```c
/*
4.Design, Develop and Implement a Program in C for converting an Infix Expression to
  Postfix Expression.
  Program should support for both parenthesized & free parenthesized expressions
  with the operators: +, -, *, /, %(Remainder), ^(Power) and alphanumeric operands.
*/

/*
In infix notation, Example: a - b + c, operators are used in between operands.
It is easy for humans to read, write and speak in infix notation. An algorithm to
process infix notation could be difficult and costly in terms of time and space.
*/

/*
The program has 5 functions. First is the main function.
push(char) function is used to push the symbol into stack.
'char' here represents that only one symbol can be pushed into stack at once.
pop() function is used to pop the symbol from the stack.
priority() functon is used to depict the priority of operators.
infixtopostfix() is the actual function that converts infix to postfix.
*/

#include <stdio.h>
#include <stdlib.h>

void push(char);

char pop();

int priority(char);

void infixtopostfix();


int top=-1;               //Indicates that initially the stack is empty.

char infix[30];          // Represents Infix Expression
char postfix[30];       //  Represents postfix Expression
char stack[30];        // The stack that is used for conversion

/*
The stack is used to hold operators rather than numbers.
The purpose of the stack is to reverse the order of the operators in the
expression.
It also serves as storage structure,since no operator can be printed until
both of its operands have appeared.
*/

void main()
{
    printf("Enter the valid Infix expression \n");
    scanf("%s",infix);

    infixtopostfix();                                  //Function call

    printf("The Infix expression is : %s\n",infix);
    printf("The Postfix expression is: %s\n",postfix);
}

// Function to push the symbol on to the stack
void push(char item)
{
    stack[++top]=item;
}

//Function to pop the item from the stack.
```

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

/*
Function to check the priority of the operators.
Higher priorities operators will be executed first in conversion.
*/

int priority(char symb)
{
    int p;                          //Represents priority number.
    switch(symb)
    {
        case '+':
        case '-': p=1;
        break;

        case '*':
        case '/':
        case '%': p=2;
        break;

        case '^': p=3;
        break;

        case '(':
        case ')': p=0;
        break;

        case '#': p=-1;
        break;

    }
    return p;
}

/*
First the infix expression is scanned from first to last.
In case of '(' , we push all symbols inside the '(' to top of stack.
In case of  ')', we pop symbol from top of stack.
We continue to pop all symbols from top of stack until '(' is encounterd.
We later store all these popped symbols in postfix.
*/

void infixtopostfix()
{
    int i=0,j=0;
    char symb;
    char temp;

    push('#');                          // Pushing operators into stack

    for(i=0;infix[i]!='\0';i++)             //Scan infix from first to last.
        {
            symb=infix[i];

            switch(symb)
            {
                case '(': push(symb);
                          break;

                case ')': temp=pop();

                while(temp!='(')
```

```
                        {
                            postfix[j++]=temp;
                            temp=pop();
                        }
                        break;

                case'+':
                case'-':
                case'*':
                case'/':
                case'%':
                case'^':
                case'$': while(priority(stack[top])>=priority(symb))
                         {
                             temp=pop();
                             postfix[j++]=temp;

                         }

                push(symb);
                break;

                default: postfix[j++]=symb;
            }
        }

/*
while scanning of all symbols is done but still some symbols are in
stack(top>0), then we pop all the remaining all symbols
from top of stack and store to postfix.
*/
        while(top>0)
            {
                temp=pop();
                postfix[j++]=temp;
            }
            postfix[j]='\0';                     // end string postfix
}
```

OUTPUT:

```
Enter the valid Infix expression
a+b
The Infix expression is : a+b
The Postfix expression is: ab+


Enter the valid Infix expression
a+b(c*d)
The Infix expression is : a+b(c*d)
The Postfix expression is: abcd*+


Enter the valid Infix expression
(A+B/C*(D+C)-F)
The Infix expression is : (A+B/C*(D+C)-F)
The Postfix expression is: ABC/DC+*+F-
```