EXPERIMENT NO. 3

EVALUATION OF A POSTFIX EXPRESSION

Aim: Write A Program For Evaluation Of Postfix Expression.

Theory: Infix, Postfix & Prefix Are 3 Different But Equivalent Notations Of Writing Algebraic Expressions. Postfix Notation Was Developed By **Jan Lukasiewicz** Who Was A **Polish Logician, Mathematician & Philosopher**. His Aim Was To Develop A Parenthesis – Free Prefix Notation & A Postfix Notation, Which Is Better Known As Reverse Polish Notation Or RPN.

DEFINITION:

Postfix expression: The Expression of the form a b op. When an operator is followed for every pair of operands.

The Expressions Written In Postfix Form Are Evaluated Faster Compared To Infix Notation As Parenthesis Are Not Required In Postfix.

Why postfix representation of the expression?

The compiler scans the expression either from left to right or from right to left. Consider the below expression: a op1 b op2 c op3 d

If op1 = +, op2 = *, op3 = +

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan.

The repeated scanning makes it **very in-efficient**. It is better to convert the expression to postfix (or prefix) form before evaluation.

The corresponding expression in postfix form is: abc*d++.The postfix expressions can be evaluated easily using a stack.

Algorithm:

1) Create a stack to store operands (or values).

2) Scan the given expression and do following for every scanned element. a. If the element is a number, push it into the stack

b. If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack.

3) when the expression is ended, the number in the stack is the final answer.

Example:

Let The Given Expression Be "2 3 1 * + 9 -". Scanning All Elements One By One.

1) Scan '2'; it's a number, so push it to stack. Stack contains '2'

2) Scan '3', again a number, push it to stack, stack now contains '2 3′ (from bottom to top)

3) Scan '1', again a number, push it to stack, stack now contains '2 3 1′

4) Scan '*', it's an operator, pop two operands from stack, apply the * operator on operands, we get 3*1 which results in 3. We push the result '3' to stack. Stack now becomes '2 3′.

5) Scan '+', it's an operator, pop two operands from stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result '5' to stack. Stack now becomes '5'.

6) Scan '9', it's a number, we push it to the stack. Stack now becomes '5 9′. 7) Scan '-', it's an operator, pop two operands from stack, apply the – operator on operands, we get 5 – 9 which results in -4. We push the result '-4′ to stack. Stack now becomes '-4′.

8) There are no more elements to scan; we return the top element from stack (which is the only element left in stack). Is the Answer (-4).

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 100
int stack[MAX];
char postfix[MAX];
int top = -1;
void push(int);
int pop();
```

```
int post_eval();
void push(int val)
{
if(top == MAX - 1)
{
printf("Stack Overflow");
}
top = top + 1;
stack[top] = val;
}
int pop()
{
int val;
if(top == -1)
{
printf("Stack underflow");
exit(1);
}
val = stack[top];
top = top - 1;
return val;
}
int post_eval()
{
int i,a,b;
for(i=0;i<strlen(postfix);i++)
{
//if the symbol is an operand
if(postfix[i] >= '0' && postfix[i] <= '9')
{
push(postfix[i] - '0');
}
else
{
//pop the topmost symbols
a = pop();
b = pop();
switch(postfix[i])
{
case '+':
push(b+a);
break;
```

```
case '-':
push(b-a);
break;
case '*':
push(b*a);
break;
case '/':
push(b/a);
break;
case '^':
push(pow(b,a));
break;
}
}
}
return pop();
}
int main()
{
int result;
printf("Enter the postfix expression: ");
gets(postfix);
result = post_eval();
printf("The result obtained after postfix evaluation is: ");
printf("%d\n",result);
return 0;
}
```

Output:

```
Enter the postfix expression: 23*54**+9-
The result obtained after postfix evaluation is: 17
```