NAME: BHOOMIKA S. SURVE ROLL NO.: 48 DIV.: S3

Experiment no 3: Evaluation of postfix Expression using stack ADT

Aim: Implementation of Evaluation of Postfix Expression using stack ADT

Objective:

1) Understand the use of stack
2) Understand importing an ADT in an application program
3) Understand the instantiation of stack ADT in an application Program
4) Understand how the member function of an ADT are accessed in an application program

Theory:

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.

Follow the steps mentioned below to evaluate postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
- If the element is a number, push it into the stack.
- If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

Algorithm:

Step 1 − scan the expression from left to right

Step 2 − if it is an operand push it to stack

Step 3 − if it is an operator pull operand from

stack and perform operation

Step 4 − store the output of step 3, back to stack

Step 5 − scan the expression until all operands

are consumed

Step 6 − pop the stack and perform operation

Code :

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
int stack[20];
int top = -1;

void push(int x)
{
   stack[++top] = x;
}
```

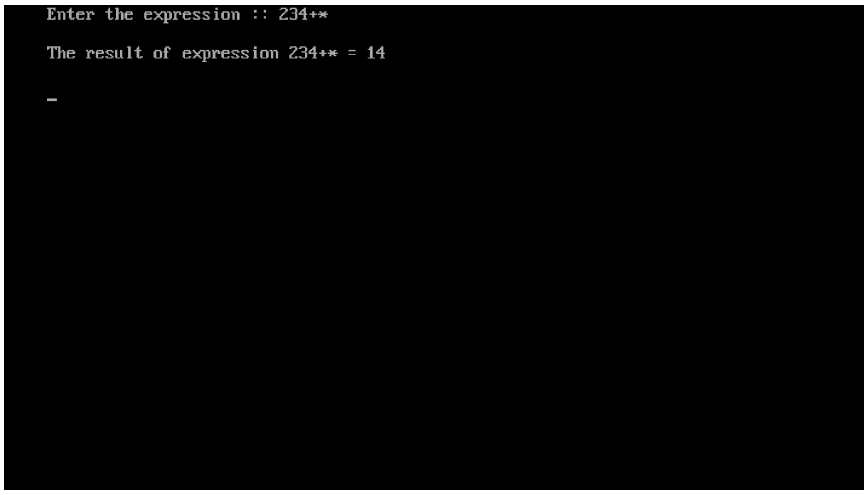
```
int pop()
{
   return stack[top--];
}

int main()
{

   char exp[20];
   char *e;
   int n1,n2,n3,num;
   clrscr();
   printf("Enter the expression :: ");
   scanf("%s",exp);
   e = exp;
   while(*e != '\0')
   {
      if(isdigit(*e))
      {
         num = *e - 48;
         push(num);
      }
      else
      {
         n1 = pop();
         n2 = pop();
         switch(*e)
         {
```

```
        case '+':
        {
           n3 = n1 + n2;
           break;
        }
        case '-':
        {
           n3 = n2 - n1;
           break;
        }
        case '*':
        {
           n3 = n1 * n2;
           break;
        }
        case '/':
        {
           n3 = n2 / n1;
           break;
        }
        }
        push(n3);
      }
      e++;
   }
   printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
   getch();
   return 0;
}
```

Output:



Conclusion :

In postfix expression, there are no parentheses and therefore the order of evaluation will be determined by the positions of the operators and related operands in the expression.