**Q) Write a program to convert a given valid parenthesized 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>

int i = 0, pos = 0, top = -1, length;

char symbol, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();

void push(char symbol);

char pop();

int pred(char symb);

int main()

{

printf("Enter infix expression:\n");

scanf("%s", infix);

infixtopostfix();

printf("\nInfix expression:\n%s", infix);

printf("\nPostfix expression:\n%s", postfix);

return 0;

}

void infixtopostfix() {

length = strlen(infix);

push('#');

while (i < length) {

symbol = infix[i];

switch (symbol) {

case '(':

push(symbol);

break;

case ')':

temp = pop();

while (temp != '(') {

postfix[pos++] = temp;

temp = pop();

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

while (pred(stack[top]) >= pred(symbol)) {

temp = pop();

postfix[pos++] = temp;

}

push(symbol);

break;

default:

postfix[pos++] = symbol;

}

i++;

}

while (top > 0) {

temp = pop();

postfix[pos++] = temp;

}

postfix[pos] = '\0';

}

void push(char symbol) {

top = top + 1;

stack[top] = symbol;

}

char pop() {

return stack[top--];

}

int pred(char symbol) {

int p;

switch (symbol) {

case '^':

p = 3;

break;

case '\*':

case '/':

p = 2;

break;

case '+':

case '-':

p = 1;

break;

case '(':

p = 0;

break;

case '#':

p = -1;

break;

default:

p = -1;

break;

}

return p;

}

Output:  
