```cpp
// Simple Stack Example
#include <iostream>
#include "console.h"
#include "stack.h"

using namespace std;
const char SPACE = ' ';

int main() {
    string sentence = "hope is what defines humanity";
    string word;
    Stack<string> wordStack;

    cout << "Original sentence: " << sentence << endl;

    for (char c : sentence) {
       if (c == SPACE and word != "") {
           wordStack.push(word);
           word = ""; // reset
       }
       else {
           word += c;
       }
    }
    if (word != "") {
        wordStack.push(word);
    }

    cout << "     New sentence: ";
    while (!wordStack.isEmpty()) {
        string word = wordStack.pop();
        cout << word << SPACE;
    }
    cout << endl;
    return 0;
}

// Queue mystery
#include <iostream>
#include "console.h"
#include "simpio.h"
#include "queue.h"

using namespace std;

int main() {
    Queue<int> queue;
    // produce: {1, 2, 3, 4, 5, 6}
    for (int i = 1; i <= 6; i++) {
        queue.enqueue(i);
    }
    // original mystery:
    for (int i = 0; i < queue.size(); i++) {
        cout << queue.dequeue() << " ";
    }

    // better idiom:
//    while (!queue.isEmpty()) {
//        cout << queue.dequeue() << " ";
//    }

    // okay idiom if you want to go through original queue elements once:
//    int origQSize = queue.size();
//    for (int i=0; i < origQSize; i++) {
//        int value = queue.dequeue();
//        cout << value << " ";
//        // re-enqueue even values
//        if (value % 2 == 0) {
//            queue.enqueue(value);
//        }
//    }

    cout << queue.toString() << "  size " << queue.size() << endl;
    return 0;
}
```

```
76
77
78 // Postfix arithmetic, implementing +, -, *, /
79
80 #include <iostream>
81 #include "console.h"
82 #include "simpio.h"
83 #include "stack.h"
84
85 using namespace std;
86
87 const string OPERATORS = "+-*x/";
88 const string SEPARATOR = " ";
89
90 // function prototypes
91 double parsePostfix(string expression);
92 string getNextToken(string &expression);
93 void performCalculation(Stack<double> &s, char op);
94
95 int main() {
96     string expression;
97     double answer;
98     do {
99         expression = getLine("Please enter a postfix expression (blank to quit): ");
100        answer = parsePostfix(expression);
101        cout << "The answer is: " << answer << endl << endl;
102    } while (expression != "");
103    return 0;
104 }
105
106 double parsePostfix(string expression) {
107     Stack<double> s;
108     string nextToken;
109
110     while (expression != "") {
111         // gets the next token and removes it from expression
112         nextToken = getNextToken(expression);
113         if (OPERATORS.find(nextToken) == string::npos) {
114             // we have a number
115             double operand = stringToDouble(nextToken);
116             s.push(operand);
117         }
118         else {
119             // we have an operator
120             char op = stringToChar(nextToken);
121             performCalculation(s,op);
122         }
123     }
124     return s.pop();
125 }
126
127 void performCalculation(Stack<double> &s, char op) {
128     double result;
129     double operand2 = s.pop(); // LIFO!
130     double operand1 = s.pop();
131     switch(op) {
132         case '+': result = operand1 + operand2;
133                   break;
134         case '-': result = operand1 - operand2;
135                   break;
136         // allow "*" or "x" for times
137         case '*':
138         case 'x': result = operand1 * operand2;
139                   break;
140         case '/': result = operand1 / operand2;
141                   break;
142     }
143     s.push(result);
144 }
145
146 string getNextToken(string &expression) {
147     // pull out the substring up to the first space
148     // and return the token, removing it from the expression
149     string token;
150     int sepLoc = expression.find(SEPARATOR);
```

```
    if (sepLoc != (int) string::npos) {
        token = expression.substr(0,sepLoc);
        expression = expression.substr(sepLoc+1,expression.size()-sepLoc);
        return token;
    }
    else {
        token = expression;
        expression = "";
        return token;
    }
}

// Match symbols using a stack
#include <iostream>
#include <cmath>
#include "console.h"
#include "simpio.h"  // for getLine
#include "stack.h"

using namespace std;

// function prototypes
string getInput();
bool bracketsMatch(string input, char &unmatched);
bool closingBracketOkay(char c, Stack<char> &s);
void printInputWithLines(string input);
void printSpaces(int n);

// constants
const string openingBrackets = "{[(";
const string closingBrackets = "}])";

int main() {
    string input = getInput();
    char unmatched;
    cout << "The input:" << endl << endl;
    printInputWithLines(input);
    if (bracketsMatch(input,unmatched)) {
        cout << "All brackets match!" << endl;
    } else {
        cout << "The following symbol didn't match: " << unmatched << endl;
    }
    return 0;
}

string getInput() {
    string allInput;
    string oneLine;
    cout << "This program will determine if your brackets are matched." << endl;
    cout << "Please enter program code, followed by a period on a line by itself." << endl;

    oneLine = getLine();
    while (oneLine != ".") {
       allInput += oneLine + "\n";
       oneLine = getLine();
    }
    return allInput;
}

bool bracketsMatch(string input, char &unmatched) {
    Stack<char> s;

    for (char c : input) {
        // look for opening brackets
        if (openingBrackets.find(c) != string::npos) {
            s.push(c);
        }
        else {
            // look for closing brackets
            if (!closingBracketOkay(c,s)) {
                    unmatched = c;
                    return false;
            }
        }
    }
```

```
226     // if we made it through the characters and the stack is not empty,
227     // then we don't have a match :(
228     if (!s.isEmpty()) {
229         // report the top of the stack
230         unmatched = s.pop();
231         return false;
232     }
233     // We made it!
234     return true;
235 }
236
237 bool closingBracketOkay(char c, Stack<char> &s) {
238     char leftSymbol;
239
240     // look for closing brackets
241     if (closingBrackets.find(c) != string::npos) {
242        if (s.isEmpty()) { // uh-oh!
243            return false;
244        }
245        else {
246            leftSymbol = s.pop();
247            // check to see if the symbols are the same type
248            if (openingBrackets.find(leftSymbol) != closingBrackets.find(c)) {
249                // nope! :(
250                return false;
251            }
252        }
253     }
254     return true;
255 }
256
257 void printInputWithLines(string input) {
258     // count the number of lines
259     int numLines = 1;
260     for (char c : input) {
261        if (c == '\n') {
262            numLines++;
263        }
264     }
265     // the log_10(numLines) is the maximum number of digits
266     int maxDigits = log(numLines) / log(10);
267
268     int lineNum = 0; // the first line
269     int numDigits;
270     for (char c : input) {
271        if (lineNum == 0 || c == '\n') {
272            if (lineNum != 0) {
273                cout << endl;
274            }
275            lineNum++;
276            numDigits = log(lineNum) / log(10);
277            printSpaces(maxDigits - numDigits);
278            cout << lineNum << "   ";
279            if (lineNum == 1) {
280                cout << c; // first character is special
281            }
282        }
283        else {
284            cout << c;
285        }
286     }
287     cout << endl;
288 }
289
290 void printSpaces(int n) {
291     for (int i=0; i < n; i++) {
292         cout << " ";
293     }
294 }
```