Lab Program - 2

2) WAP to convert a given valid paren-thesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), and / (divide).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#define SIZE 50

char infix_exp[SIZE];
char postfix_exp[SIZE];
char stack[SIZE];
int top = -1;

void push(char x) {
    if (top == SIZE-1) {
        printf("Error: The stack is full.
                \n");
        return;
    }
    top = top + 1;
    stack[top] = x;
}


char pop() {
    char x;
    if (top==-1) {
      printf("Error: There's no element
             in the stack to pop.\n");
      return '0';
    }
    x = stack[top];
    top = top - 1;
    return x;
}

int  IsOperator (char element) {
  if (element == '^' || element == '*' ||
      element == '/' || element == '+' ||
      element == '-')
  return 1;
  else
  return 0;
}

int precedence (char element) {
   if (element == '^')
      return (3);
   else if (element == '*' || element == '/')
      return (2);
   else if (element == '+' || element == '-')
      return (1);
   else
   return (0);
}

void  InfixToPostfix ( char infix_exp[],
                          char postfix_exp[],
                          char stack[]) {
  int =0;
  int j=0;
  char ele;
  char x;
  ele = infix_exp [i];
```

```
while (ele!='\0')
{
    if (isalpha(ele) || isdigit(ele)){
      postfix_exp [j] =ele;
      j++;
    }
    else if (IsOperator(ele)){
       if (top!=-1){
          if (precedence(ele)>precedence
               (stack[top])){
            push(ele);
          }
          else{
             x=pop();
             postfix_exp[j]=x;
             j++;
             push(ele);
          }
       }
       else{
          push(ele);
       }
    }
    else if (ele=='('){
       push(ele);
    }
    else if (ele==')'){
       while (stack[top]!='('){
          x=pop();
          postfix_exp[j]=x;
          j++;
       }
       top--;
    }
```

```
        i++;
        ele = infix_exp[i];
}

    while (top!=-1){
        x=pop();
        postfix_exp[j]=x;
        j++;
    }
    postfix_exp[j]='\0';
}

int main(){

    printf("\n Enter the infix expression
                    :\n");
    gets(infix_exp);
    printf("The expression is :\n");
    puts(infix_exp);

    InfixToPostfix(infix_exp, postfix_
    exp, stack);
    printf("\n The postfix expression
            is as follows:\n");
    puts(postfix_exp);
    return 0;
}.
```