**Institute of Computer Technology**

**B. Tech. Computer Science and Engineering**

**Sub: DS**

**Course Code: 2CSE302**

**Practical – 4**

**Name: Jaymin Gondaliya**

**Enrollment No: 23162171007**

**Sem - 3**

**Branch: CS**

**Class: A**

**Batch: 32**

**Problem Definition-1:** Stack infix to postfix

Parishram is a 7th semester, who is studying at GUNI-ICT. During his “Compiler Design” course, his course faculty explained him that compiler work differently while it does evaluation of an expression due to below reasons:

Infix expressions are readable and solvable by humans because of easily distinguishable order of operators, but compiler doesn’t have integrated order of operators. To avoid this traversing, Infix expressions are converted to postfix expression before evaluation.

Make a program to convert infix expression into postfix using stack.

**Code:**

#include <stdio.h>

#define max 100

int top = -1, a[max]; *// Stack to hold operators with `top` initialized to -1 (indicating an empty stack)*

*// Function to push an operator onto the stack*

void push(char x)

{

    a[++top] = x; *// Increment `top` and then store `x` at that position*

}

*// Function to pop an operator from the stack*

char pop()

{

    if (top == -1)

        return -1; *// Return -1 if the stack is empty*

    else

        return a[top--]; *// Return the top element and then decrement `top`*

}

*// Function to define precedence of operators*

int prcd(char c)

{

    if (c == '(')

        return 0; *// '(' has the lowest precedence*

    else if (c == '+' || c == '-')

        return 1; *// '+' and '-' have the same precedence*

    else if (c == '\*' || c == '/')

        return 2; *// '\*' and '/' have the same precedence, which is higher than '+' and '-'*

}

*// Function to convert infix expression to postfix*

int infixtopostfix(char infix[max], char postfix[max])

{

    char temp, x;

    int i = 0, j = 0;

    while (infix[i] != '\0') *// Loop through the entire infix expression*

    {

        temp = infix[i];

        if (isalnum(temp)) *// If the character is an operand (number/letter)*

        {

            postfix[j++] = temp; *// Add it directly to the postfix expression*

        }

        else if (temp == '(')

        {

            push(temp); *// Push '(' onto the stack*

        }

        else if (temp == ')')

        {

            while ((x = pop()) != '(') *// Pop from the stack until '(' is found*

            {

                postfix[j++] = x; *// Add the popped operators to postfix*

            }

        }

        else *// If the character is an operator*

        {

            while (prcd(a[top]) >= prcd(temp)) *// Check precedence and pop operators from stack if needed*

            {

                postfix[j++] = pop(); *// Add the popped operator to postfix*

            }

            push(temp); *// Push the current operator onto the stack*

        }

        i++;

    }

    while (top != -1) *// Pop any remaining operators in the stack*

    {

        postfix[j++] = pop();

    }

    postfix[j] = '\0'; *// Null-terminate the postfix expression*

}

int main()

{

    char infix[max], postfix[max];

    printf("Enter the infix expression\n");

    gets(infix); *// Take infix expression input*

    printf("The infix expression is %s\n", infix);

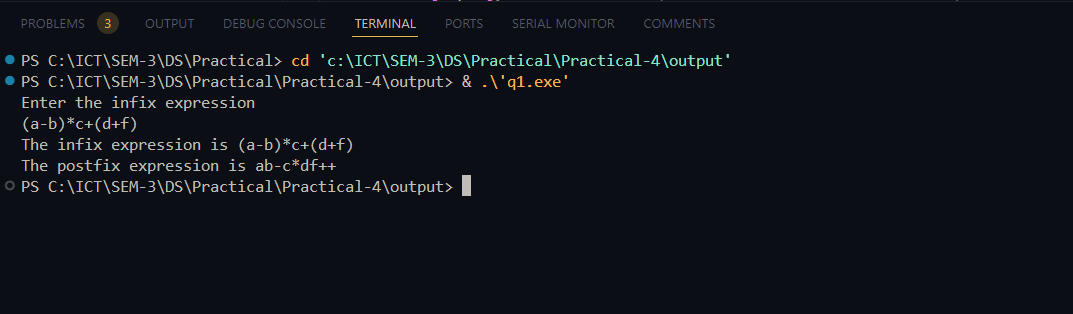
    infixtopostfix(infix, postfix); *// Convert infix to postfix*

    printf("The postfix expression is %s\n", postfix); *// Print the resulting postfix expression*

    return 0;

}

**Output:**

****