```
/*
 * Programming Challenge 5
 */
#include <cassert>
#include <iostream>
#include <sstream>
#include <string>
#include <streambuf>
using namespace std;

// CODE HERE -- FUNCTION DECLARATIONS/PROTOTYPES

/*
 * function name: hello
 * parameters: none
 * default arguments: n/a
 * return type: void
 *
 * Display "Hello world!" to stdout (no newline character after)
 */

 void hello();

/*
 * function name: printMessage
 * parameters: string message (call-by-value)
 * default arguments: none
 * return type: void
 *
 * Display message to stdout (no newline character after)
 */

 void printMessage( string message );

/*
 * function name: getAnswer
 * parameters: none
 * default arguments: n/a
 * return type: int
 *
 * Return the value 42
 */

 int getAnswer();

/*
 * function name: findLarger
 * parameters: int n1 (call-by-value), int n2 (call-by-value)
 * default arguments: none
 * return type: int
```

```
 *
 * Return the larger of the two parameter values. Should work correctly
 * if the values are equivalent.
 */

 int findLlarger( int n1, int n2 );

/*
 * function name: getStats
 * parameters: string s (call-by-value), int alphaCount (call-by-reference), int digitCount (call-by-reference)
 * default arguments: none
 * return type: int
 *
 * Return the length of string s. On return alphaCount should contain a count of the number of alphabetic
 * characters in s, digitCount should contain a count of the number of digits in s.
 */

 int getStats( string s, int& alphaCount, int& digitCount );

/*
 * function name: buildMessage
 * parameters: string s (call-by-value), bool allCaps (call-by-value)
 * default arguments: s = "" (empty string), allCaps = false
 * return type: string
 *
 * Return the string "Message: STRING", where STRING is replaced by the value of the parameter s. If allCaps is
 * true, convert s to all uppercase letters before concatenating it with "Message: ". If s is empty string,
 * return "Message: empty".
 */

 string buildMessage( string s = "", bool = false );

/* for unit testing -- do not alter */
template <typename X, typename A>
void btassert(A assertion);
void unittest ();

int main (int argc, char* argv[])
{
    unittest();

    return 0;
}

// CODE HERE -- FUNCTION DEFINITIONS

 void hello()   {

    cout << "Hello world!";

```

```
101    }
102
103    void printMessage( string message )    {
104
105       cout << message;
106
107    }
108
109    int getAnswer()    {
110
111       return 42;
112
113    }
114
115    int findLarger( int n1, int n2 )   {
116
117       if( n1 > n2 )   {   return n1;  } else {    return n2;  }
118
119    }
120
121    int getStats( string s, int& alphaCount, int& digitCount ) {
122
123       int tally   = 0;
124       for( int count = 0; count < s.length(); count++ )   {
125
126           if( isalpha( s.at( count ) ) )  {   tally++;    }
127
128       }
129
130       alphaCount = tally;
131
132       tally = 0;
133       for( int count = 0; count < s.length(); count++ )   {
134
135           if( isdigit( s.at( count ) ) )  {   tally++;    }
136
137       }
138
139       digitCount = tally;
140
141       return s.length();
142
143    }
144
145    string buildMessage( string s, bool allCaps )  {
146
147       if( allCaps )   {
148
149           for( int count = 0; count < s.length(); count++ )   {
150
```

```
            s.at(count) = toupper( s.at(count) );

        }

    }

    if( s.length() == 0 )   {   return "Message: empty";    } else {    return "Message: " + s;     }

 }

/*
 * Unit testing functions. Do not alter.
 */

void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    streambuf* oldCout = cout.rdbuf();
    ostringstream captureCout;
    cout.rdbuf(captureCout.rdbuf());

    hello();
    cout.rdbuf(oldCout);
    try {
        btassert<bool>(captureCout.str() == "Hello world!");
        cout << "Passed TEST 1: hello()\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 hello() #\n";
    }

    captureCout.str("");
    cout.rdbuf(captureCout.rdbuf());
    printMessage("Hello again!");
    cout.rdbuf(oldCout);
    try {
        btassert<bool>(captureCout.str() == "Hello again!");
        cout << "Passed TEST 2: printMessage(\"Hello again!\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 printMessage(\"Hello again!\") #\n";
    }

    try {
        btassert<bool>(getAnswer() == 42);
        cout << "Passed TEST 3: getAnswer()\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 getAnswer() #\n";
    }

    try {
```

```
        btassert<bool>(findLarger(-1, 1) == 1);
        cout << "Passed TEST 4: findLarger(-1, 1)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 findLarger(-1, 1) #\n";
    }

    try {
        btassert<bool>(findLarger(1, -1) == 1);
        cout << "Passed TEST 5: findLarger(1, -1)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 findLarger(1, -1) #\n";
    }

    try {
        btassert<bool>(findLarger(1, 1) == 1);
        cout << "Passed TEST 6: findLarger(1, 1)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 findLarger(1, 1) #\n";
    }

    int alpha=0, digit=0;
    try {
        btassert<bool>(getStats("abc 123", alpha, digit) == 7 && alpha == 3 && digit == 3);
        cout << "Passed TEST 7: getStats(\"abc 123\", alpha, digit)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 getStats(\"abc 123\", alpha, digit) #\n";
    }

    try {
        btassert<bool>(getStats("abc", alpha, digit) == 3 && alpha == 3 && digit == 0);
        cout << "Passed TEST 8: getStats(\"abc\", alpha, digit)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 getStats(\"abc\", alpha, digit) #\n";
    }

    try {
        btassert<bool>(getStats("123", alpha, digit) == 3 && alpha == 0 && digit == 3);
        cout << "Passed TEST 9: getStats(\"123\", alpha, digit)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 getStats(\"123\", alpha, digit) #\n";
    }

    try {
        btassert<bool>(getStats("", alpha, digit) == 0 && alpha == 0 && digit == 0);
        cout << "Passed TEST 10: getStats(\"\", alpha, digit)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 getStats(\"\", alpha, digit) #\n";
    }

    try {
```

```
        btassert<bool>(buildMessage("hello") == "Message: hello");
        cout << "Passed TEST 11: buildMessage(\"hello\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 11 buildMessage(\"hello\") #\n";
    }

    try {
        btassert<bool>(buildMessage("hello", true) == "Message: HELLO");
        cout << "Passed TEST 12: buildMessage(\"hello\", true)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 12 buildMessage(\"hello\", true) #\n";
    }

    try {
        btassert<bool>(buildMessage("HELLO", false) == "Message: HELLO");
        cout << "Passed TEST 13: buildMessage(\"HELLO\", false)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 13 buildMessage(\"HELLO\", false) #\n";
    }

    try {
        btassert<bool>(buildMessage("HELLO", true) == "Message: HELLO");
        cout << "Passed TEST 14: buildMessage(\"HELLO\", true)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 14 buildMessage(\"HELLO\", true) #\n";
    }

    try {
        btassert<bool>(buildMessage() == "Message: empty");
        cout << "Passed TEST 15: buildMessage()\n";
    } catch (bool b) {
        cout << "# FAILED TEST 15 buildMessage() #\n";
    }

    cout << "\nUNIT TEST COMPLETE\n\n";
}

template <typename X, typename A>
void btassert (A assertion)
{
    if (!assertion)
        throw X();
}
```