```
#include <iostream>
#include <stack>
#include <cstring>

using namespace std;

int precedence(char op)
{
    if (op == '+' || op == '-')
    return 1;

    else if (op == '*' || op == '/')
    return 2;

    else if (op == '^')
    return 3;

    else
    return 0;
}

bool isOperator(char c)
{
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

string infixToPostfix(string infix)
{
    stack<char> s;
    string postfix = "";

    for (int i = 0; i < infix.length(); i++)
    {
        char c = infix[i];

        // If the character is an operand, add it to the postfix expression
        if (isalnum(c))
        {
            postfix += c;
        }

        // If the character is '(', push it to stack
        else if (c == '(')
        {
```

```
            s.push(c);
        }

        // If the character is ')', pop and output from the stack
        // until an '(' is encountered
        else if (c == ')')
        {
            while (!s.empty() && s.top() != '(')
            {
                postfix += s.top();
                s.pop();
            }
            s.pop(); // Remove '(' from the stack
        }

        // An operator is encountered
        else if (isOperator(c))
        {
            while (!s.empty() && precedence(s.top()) >= precedence(c))
            {
                postfix += s.top();
                s.pop();
            }
            s.push(c);
        }
    }

    // Pop all the operators from the stack
    while (!s.empty())
    {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}

int main()
{
    string infix;
    cout << "Enter the Infix Expression: ";
    cin >> infix;

    string postfix = infixToPostfix(infix);
```

```
    cout << "Postfix Expression: " << postfix << endl;

    return 0;
}
```