

```

1  /*
2   * Programming Challenge 5
3   */
4  #include <cassert>
5  #include <iostream>
6  #include <sstream>
7  #include <string>
8  #include <streambuf>
9  using namespace std;
10
11 // CODE HERE -- FUNCTION DECLARATIONS/PROTOTYPES
12
13 /*
14  * function name: hello
15  * parameters: none
16  * default arguments: n/a
17  * return type: void
18  *
19  * Display "Hello world!" to stdout (no newline character after)
20  */
21
22 void hello();
23
24 /*
25  * function name: printMessage
26  * parameters: string message (call-by-value)
27  * default arguments: none
28  * return type: void
29  *
30  * Display message to stdout (no newline character after)
31  */
32
33 void printMessage( string message );
34
35 /*
36  * function name: getAnswer
37  * parameters: none
38  * default arguments: n/a
39  * return type: int
40  *
41  * Return the value 42
42  */
43
44 int getAnswer();
45
46 /*
47  * function name: findLarger
48  * parameters: int n1 (call-by-value), int n2 (call-by-value)
49  * default arguments: none
50  * return type: int

```

```

51  *
52  * Return the larger of the two parameter values. Should work correctly
53  * if the values are equivalent.
54  */
55
56  int findLarger( int n1, int n2 );
57
58  /*
59  * function name: getStats
60  * parameters: string s (call-by-value), int alphaCount (call-by-reference), int digitCount (call-by-reference)
61  * default arguments: none
62  * return type: int
63  *
64  * Return the length of string s. On return alphaCount should contain a count of the number of alphabetic
65  * characters in s, digitCount should contain a count of the number of digits in s.
66  */
67
68  int getStats( string s, int& alphaCount, int& digitCount );
69
70  /*
71  * function name: buildMessage
72  * parameters: string s (call-by-value), bool allCaps (call-by-value)
73  * default arguments: s = "" (empty string), allCaps = false
74  * return type: string
75  *
76  * Return the string "Message: STRING", where STRING is replaced by the value of the parameter s. If allCaps is
77  * true, convert s to all uppercase letters before concatenating it with "Message: ". If s is empty string,
78  * return "Message: empty".
79  */
80
81  string buildMessage( string s = "", bool = false );
82
83  /* for unit testing -- do not alter */
84  template <typename X, typename A>
85  void btassert(A assertion);
86  void unittest ();
87
88  int main (int argc, char* argv[])
89  {
90      unittest();
91
92      return 0;
93  }
94
95  // CODE HERE -- FUNCTION DEFINITIONS
96
97  void hello()  {
98
99      cout << "Hello world!";
100

```

```
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++ )    {
```

```

151         s.at(count) = toupper( s.at(count) );
152     }
153 }
154
155 }
156
157 if( s.length() == 0 ) { return "Message: empty"; } else { return "Message: " + s; }
158
159 }
160
161 /*
162  * Unit testing functions. Do not alter.
163  */
164
165 void unittest ()
166 {
167     cout << "\nSTARTING UNIT TEST\n\n";
168
169     streambuf* oldCout = cout.rdbuf();
170     ostringstream captureCout;
171     cout.rdbuf(captureCout.rdbuf());
172
173     hello();
174     cout.rdbuf(oldCout);
175     try {
176         btassert<bool>(captureCout.str() == "Hello world!");
177         cout << "Passed TEST 1: hello()\n";
178     } catch (bool b) {
179         cout << "# FAILED TEST 1 hello() #\n";
180     }
181
182     captureCout.str("");
183     cout.rdbuf(captureCout.rdbuf());
184     printMessage("Hello again!");
185     cout.rdbuf(oldCout);
186     try {
187         btassert<bool>(captureCout.str() == "Hello again!");
188         cout << "Passed TEST 2: printMessage(\"Hello again!\")\n";
189     } catch (bool b) {
190         cout << "# FAILED TEST 2 printMessage(\"Hello again!\") #\n";
191     }
192
193     try {
194         btassert<bool>(getAnswer() == 42);
195         cout << "Passed TEST 3: getAnswer()\n";
196     } catch (bool b) {
197         cout << "# FAILED TEST 3 getAnswer() #\n";
198     }
199
200     try {

```

```

201         btassert<bool>(findLarger(-1, 1) == 1);
202         cout << "Passed TEST 4: findLarger(-1, 1)\n";
203     } catch (bool b) {
204         cout << "# FAILED TEST 4 findLarger(-1, 1) #\n";
205     }
206
207     try {
208         btassert<bool>(findLarger(1, -1) == 1);
209         cout << "Passed TEST 5: findLarger(1, -1)\n";
210     } catch (bool b) {
211         cout << "# FAILED TEST 5 findLarger(1, -1) #\n";
212     }
213
214     try {
215         btassert<bool>(findLarger(1, 1) == 1);
216         cout << "Passed TEST 6: findLarger(1, 1)\n";
217     } catch (bool b) {
218         cout << "# FAILED TEST 6 findLarger(1, 1) #\n";
219     }
220
221     int alpha=0, digit=0;
222     try {
223         btassert<bool>(getStats("abc 123", alpha, digit) == 7 && alpha == 3 && digit == 3);
224         cout << "Passed TEST 7: getStats(\"abc 123\", alpha, digit)\n";
225     } catch (bool b) {
226         cout << "# FAILED TEST 7 getStats(\"abc 123\", alpha, digit) #\n";
227     }
228
229     try {
230         btassert<bool>(getStats("abc", alpha, digit) == 3 && alpha == 3 && digit == 0);
231         cout << "Passed TEST 8: getStats(\"abc\", alpha, digit)\n";
232     } catch (bool b) {
233         cout << "# FAILED TEST 8 getStats(\"abc\", alpha, digit) #\n";
234     }
235
236     try {
237         btassert<bool>(getStats("123", alpha, digit) == 3 && alpha == 0 && digit == 3);
238         cout << "Passed TEST 9: getStats(\"123\", alpha, digit)\n";
239     } catch (bool b) {
240         cout << "# FAILED TEST 9 getStats(\"123\", alpha, digit) #\n";
241     }
242
243     try {
244         btassert<bool>(getStats("", alpha, digit) == 0 && alpha == 0 && digit == 0);
245         cout << "Passed TEST 10: getStats(\"\", alpha, digit)\n";
246     } catch (bool b) {
247         cout << "# FAILED TEST 10 getStats(\"\", alpha, digit) #\n";
248     }
249
250     try {

```

```

251     btassert<bool>(buildMessage("hello") == "Message: hello");
252     cout << "Passed TEST 11: buildMessage(\"hello\")\n";
253 } catch (bool b) {
254     cout << "# FAILED TEST 11 buildMessage(\"hello\") #\n";
255 }
256
257 try {
258     btassert<bool>(buildMessage("hello", true) == "Message: HELLO");
259     cout << "Passed TEST 12: buildMessage(\"hello\", true)\n";
260 } catch (bool b) {
261     cout << "# FAILED TEST 12 buildMessage(\"hello\", true) #\n";
262 }
263
264 try {
265     btassert<bool>(buildMessage("HELLO", false) == "Message: HELLO");
266     cout << "Passed TEST 13: buildMessage(\"HELLO\", false)\n";
267 } catch (bool b) {
268     cout << "# FAILED TEST 13 buildMessage(\"HELLO\", false) #\n";
269 }
270
271 try {
272     btassert<bool>(buildMessage("HELLO", true) == "Message: HELLO");
273     cout << "Passed TEST 14: buildMessage(\"HELLO\", true)\n";
274 } catch (bool b) {
275     cout << "# FAILED TEST 14 buildMessage(\"HELLO\", true) #\n";
276 }
277
278 try {
279     btassert<bool>(buildMessage() == "Message: empty");
280     cout << "Passed TEST 15: buildMessage()\n";
281 } catch (bool b) {
282     cout << "# FAILED TEST 15 buildMessage() #\n";
283 }
284
285 cout << "\nUNIT TEST COMPLETE\n\n";
286 }
287
288 template <typename X, typename A>
289 void btassert (A assertion)
290 {
291     if (!assertion)
292         throw X();
293 }

```