

**WEEK-9**

Date:10-09-2025

**List of programs:**

1. Write a C program to convert an infix expression into postfix expression using Stacks.
2. Write a C program to evaluate postfix expression using Stacks.

- 1. Aim:** Write a C program to convert an infix expression into postfix expression using Stacks.

**Program:**

```
#include<stdio.h>

#include<string.h>

#include <ctype.h>

#define SIZE 50

char s[SIZE];

int top=-1;

void push(char ele)

{

    s[++top]=ele;

}

char pop()

{

    return(s[top--]);

}

int priority(char ele)

{

    switch(ele)

    {

        case '$': return 0;
```

```
case '(': return 1;
case '+': return 2;
case '-': return 2;
case '*': return 3;
case '/': return 3;
}
}
void main()
{
    char infix[50],postfx[50],ch;
    int i=0,k=0;
    printf("Read the Infix Expression : ");
    scanf("%s",infix);
    push('$');
    while( (ch=infix[i++]) != '\0')
    {
        if( ch == '(')
            push(ch);
        else
            if(isalnum(ch))
                postfx[k++]=ch;
            else
                if( ch == ')')
                {
                    while( s[top] != '(')
                        postfx[k++]=pop();
                    pop();
                }
    }
```

```
        else
        {
            while( priority(s[top]) >= priority(ch) )
                postfix[k++]=pop();
            push(ch);
        }
    }
    while( s[top] != '$')
        postfix[k++]=pop();
    postfix[k]='\0';
    printf("Given Infix Expn: %s \n Postfix Expn: %s\n",infx,postfx);
}
```

**Output:**

```
Read the Infix Expression : a+b*c/d
Given Infix Expn: a+b*c/d
Postfix Expn: abc*d/+

=== Code Exited With Errors ===
```

**2. Aim:** Write a C program to evaluate postfix expression using Stacks.

**Program:**

```
#include<stdio.h>

#include <stdlib.h>

#include<ctype.h>

int s[50];

int top=-1;

void push(int elem)

{

    s[++top]=elem;

}

int pop()

{

    return(s[top--]);

}

void main()

{

    char pofx[50],ch;

    int i=0,op1,op2;

    printf("\n\nEnter the Postfix Expression ");

    scanf("%s",pofx);

    while( (ch=pofx[i++]) !='\0')

    {

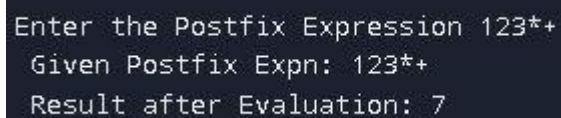
        if(isdigit(ch))

            push(ch-'0');
```

```
else
{
    op2=pop();
    op1=pop();
    switch(ch)
    {
        case '+':push(op1+op2);
                break;
        case '-':push(op1-op2);
                break;
        case '*':push(op1*op2);
                break;
        case '/':push(op1/op2);
                break;
    }
}
}

printf(" Given Postfix Expn: %s\n",pofx);
printf(" Result after Evaluation: %d\n",s[top]);
}
```

### Output:

A screenshot of a terminal window with a dark background and light-colored text. It shows the output of a program: 'Enter the Postfix Expression 123\*+', 'Given Postfix Expn: 123\*+', and 'Result after Evaluation: 7'.

```
Enter the Postfix Expression 123*+
Given Postfix Expn: 123*+
Result after Evaluation: 7
```

**Inferences:**

- The working principle for infix to postfix conversion is Converts an **infix expression** (e.g.,  $A + B$ ) to **postfix** ( $A B +$ ) using a stack. Operators are pushed to the stack; operands go directly to the output. Operator precedence and parentheses are managed via stack operations.
- Postfix evaluation is **fast ( $O(n)$ )**, **simple**, and **stack-based**, making it ideal for **expression evaluation in compilers and calculators**. It avoids complexities of parentheses and operator precedence, unlike infix evaluation.