```
#include<iostream>
#include<cstring>
#include<stack>
#include<sstream>
#include<math.h>

using namespace std;

// get weight of operators as per precedence
// higher weight given to operators with higher pre
cedence
// for non operators, return 0
int getWeight(char ch) {
    switch (ch) {
        case '/':
        case '^':
        case 's':
        case '*': return 2;
        case '+':
        case '-': return 1;
        default : return 0;
   }
}

// convert infix expression to postfix using a stac
k
void infix2postfix(char infix[], char postfix[], in
t size) {
    stack<char> s;
    int weight;
    int i = 0;
    int k = 0;
    char ch;
    // iterate over the infix expression
    while (i < size) {
        ch = infix[i];
        if (ch == '(') {
            // simply push the opening parenthesis
            s.push(ch);
            i++;
            continue;
        }
        if (ch == ')') {
            // if we see a closing parenthesis,
            // pop of all the elements and append i
t to
            // the postfix expression till we encou
nter
            // a opening parenthesis
            while (!s.empty() && s.top() != '(') {
                postfix[k++] = s.top();
                s.pop();
            }
            // pop off the opening parenthesis also

            if (!s.empty()) {
                s.pop();
            }
            i++;
            continue;
        }
        weight = getWeight(ch);
        if (weight == 0) {
            // we saw an operand
            // simply append it to postfix expressi
on
            postfix[k++] = ch;
        }
        else {
            // we saw an operator
            if (s.empty()) {
            // simply push the operator onto stack
if
            // stack is empty
            s.push(ch);
            }
            else {
                // pop of all the operators from th
e stack and
                // append it to the postfix express
ion till we
                // see an operator with a lower pre
cedence that
                // the current operator
                while (!s.empty() && s.top() != '('
 &&
                    weight <= getWeight(s.top())) {

                    postfix[k++] = s.top();
                    s.pop();
                }
                // push the current operator onto s
tack
                s.push(ch);
            }
        }
        i++;
    }
    // pop of the remaining operators present in th
e stack
    // and append it to postfix expression
    while (!s.empty()) {
        postfix[k++] = s.top();
        s.pop();
    }
    postfix[k] = 0; // null terminate the postfix e
xpression
}
// main
int main() {
    //char infix[] = "x^2-2*x+3";
    //char infix[] = "(x+3)/(x-3)";
    //char infix[] = "-x^2+3";
    //char infix[] = "s(x)+2*x";
    char infix[] = "x^2-2*x+3";
    int size = strlen(infix);
    char postfix[size];
    infix2postfix(infix,postfix,size);
    cout<<"\nInfix Expression :: "<<infix;
    cout<<"\nPostfix Expression :: "<<postfix;
    cout<<endl;
    cout << "Input each entry literal by literal an
d press ENTER." << endl;
    cout << "You can also substitude <x> with any d
esired value." << endl;
    cout << "At the end input < = > symbol to see t
```

```
he result." << endl;
    cout << "Press <ctl+ c> to quit." << endl;
    stack<double> da_stack;
    while(true){
        string in;
        cin >> in;
        if(in == "+"){
            double a = da_stack.top();
            da_stack.pop();
            double b = da_stack.top();
            da_stack.pop();
            da_stack.push(a + b);
        }
        else if(in == "-"){
            double a = da_stack.top();
            da_stack.pop();
            double b = da_stack.top();
            da_stack.pop();
            da_stack.push(a - b);

        }
        else if(in == "s"){ // s for sin()
            double a = da_stack.top();
            da_stack.pop();
            da_stack.push(sin(a));
        }
        else if(in == "^"){
            double a = da_stack.top();
            da_stack.pop();
            double b = da_stack.top();
            da_stack.pop();
            da_stack.push(pow(b,a));
        }
        else if(in == "*"){
            double a = da_stack.top();
            da_stack.pop();
            double b = da_stack.top();
            da_stack.pop();
            da_stack.push(a * b);
        }
        else if(in == "/"){
            double a = da_stack.top();
```

```
                da_stack.pop();
                double b = da_stack.top();
                da_stack.pop();
                da_stack.push(a / b);
            }
            else if(in == "="){
                if(da_stack.size() != 1){
                    cout << "Bad expression" << endl;
                }
                else{
                    cout << "result: " << da_stack.top(
) << endl;
                    da_stack.pop();
                }
            }
            else{
                double d;
                stringstream ss(in);
                ss >> d;

                da_stack.push(d);
            }
        }
    return 0;
}
```