Experiment No 2: Conversion of Infix to postfix expression using stack ADT

Aim: To convert infix expression to postfix expression using stack ADT
Objective:
1)Understand the use of stack
2)Understand how to import 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

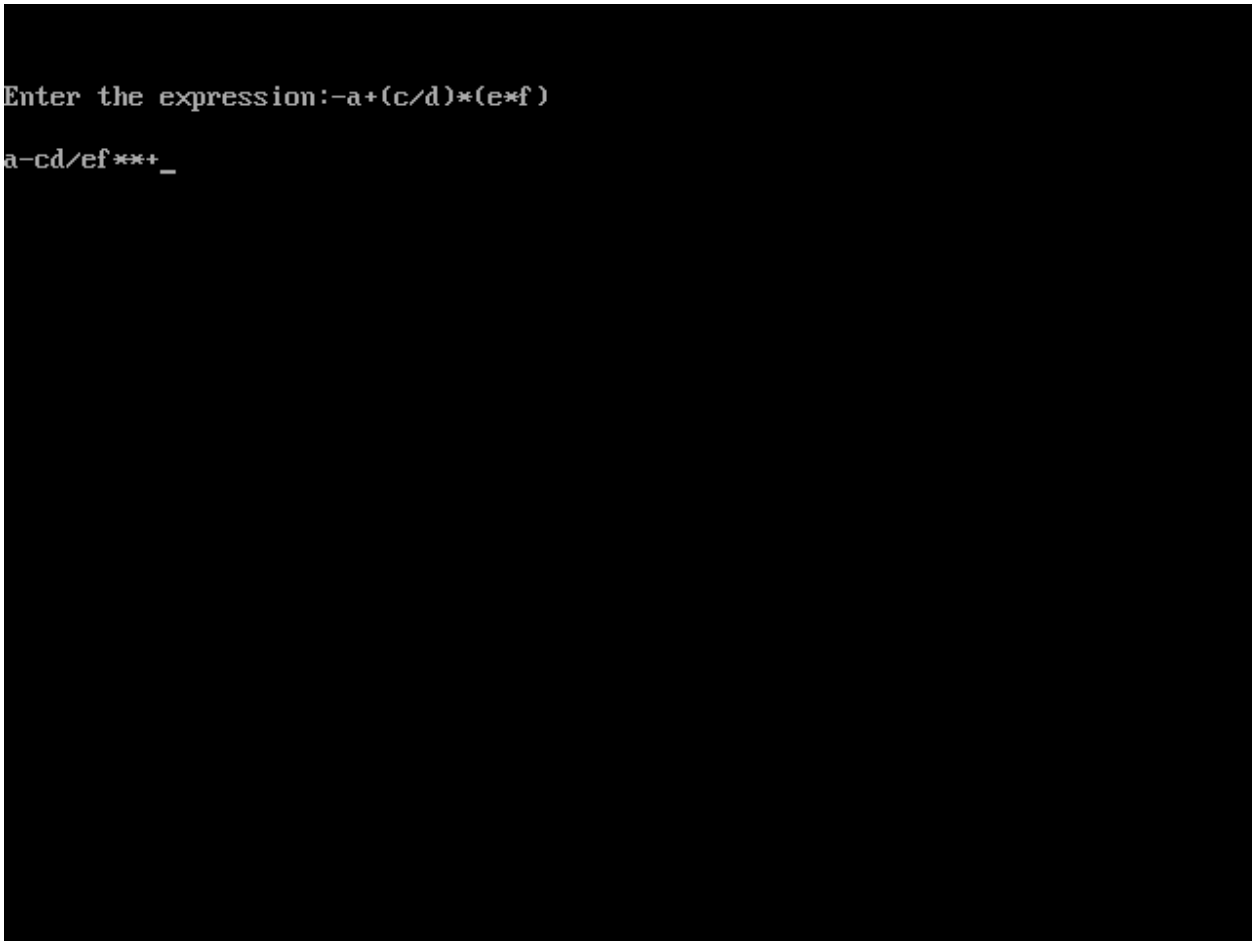
Theory:
Infix, postfix, and prefix notations are three different but equivalent notations of writing algebraic expressions.
Infix :-
While writing an arithmetic expression using infix notation, the operator is placed in between the operands. For example, A+B; here, plus operator is placed between the two operands A and B. Although easy for humans to understand, computers find it difficult to parse as the computer needs a lot of information to evaluate the expression. Information is needed about operator precedence and associativity rules, and brackets which override these rules. So, computers work more efficiently with expressions written using prefix and postfix notations
Postfix :-
Postfix notation is a parenthesis-free prefix notation. In postfix notation, as the name suggests, the operator is placed after the operands. The order of evaluation of a postfix expression is always from left to right. Even brackets cannot alter the order of evaluation. A postfix operation does not even follow the rules of operator precedence. Thus, in a postfix notation, operators are applied to the operands that are immediately left to them. For example, the infix expression (A + B) * C can be written as AB+C* in the postfix notation.
Algorithm:
Step 1: Add “)” to the end of the infix expression
Step 2: Push “(“ on to the stack
Step 3: Repeat until each character in the infix notation is scanned
IF a “(“ is encountered, push it on the stack
IF an operand (whether a digit or a character) is encountered, add it to the postfix expression.
IF a “)” is encountered, then:-
a. Repeatedly pop from stack and add it to the postfix expression until a “(“ is encountered.
b. Discard the “(“ . That is, remove the “(“ from stack and do not add it to the postfix expression
IF an operator 0 is encountered, then
a. Repeatedly pop from stack and add each operator (popped from the stack) to the postfix expression which has the same precedence or a higher precedence than 0
b. Push the operator 0 to the stack
[END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT

Code -

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<conio.h>

char stack[100];
int top = -1;

void push(char x)
{
  stack[++top] = x;
}
 char pop()
  {
  if(top == -1)
     return -1;
      else
      return stack[top--];
  }

  int priority(char x)
  {
  if(x=='(')
   return 0;
  if(x == '+'|| x == '-')
   return 1;
  if(x == '*' || x == '/')
   return 2;
  return 0;
 }


  int main()
  {
   char exp[100];
   char *e, x;
   clrscr();
   printf("Enter the expression:");
   scanf("%s",exp);
   printf("\n");
   e = exp;
   while(*e!='\0')
   {
```

```
    if(isalnum(*e))
     printf("%c",*e);
    else if(*e == '(')
    push(*e);
    else if(*e == ')')
  {
   while((x = pop()) != '(')
     printf("%c", x);
  }
  else
  {
    while(priority(stack[top]) >= priority(*e))
      printf("%c", pop());
    push (*e);
  }
  e++;
 }
  while(top != -1)
  {
    printf("%c", pop());
  }
  getch();

  return 0;
}
```

Output -



Conclusion -

Infix expression is converted to postfix expression using the ADT stack with arrays.