

```
1  /*
2   * Programming Challenge 18
3   */
4 #include <cassert>
5 #include <iostream>
6 #include <sstream>
7 using namespace std;
8
9 /*
10  * Instances of MyException will be thrown by the
11  * evaluate function, which will be used as part of the unit test.
12  */
13 struct MyException
14 {
15     MyException (string newMessage="error")
16     : message(newMessage)
17     {
18     }
19
20     string message;
21 };
22
23 /*
24  * To be called from main. Runs a series of unit tests on the following
25  * functions:
26  * - reduceByOne
27  * - average
28  * - toString
29  * - getLetterGrade
30  * - checkForDuplicates
31  * - caesarShift
32  */
33 void unittest ();
34
35 /*
36  * Evaluate a boolean expression and throw a MyException
37  * with the message "FAILED" if the boolean expression is false.
38  * @param expression the boolean expression to evaluate
39  * @throw MyException with the message "FAILED" when expression
40  *      is false
41  */
42 void evaluate (bool expression);
43
44 // DO NOT ALTER THE FOLLOWING DECLARATIONS-- DECLARATIONS OF FUNCTIONS TO BE TESTED
45
46 /*
47  * Function to help demonstrate the setup of the unit test for this exercise.
48  * Return an integer, reduced by one.
49  * @param num the integer from which the result value will be produced
50  * @return an integer containing the value of the parameter reduced by one if the parameter
```

```
51 *      value is greater than zero; if the parameter value is zero or less, return
52 *      the original parameter value
53 */
54 int reduceByOne (int num);
55
56 /*
57 * Return the average of the values in an integer array. Size must be >= 1 -- the
58 * function will not check size for validity.
59 * @param theArray an integer array
60 * @param size an unsigned integer containing the number of elements in theArray
61 * @return a double containing the average of the values in theArray
62 */
63 double average (int values[], unsigned int size);
64
65 /*
66 * Return a string containing the contents of a bool array, formatted as
67 * "TRUE-FALSE-TRUE-FALSE" (reflecting the values of the elements in the array).
68 * Size must be >= 1 -- the function will not check size for validity.
69 * @param theArray a bool array
70 * @param size an unsigned integer containing the number of elements in theArray
71 * @return a string containing the contents of theArray formatted as "VAL1, VAL2, VAL3"
72 */
73 string toString (bool theArray[], unsigned int size);
74
75 /*
76 * Return a letter grade, given a student average, based on the traditional 100 point
77 * scale of (90 or above 'A', 80 or above 'B', 70 or above 'C', 60 or above 'D', else 'F').
78 * @param average an int containing an average for which a letter grade will be assigned
79 * @return a char containing the appropriate letter grade, in upper-case, based on the average
80 */
81 char getLetterGrade (int average);
82
83 /*
84 * Return true if an array contains any duplicate values. Size must be >= 1 -- the function
85 * will not check size for validity.
86 * @param theArray an integer array
87 * @param size an unsigned integer containing the number of elements in theArray
88 * @return a bool true if there are any duplicates in theArray, else false
89 */
90 bool checkForDuplicates (int theArray[], unsigned int size);
91
92 /*
93 * Perform a Caesar cipher shift on a string. Only works on capitalized alphabetic characters --
94 * if the shift amount causes a letter to "go over or under" the alphabet, will wrap around to the
95 * "other end" of the alphabet. Shift amount must be between -26 and 26 inclusive.
96 * @param original the string to be encrypted or decrypted
97 * @param shiftAmount an int that determines how much up, or down, to shift
98 *      the letter in original; the caller of the function must use the proper
99 *      positive or negative shift amount to perform an encryption or decryption
100 * @return the encrypted or decrypted string
```

```
101  /*
102  string caesarShift (string original, int shiftAmount);
103
104 int main (int argc, char* argv[])
105 {
106     unittest();
107
108     return 0;
109 }
110
111 void unittest ()
112 {
113     cout << "\nSTARTING UNIT TEST\n\n";
114
115     // CODE HERE
116     // WRITE 1 FUNCTION CALL FOR EACH OF THE 5 FUNCTIONS BEING TESTED
117     // EACH OF THESE FUNCTION CALLS SHOULD RESULT IN A "PASS" BY THE FUNCTION
118     // TOTAL OF 5 CALLS THAT THE FUNCTIONS "PASS"
119
120     cout << "TEST 1A -- reduceByOne: ";
121     try {
122         evaluate( reduceByOne( 0 ) == 0 );
123         cout << "PASSED";
124     } catch( MyException e ) {
125         cout << e.message;
126     }
127     cout << endl;
128
129     cout << "TEST 1B -- average: ";
130     int intTest1BValues[] = { 10, 30, 50, 70, 90 };
131     int intTest1BSize    = 5;
132     try {
133         evaluate( average( intTest1BValues, intTest1BSize ) == 50 );
134         cout << "PASSED";
135     } catch( MyException e ) {
136         cout << e.message;
137     }
138     cout << endl;
139
140     cout << "TEST 1C -- toString: ";
141     bool boolTest1CValues[] = { true, false };
142     int intTest1CValuesSize = 2;
143     try {
144         evaluate( toString( boolTest1CValues, intTest1CValuesSize ) == "TRUE-FALSE" );
145         cout << "PASSED";
146     } catch( MyException e ) {
147         cout << e.message;
148     }
149     cout << endl;
150 }
```

```

151 cout << "TEST 1D -- getLetterGrade: ";
152 try {
153     evaluate( getLetterGrade( 85 ) == 'B' );
154     cout << "PASSED";
155 } catch( MyException e ) {
156     cout << e.message;
157 }
158 cout << endl;
159
160 cout << "TEST 1E -- checkForDuplicates: ";
161 int intTest1EValues[] = { 11, 22, 44, 55, 11 };
162 int intTest1EValuesSize = 5;
163 try {
164     evaluate( checkForDuplicates( intTest1EValues, intTest1EValuesSize ) == true );
165     cout << "PASSED";
166 } catch( MyException e ) {
167     cout << e.message;
168 }
169 cout << endl;
170
171 cout << "TEST 1F -- caesarShift: ";
172 try {
173     evaluate( caesarShift( "ADDRESS", 2 ) == "CFFTGUU" );
174     cout << "PASSED";
175 } catch( MyException e ) {
176     cout << e.message;
177 }
178 cout << endl;
179
180 // CODE HERE
181 // WRITE 1 FUNCTION CALL FOR EACH OF THE 5 FUNCTIONS BEING TESTED
182 // EACH OF THESE FUNCTION CALLS SHOULD RESULT IN A "FAIL" BY THE FUNCTION
183 // TOTAL OF 5 CALLS THAT THE FUNCTIONS "FAIL"
184
185 cout << "TEST 2A -- reduceByOne: ";
186 try {
187     evaluate( reduceByOne( 1 ) == 0 );
188     cout << "PASSED";
189 } catch( MyException e ) {
190     cout << e.message;
191 }
192 cout << endl;
193
194 cout << "TEST 2B -- average: ";
195 int intTest2BValues[] = { 5, 6 };
196 int intTest2BValuesSize = 2;
197 try {
198     evaluate( average( intTest2BValues, intTest2BValuesSize ) == 5.5 );
199     cout << "PASSED";
200 } catch( MyException e ) {

```

```

201     cout << e.message;
202 }
203 cout << endl;
204
205 cout << "TEST 2C -- toString: ";
206 bool boolTest2CValues[] = { true, false, true, false };
207 int intTest2CValuesSize = 2;
208 try {
209     evaluate( toString( boolTest2CValues, intTest2CValuesSize ) == "TRUE-FALSE-TRUE-FALSE" );
210     cout << "PASSED";
211 } catch( MyException e ) {
212     cout << e.message;
213 }
214 cout << endl;
215
216 cout << "TEST 2D -- getLetterGrade: ";
217 try {
218     evaluate( getLetterGrade( 90 ) == 'A' );
219     cout << "PASSED";
220 } catch( MyException e ) {
221     cout << e.message;
222 }
223 cout << endl;
224
225 cout << "TEST 2E -- checkForDuplicates: ";
226 int intTest2EValues[] = { 22, 44, 11, 55, 33 };
227 int intTest2EValuesSize = 5;
228 try {
229     evaluate( checkForDuplicates( intTest2EValues, intTest2EValuesSize ) == false );
230     cout << "PASSED";
231 } catch( MyException e ) {
232     cout << e.message;
233 }
234 cout << endl;
235
236 cout << "TEST 2F -- caesarShift: ";
237 try {
238     evaluate( caesarShift( "ADDRESS", -2 ) == "YBBPCQQ" );
239     cout << "PASSED";
240 } catch( MyException e ) {
241     cout << e.message;
242 }
243 cout << endl;
244 }
245
246 void evaluate (bool expression)
247 {
248     if( !expression )
249         throw MyException( "FAILED" );
250 }
```

```
251
252 // DO NOT ALTER THE FOLLOWING DEFINITIONS -- DEFINITIONS OF FUNCTIONS TO BE TESTED
253 int reduceByOne (int num)
254 {
255     if (num > 0)
256         return num--;
257     else
258         return num;
259 }
260
261 double average (int values[], unsigned int size)
262 {
263     int sum = 0;
264     for (unsigned int i=0; i<size; i++)
265         sum += values[i];
266
267     return static_cast<double>(sum/size);
268 }
269
270 string toString (bool theArray[], unsigned int size)
271 {
272     ostringstream oss;
273     for (unsigned int i=0; i<size; i++)
274     {
275         if (theArray[i])
276             oss << "TRUE";
277         else
278             oss << "FALSE";
279
280         if ((i+1)%2 != 0)
281             oss << ' - ';
282     }
283     return oss.str();
284 }
285
286 char getLetterGrade (int average)
287 {
288     if (average > 90)
289         return 'A';
290     else if (average > 80)
291         return 'B';
292     else if (average > 70)
293         return 'C';
294     else if (average > 60)
295         return 'D';
296     else
297         return 'F';
298 }
299
300 bool checkForDuplicates (int theArray[], unsigned int size)
```

```
301 {  
302     for (int i=0; i<size; i++)  
303     {  
304         for (int j=1; j<size; j++)  
305         {  
306             if (theArray[i] == theArray[j])  
307                 return true;  
308         }  
309     }  
310     return false;  
311 }  
312  
313 string caesarShift (string original, int shiftAmount)  
314 {  
315     if (shiftAmount > -26 && shiftAmount < 26)  
316     {  
317         int nextCharValue = 0;  
318         for (unsigned int i=0; i<original.length(); i++)  
319         {  
320             nextCharValue = toupper(original[i]) + shiftAmount;  
321             if (nextCharValue > 90)  
322                 nextCharValue -= 25;  
323             else if (nextCharValue < 65)  
324                 nextCharValue += 25;  
325             original[i] = nextCharValue;  
326         }  
327     }  
328     return original;  
329 }
```