/":

if (operand2 == 0) {

System.out.println("Error: Division by zero.");

return;

}

result = operand1 / operand2;

break;

default:

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

return;

}

// Push the result back onto the stack

stack.push(result);

}

}

// The final result will be the only item left in the stack

int finalResult = stack.pop();

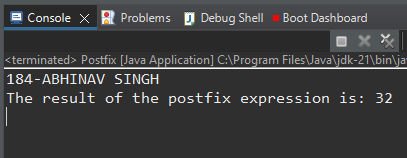
System.out.println("The result of the postfix expression is: " + finalResult);

// TODO Auto-generated method stub

}

}

**OUTPUT:-**

****

**Program:**

**2. Balancing Parenthesis**

package Stack1;

import java.util.Stack;

public class Balancing {

public static void main(String[] args) {

// TODO Auto-generated method stub

System.out.println("184-ABHINAV SINGH");

String expression = "{[()]}"; // Example expression

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

// Flag to indicate if the expression is balanced

boolean isBalanced = true;

// Traverse the expression

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

// If the character is a starting bracket, push it onto the stack

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

stack.push(ch);

}

// If the character is a closing bracket

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

// Check if the stack is empty

if (stack.isEmpty()) {

isBalanced = false; // Unmatched closing bracket

break;

}

// Pop the top item from the stack

char top = stack.pop();

// Check if it matches the corresponding opening bracket

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

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

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

isBalanced = false; // Mismatched brackets

break;

}

}

}

// After complete traversal, check if the stack is empty

if (isBalanced && stack.isEmpty()) {

System.out.println("The expression is balanced.");

} else {

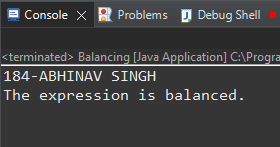
System.out.println("The expression is not balanced.");

}

}

}

**OUTPUT:-**

****

**Conclusion:** Applied stack logic to evaluate postfix expression and checking parenthesis are balanced in given expression.