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

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

char stk[SIZE];
int top = -1;

void push(char elem)   /* Function for push operation */
{
      stk[++top] = elem;
}

char pop()                    /* Function for pop operation */
{
      return stk[top--];
}

int precd(char sym) /* Function for Operator precedence */
{
      switch (sym)
      {
      case '#':
            return 0;
      case '(':
            return 1;

      case '+':
      case '-':
            return 2;
      case '*':
      case '/':
      case '%':
            return 3;
    case '^':
            return 4;
      default: printf("Invalid Operator : %c\n", sym);
            exit(0);
            break;
      }
}

void infix_to_postfix(char *infix, char *postfix) /* Function for
converting infix to postfix */
{
      char sym, ele;
      int i = 0, j = 0;
      push('#');                                    /*Initially push '#'
to empty stk[]*/
      while ((sym = infix[i++] )!= '\0')
      {
```

```
            if (sym == '(')
                  push(sym);                              /*For '(' symbol then
push to stk[]*/
            else if (isalnum(sym))
                  postfix[j++] = sym;                     /*For alphanumeric
symbol then add to postfix[] string */
            else if (sym == ')')
            {
                  while (stk[top] != '(')
                        postfix[j++] = pop();             /*If ')' symbol then
pop until '(' symbol to postfix[] string*/
                  ele = pop();                            /* Remove '(' symbol
*/
            }
            else
            {
                  while (precd(stk[top]) >= precd(sym)) /*if input
precedence is less than stk[] precedence*/
                  {
                        postfix[j++] = pop();             /*pop all input
symbol*/
                  }
                  push(sym);                              /*otherwise push to
stk[]*/
            }
      }
      while (stk[top] != '#')                             /* Pop from stk[]
till empty '#' */
      {
            postfix[j++] = pop();
      }
      postfix[j] = '\0';                                        /* Make
postfix[] as valid string */
}

int main()
{
      char infix[30], postfix[30];
      printf("Enter a valid infix expression\n");
      scanf("%s",infix);
      infix_to_postfix(infix, postfix);
      printf("Postfix expression is %s", postfix);
      return 0;
}
```