```c
#include <stdio.h>
#include <math.h>
#include "stack.h"

#define MAX     50

int IsSpace(char);
int IsOperator(char);
long PostfixEvaluation(const char *);
void InfixToPostfix(const char *, char *);

int main(void)
{
    long lResult;
    char szInfix[MAX] = {0};
    char szPostfix[MAX] = {0};

    printf("Enter infix expression :-\n");
    gets(szInfix);

    printf("\nInfix expression is :-\n%s\n", szInfix);

    InfixToPostfix(szInfix, szPostfix);
    printf("\nPostfix expression is:-\n%s\n", szPostfix);

    lResult = PostfixEvaluation(szPostfix);

    printf("\nResult is %ld\n", lResult);

    return 0;
}

void InfixToPostfix(const char *pszInfix, char *pszPostfix)
{
    int iCounter1;
    int iCounter2;
    int iPriority;
    char chSymbol;

    extern int g_iTop;
    extern int g_Stack[STACK_MAX];

    iCounter2 = 0;
    for(iCounter1 = 0; pszInfix[iCounter1] != '\0'; iCounter1++)
    {
        chSymbol = pszInfix[iCounter1];
        if(IsSpace(chSymbol))
            continue;

        if((iPriority = IsOperator(chSymbol)) != 0)
        {
            while(!IsEmpty() && IsOperator(g_Stack[g_iTop]) >= iPriority)
                pszPostfix[iCounter2++] = Pop();

            Push(chSymbol);
        }
        else if(chSymbol == '(')
            Push(chSymbol);
        else if(chSymbol == ')')
        {
            while((chSymbol = Pop()) != '(')
                pszPostfix[iCounter2++] = chSymbol;
        }
        else
        {
            pszPostfix[iCounter2++] = '[';
            while(1)
```

```
68              {
69                  pszPostfix[iCounter2++] = chSymbol;
70                  chSymbol = pszInfix[++iCounter1];
71                  if(IsSpace(chSymbol) || IsOperator(chSymbol) || chSymbol == '\0' ||
                    chSymbol == ')')
72                      break;
73              }
74              pszPostfix[iCounter2++] = ']';
75              iCounter1--;
76          }
77      }
78
79      while(!IsEmpty())
80          pszPostfix[iCounter2++] = Pop();
81
82      pszPostfix[iCounter2] = '\0';
83  }
84
85  int IsSpace(char chSymbol)
86  {
87      if(chSymbol == ' ' || chSymbol == '\t')
88          return 1;
89      return 0;
90  }
91
92  int IsOperator(char chSymbol)
93  {
94      switch(chSymbol)
95      {
96          case '^':
97              return 3;
98          case '*':
99          case '/':
100             return 2;
101         case '+':
102         case '-':
103             return 1;
104         default:
105             return 0;
106     }
107 }
108
109 long PostfixEvaluation(const char *pszPostfix)
110 {
111     int iCounter1;
112     int iCounter2;
113     char chSymbol;
114
115     long lResult;
116     long lOperand1;
117     long lOperand2;
118
119     char szTemp[50];
120
121     for(iCounter1 = 0; pszPostfix[iCounter1] != '\0'; iCounter1++)
122     {
123         chSymbol = pszPostfix[iCounter1];
124         if(IsOperator(chSymbol))
125         {
126             lOperand2 = Pop();
127             lOperand1 = Pop();
128
129             switch(chSymbol)
130             {
131                 case '+':
132                     lResult = lOperand1 + lOperand2;
133                     break;
```

```
                    case '-':
                        lResult = lOperand1 - lOperand2;
                        break;
                    case '*':
                        lResult = lOperand1 * lOperand2;
                        break;
                    case '/':
                        lResult = lOperand1 / lOperand2;
                        break;
                    case '^':
                        lResult = pow(lOperand1, lOperand2);
                }
                Push(lResult);
            }
            else
            {
                iCounter2 = 0;
                while(1)
                {
                    chSymbol = pszPostfix[++iCounter1];
                    if(chSymbol == ']')
                        break;
                    szTemp[iCounter2++] = chSymbol;
                }

                szTemp[iCounter2] = '\0';
                Push(atol(szTemp));
            }
        }

        return Pop();
    }
```