Lab 2: Infix to Postfix Conversion Using Stacks.

Theory:

The Polish Mathematician Han Lukasiewicz suggested a notation called *Polish notation*, which gives two alternatives to represent an arithmetic expression, namely the *postfix* and *prefix* notations. In postfix notation, the operator is written after operands. The advantage of the postfix notation (also called reverse polish notation) are:

1. The need for parenthesis as in an infix expression is overcome in postfix and prefix notations.
2. The priority of operators is no longer relevant.
3. The order of evaluation depends on the position of the operator but not on priority and associativity.

Algorithm to convert infix expression to postfix:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
   3.1 If the ICP>ISP or the stack is empty, push it.
   3.2 Else, pop the operator from the stack until ICP<=ISP. Push the scanned operator to the stack.
4. If the scanned character is an "(" push it onto the stack.
5. If the scanned character is an ")", pop and output from the stack until an "(" is encountered.
6. Repeat steps 2-6 until the infix expression is totally scanned.
7. Pop and output from the stack until it is not empty.

.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 20
struct Stack
{
    char data[MAX];
    int TOP;
};
void push(struct Stack *S, char element)
{
    S->data[++S->TOP] = element;
}
char pop(struct Stack *S)
{
    return S->data[S->TOP--];
}
int priority(char operator)
{
    if (operator == '^')
        return 3;
    if(operator == '/' || operator == '*')
        return 2;
    if(operator == '+' || operator == '-')
        return 1;
    if(operator == '(')
        return 0;
}
int main()
{
    struct Stack SS;
    SS.TOP = -1;
    char infix[MAX], postfix[MAX],aVar;
    printf("ENTER INFIX: ");
    gets(infix);
    int i=0,j=0;
    while(infix[i] != '\0')
    {
        switch(infix[i])
        {
            case '(':
                push(&SS, infix[i]);
                break;
```

```
            case ')':
                aVar = pop(&SS);
                while(aVar != '(')
                {
                    postfix[j++] = aVar;
                    aVar = pop(&SS);
                }
                break;

            case '^':
            case '/':
            case '*':
            case '+':
            case '-':
                if (SS.TOP == -1)
                    push(&SS, infix[i]);
                else if (priority(infix[i]) > priority(SS.data[SS.TOP]))
                    push(&SS,infix[i]);
                else
                {
                    while(priority(infix[i]) <= priority(SS.data[SS.TOP]))
                    {
                        aVar = pop(&SS);
                        postfix[j++] = aVar;
                    }
                    push(&SS,infix[i]);
                }
                break;

            default:
                postfix[j++] = infix[i];
                break;
        }
        i++;
    }
    while(SS.TOP != -1)
    {
        postfix[j++] = pop(&SS);
    }
    printf("The POSTFIX expression is %s\n",postfix);
    return 0;
}
```

Output:

```
ENTER INFIX: (a+b^d-c*e)
The POSTFIX expression is abd^+ce*-
```

Conclusion: Hence, we wrote a program that converts a given infix expression into postfix expression using stacks.