

```

1  /*
2   * Programming Challenge 6
3   */
4  #include <cassert>
5  #include <iostream>
6  #include <sstream>
7  #include <string>
8  using namespace std;
9
10 /*
11  * Create a string containing the contents of an array, each element
12  * separated by a specified character. For example, if the array contents
13  * are {1, 2, 3} and the separator character is ':', the string returned
14  * will be "1:2:3"
15  * @param values an integer array
16  * @param size the size of the integer array values
17  * @param separator the character to use as the separator character in
18  *     the returned string
19  * @return a string containing the contents of values separated by the
20  *     specified separator character
21  */
22 string prepareForDisplay (int values[], int size, char separator = ',');
23
24 /*
25  * Test to see if an array contains a specified value.
26  * @param values an integer array
27  * @param size the size of the integer array values
28  * @param value the value to search for within the array values
29  * @return true if value is found in values, else false
30  */
31 bool hasValue (int values[], int size, int value);
32
33 /*
34  * Return the value from an array at a specified index.
35  * @param values an integer array
36  * @param size the size of the integer array values
37  * @param index the position in the array from which to return a value
38  * @param error a bool flag that will be set to true if index
39  *     is invalid for the array, else it will be set to false
40  * @return an integer containing the value at the specified index in the
41  *     array and error set to false; if index is invalid, returns 0 and
42  *     sets error to true
43  */
44 int valueAt (int values[], int size, int index, bool& error);
45
46 /*
47  * Return the sum of the values in an integer array.
48  * @param values an integer array
49  * @param size the size of the integer array values
50  * @return an integer containing a sum of the values in the array

```

```

51  */
52  int sum (int values[], int size);
53
54  /*
55   * Swap the positions of two values in an integer array. The two
56   * index values must be valid for the array.
57   * @param values an integer array
58   * @param index1 the position of the first value to be swapped
59   * @param index2 the position of the second value to be swapped
60   */
61  void swapValues (int values[], int index1, int index2);
62
63  /* for unit testing -- do not alter */
64  template <typename X, typename A>
65  void btassert(A assertion);
66  void unittest ();
67
68  int main (int argc, char* argv[])
69  {
70      unittest();
71
72      return 0;
73  }
74
75  // CODE HERE -- FUNCTION DEFINITIONS
76
77  string prepareForDisplay( int values[], int size, char separator ) {
78
79      stringstream ss;
80
81      for( int count = 0; count < size; count++ ) {
82
83          ss << values[count];
84
85          if( (count + 1) != size )    {
86
87              ss << separator;
88
89          }
90
91      }
92
93      return ss.str();
94
95  }
96
97  bool hasValue ( int values[], int size, int value ) {
98
99      for( int count = 0; count < size; count++ ) {
100

```

```

101         if( values[count] == value )    {    return true;    }
102
103     }
104
105     return false;
106
107 }
108
109 int valueAt( int values[], int size, int index, bool& error )    {
110
111     if( index >= size || index < 0 )    {
112
113         error    = true;
114         return 0;
115
116     }
117
118     error        = false;
119     return values[index];
120
121 }
122
123 int sum( int values[], int size )    {
124
125     int intSumOfArrayNumbers    = 0;
126
127     for( int count = 0; count < size; count++ ) {
128
129         intSumOfArrayNumbers += values[count];
130
131     }
132
133     return intSumOfArrayNumbers;
134
135 }
136
137 void swapValues( int values[], int index1, int index2 ) {
138
139     int intTempValue    = 0;
140
141     intTempValue        = values[index1];
142     values[index1]      = values[index2];
143     values[index2]      = intTempValue;
144
145 }
146
147 /*
148  * Unit testing functions. Do not alter.
149  */
150

```

```

151 void unittest ()
152 {
153     cout << "\nSTARTING UNIT TEST\n\n";
154
155     int values[] = {3, 5, 7, 9, 11};
156     int size = 5;
157
158     try {
159         btassert<bool>(prepareForDisplay(values, size) == "3,5,7,9,11");
160         cout << "Passed TEST 1: prepareForDisplay(values, size)\n";
161     } catch (bool b) {
162         cout << "# FAILED TEST 1 prepareForDisplay(values, size) #\n";
163     }
164
165     try {
166         btassert<bool>(prepareForDisplay(values, size, ' ') == "3 5 7 9 11");
167         cout << "Passed TEST 2: prepareForDisplay(values, size, ' ')\n";
168     } catch (bool b) {
169         cout << "# FAILED TEST 2 prepareForDisplay(values, size, ' ') #\n";
170     }
171
172     try {
173         btassert<bool>(prepareForDisplay(values, size, ':') == "3:5:7:9:11");
174         cout << "Passed TEST 3: prepareForDisplay(values, size, ':')\n";
175     } catch (bool b) {
176         cout << "# FAILED TEST 3 prepareForDisplay(values, size, ':') #\n";
177     }
178
179     try {
180         btassert<bool>(hasValue(values, size, 0) == false);
181         cout << "Passed TEST 4: hasValue(values, size, 0)\n";
182     } catch (bool b) {
183         cout << "# FAILED TEST 4 hasValue(values, size, 0) #\n";
184     }
185
186     try {
187         btassert<bool>(hasValue(values, size, 3) == true);
188         cout << "Passed TEST 5: hasValue(values, size, 3)\n";
189     } catch (bool b) {
190         cout << "# FAILED TEST 5 hasValue(values, size, 3) #\n";
191     }
192
193     try {
194         btassert<bool>(hasValue(values, size, 11) == true);
195         cout << "Passed TEST 6: hasValue(values, size, 11)\n";
196     } catch (bool b) {
197         cout << "# FAILED TEST 6 hasValue(values, size, 11) #\n";
198     }
199
200     bool error = true;

```

```

201
202     try {
203         btassert<bool>(valueAt(values, size, 0, error) == 3 && error == false);
204         cout << "Passed TEST 7: valueAt(values, size, 0, error)\n";
205     } catch (bool b) {
206         cout << "# FAILED TEST 7 valueAt(values, size, 0, error) #\n";
207     }
208
209     try {
210         btassert<bool>(valueAt(values, size, 5, error) == 0 && error == true);
211         cout << "Passed TEST 8: valueAt(values, size, 5, error)\n";
212     } catch (bool b) {
213         cout << "# FAILED TEST 8 valueAt(values, size, 5, error) #\n";
214     }
215
216     try {
217         btassert<bool>(valueAt(values, size, 4, error) == 11 && error == false);
218         cout << "Passed TEST 9: valueAt(values, size, 4, error)\n";
219     } catch (bool b) {
220         cout << "# FAILED TEST 9 valueAt(values, size, 4, error) #\n";
221     }
222
223     try {
224         btassert<bool>(sum(values, size) == 35);
225         cout << "Passed TEST 10: sum(values, size)\n";
226     } catch (bool b) {
227         cout << "# FAILED TEST 10 sum(values, size) #\n";
228     }
229
230     try {
231         btassert<bool>(sum(values, 1) == 3);
232         cout << "Passed TEST 11: sum(values, 1)\n";
233     } catch (bool b) {
234         cout << "# FAILED TEST 11 sum(values, 1) #\n";
235     }
236
237     try {
238         btassert<bool>(sum(values, 0) == 0);
239         cout << "Passed TEST 12: sum(values, 0)\n";
240     } catch (bool b) {
241         cout << "# FAILED TEST 12 sum(values, 0) #\n";
242     }
243
244     try {
245         swapValues(values, 0, 4);
246         btassert<bool>(values[0] == 11 && values[4] == 3);
247         cout << "Passed TEST 13: swapValues(values, 0, 4)\n";
248     } catch (bool b) {
249         cout << "# FAILED TEST 13 swapValues(values, 0, 4) #\n";
250     }

```

```

251
252     try {
253         swapValues(values, 1, 3);
254         btassert<bool>(values[1] == 9 && values[3] == 5);
255         cout << "Passed TEST 14: swapValues(values, 1, 3)\n";
256     } catch (bool b) {
257         cout << "# FAILED TEST 14 swapValues(values, 1, 3) #\n";
258     }
259
260     try {
261         swapValues(values, 2, 2);
262         btassert<bool>(values[2] == 7);
263         cout << "Passed TEST 15: swapValues(values, 2, 2)\n";
264     } catch (bool b) {
265         cout << "# FAILED TEST 15 swapValues(values, 2, 2) #\n";
266     }
267
268     cout << "\nUNIT TEST COMPLETE\n\n";
269 }
270
271 template <typename X, typename A>
272 void btassert (A assertion)
273 {
274     if (!assertion)
275         throw X();
276 }

```