**Q1: Expression Evaluation (Infix to Postfix Conversion):** Implement a **calculator** that converts **infix expressions to postfix notation** using stacks. Evaluate the postfix expression to return the result. Handle complex expressions with parentheses and operator precedence efficiently.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Infix to Postfix Calculator</title>

<style>

body {

font-family: Arial, sans-serif;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #f5f5f5;

}

.calculator {

background: #fff;

padding: 20px;

border-radius: 10px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

text-align: center;

}

h1 {

margin-bottom: 20px;

}

input {

width: 100%;

padding: 10px;

margin-bottom: 10px;

border: 1px solid #ddd;

border-radius: 5px;

}

button {

padding: 10px 20px;

background: #007BFF;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background: #0056b3;

}

#result {

margin-top: 20px;

font-size: 18px;

color: #333;

}

</style>

</head>

<body>

<div class="calculator">

<h1>Infix to Postfix Calculator</h1>

<input type="text" id="infixExpression" placeholder="Enter infix expression (e.g., 3 + 5 \* (2 - 8))">

<button id="convert">Convert & Evaluate</button>

<div id="result"></div>

</div>

<script>

document.getElementById("convert").addEventListener("click", () => {

const infix = document.getElementById("infixExpression").value;

if (!infix) {

document.getElementById("result").innerText = "Please enter a valid expression.";

return;

}

try {

const postfix = infixToPostfix(infix);

const result = evaluatePostfix(postfix);

document.getElementById("result").innerHTML = `

<p><strong>Postfix:</strong> ${postfix}</p>

<p><strong>Result:</strong> ${result}</p>

`;

} catch (error) {

document.getElementById("result").innerText = "Error: " + error.message;

}

});

function infixToPostfix(expression) {

const precedence = { '+': 1, '-': 1, '\*': 2, '/': 2, '^': 3 };

const isOperator = (ch) => ['+', '-', '\*', '/', '^'].includes(ch);

const output = [];

const stack = [];

const tokens = expression.match(/\d+|[+\*/^()-]/g);

for (const token of tokens) {

if (!isNaN(token)) {

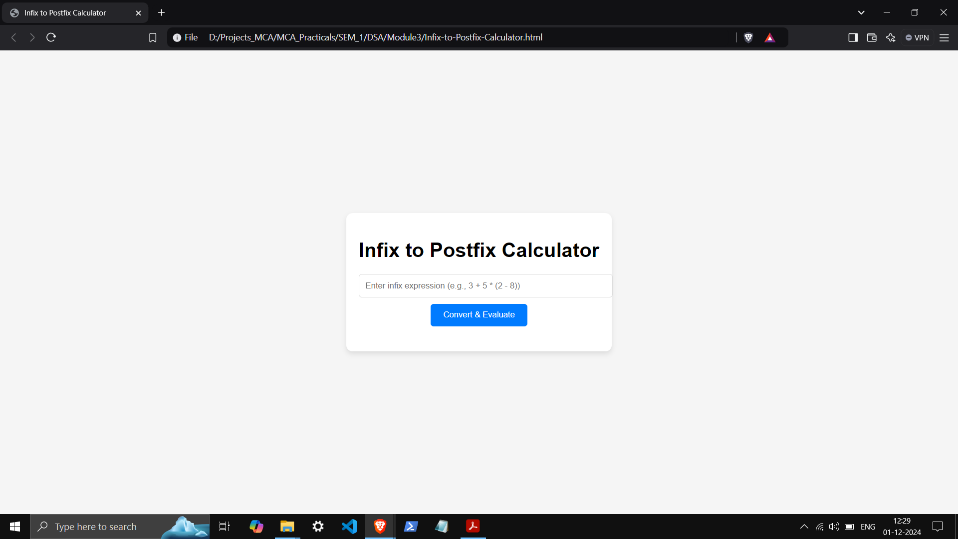
output.push(token); // Operand

} else if (token === '(') {

stack.push(token);

} else if (token === ')') {

while (stack.length && stack[stack.length - 1] !== '(') {

 output.push(stack.pop());

}

stack.pop(); // Remove '('

} else if (isOperator(token)) {

while (stack.length && precedence[stack[stack.length - 1]] >= precedence[token]) {

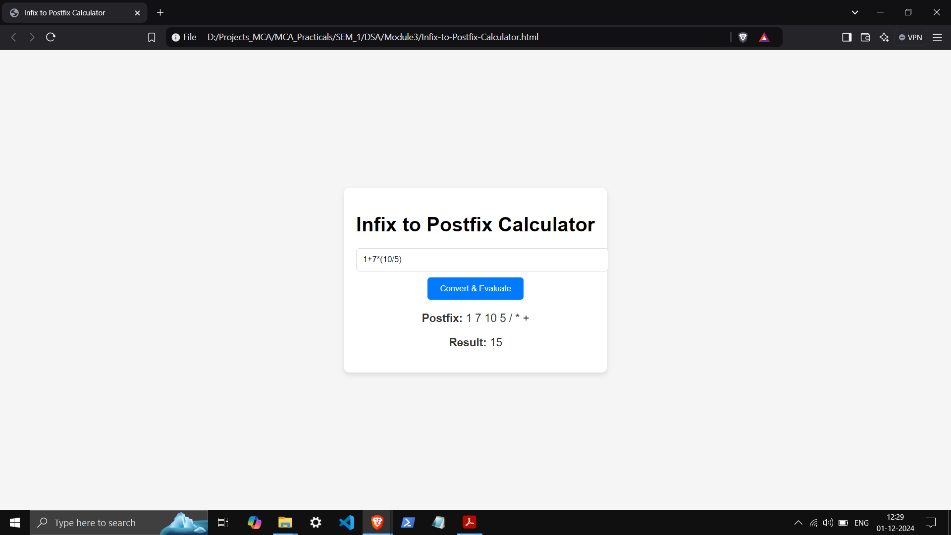
output.push(stack.pop());

}

stack.push(token);

}

}

 while (stack.length) {

output.push(stack.pop());

}

return output.join(' ');

}

function evaluatePostfix(expression) {

const stack = [];

const tokens = expression.split(' ');

for (const token of tokens) {

if (!isNaN(token)) {

stack.push(Number(token)); // Operand

} else {

const b = stack.pop();

const 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;

case '^': stack.push(Math.pow(a, b)); break;

default: throw new Error("Invalid operator: " + token);

}

}

}

return stack.pop();

}

</script>

</body>

</html>