

```

1  /*
2   * Programming Challenge 16
3   */
4  #include <cassert>
5  #include <iostream>
6  using namespace std;
7
8  /*
9   * Apply the bubble sort algorithm to sort an array of integers.
10   * @param theArray the integer array to be sorted
11   * @param size an unsigned integer containing the size of theArray
12   */
13  void bubbleSort (int theArray[], unsigned int size);
14
15  /*
16   * Apply the selection sort algorithm to sort an array of integers.
17   * @param theArray the integer array to be sorted
18   * @param size an unsigned integer containing the size of theArray
19   */
20  void selectionSort (int theArray[], unsigned int size);
21
22  /*
23   * Swap the positions of two values in an integer array. The two
24   * index values must be valid for the array.
25   * @param values an integer array
26   * @param index1 the position of the first value to be swapped
27   * @param index2 the position of the second value to be swapped
28   */
29  void swapValues (int values[], int index1, int index2);
30
31  /* for unit testing -- do not alter */
32  template <typename X, typename A>
33  void btassert(A assertion);
34  void unittest ();
35  bool compareArrays (int arrayOne[], int arrayTwo[], unsigned int size);
36
37  int main (int argc, char* argv[])
38  {
39      unittest();
40
41      return 0;
42  }
43
44  // CODE HERE -- FUNCTION DEFINITIONS
45
46  /*
47   * Apply the bubble sort algorithm to sort an array of integers.
48   * @param theArray the integer array to be sorted
49   * @param size an unsigned integer containing the size of theArray
50   */

```

```

51 void bubbleSort (int theArray[], unsigned int size)
52 {
53     bool blnSwapped = false;
54     int intSwapValue = 0;
55
56     do {
57         blnSwapped = false;
58         for( int count = 1; count <= size - 1; count++ )
59         {
60             if( theArray[(count - 1)] > theArray[count] )
61             {
62                 swapValues( theArray, (count - 1), count );
63                 blnSwapped = true;
64             }
65         }
66
67         size--;
68
69     } while( blnSwapped );
70 }
71
72 void selectionSort (int theArray[], unsigned int size)
73 {
74     int intPosition = 0;
75     int intSwapValue = 0;
76
77     for( int count01 = 0; count01 < size; count01++ )
78     {
79         intPosition = count01;
80
81         for( int count02 = count01 + 1; count02 < size; count02++ )
82         {
83             if( theArray[count02] < theArray[intPosition] )
84                 intPosition = count02;
85         }
86
87         swapValues( theArray, count01, intPosition );
88     }
89 }
90
91 void swapValues (int values[], int index1, int index2)
92 {
93     int intSwapValue = 0;
94
95     intSwapValue = values[index1];
96     values[index1] = values[index2];
97     values[index2] = intSwapValue;
98 }
99
100 /*

```

```

101  * Unit testing functions. Do not alter.
102  */
103
104  void unittest ()
105  {
106      cout << "\nSTARTING UNIT TEST\n\n";
107
108      int master[5] = {10, 20, 30, 40, 50};
109      int tester[5] = {50, 40, 30, 20, 10};
110
111      bubbleSort(tester, 5);
112      try {
113          btassert<bool>(compareArrays(master, tester, 5));
114          cout << "Passed TEST 1: bubble sort (50,40,30,20,10) \n";
115      } catch (bool b) {
116          cout << "# FAILED TEST 1: bubble sort (50,40,30,20,10) #\n";
117      }
118
119      bubbleSort(tester, 5);
120      try {
121          btassert<bool>(compareArrays(master, tester, 5));
122          cout << "Passed TEST 2: bubble sort (10,20,30,40,50) \n";
123      } catch (bool b) {
124          cout << "# FAILED TEST 2: bubble sort (10,20,30,40,50) #\n";
125      }
126
127      tester[0]=10, tester[1]=30, tester[2]=20, tester[3]=50, tester[4]=40;
128
129      bubbleSort(tester, 5);
130      try {
131          btassert<bool>(compareArrays(master, tester, 5));
132          cout << "Passed TEST 3: bubble sort (10,30,20,50,40) \n";
133      } catch (bool b) {
134          cout << "# FAILED TEST 3: bubble sort (10,30,20,50,40) #\n";
135      }
136
137      master[0]=10, master[1]=30, master[2]=30, master[3]=50, master[4]=50;
138      tester[0]=50, tester[1]=30, tester[2]=10, tester[3]=30, tester[4]=50;
139
140      bubbleSort(tester, 5);
141      try {
142          btassert<bool>(compareArrays(master, tester, 5));
143          cout << "Passed TEST 4: bubble sort (50,30,10,30,50) \n";
144      } catch (bool b) {
145          cout << "# FAILED TEST 4: bubble sort (50,30,10,30,50) #\n";
146      }
147
148      master[0]=50, master[1]=50, master[2]=50, master[3]=50, master[4]=50;
149      tester[0]=50, tester[1]=50, tester[2]=50, tester[3]=50, tester[4]=50;
150

```

```

151 bubbleSort(tester, 5);
152 try {
153     btassert<bool>(compareArrays(master, tester, 5));
154     cout << "Passed TEST 5: bubble sort (50,50,50,50,50) \n";
155 } catch (bool b) {
156     cout << "# FAILED TEST 5: bubble sort (50,50,50,50,50) #\n";
157 }
158
159 master[0]=10, master[1]=20, master[2]=30, master[3]=40, master[4]=50;
160 tester[0]=50, tester[1]=40, tester[2]=30, tester[3]=20, tester[4]=10;
161
162 selectionSort(tester, 5);
163 try {
164     btassert<bool>(compareArrays(master, tester, 5));
165     cout << "Passed TEST 6: selection sort (50,40,30,20,10) \n";
166 } catch (bool b) {
167     cout << "# FAILED TEST 6: selection sort (50,40,30,20,10) #\n";
168 }
169
170 selectionSort(tester, 5);
171 try {
172     btassert<bool>(compareArrays(master, tester, 5));
173     cout << "Passed TEST 7: selection sort (10,20,30,40,50) \n";
174 } catch (bool b) {
175     cout << "# FAILED TEST 7: selection sort (10,20,30,40,50) #\n";
176 }
177
178 tester[0]=10, tester[1]=30, tester[2]=20, tester[3]=50, tester[4]=40;
179
180 selectionSort(tester, 5);
181 try {
182     btassert<bool>(compareArrays(master, tester, 5));
183     cout << "Passed TEST 8: selection sort (10,30,20,50,40) \n";
184 } catch (bool b) {
185     cout << "# FAILED TEST 8: selection sort (10,30,20,50,40) #\n";
186 }
187
188 master[0]=10, master[1]=30, master[2]=30, master[3]=50, master[4]=50;
189 tester[0]=50, tester[1]=30, tester[2]=10, tester[3]=30, tester[4]=50;
190
191 selectionSort(tester, 5);
192 try {
193     btassert<bool>(compareArrays(master, tester, 5));
194     cout << "Passed TEST 9: selection sort (50,30,10,30,50) \n";
195 } catch (bool b) {
196     cout << "# FAILED TEST 9: selection sort (50,30,10,30,50) #\n";
197 }
198
199 master[0]=50, master[1]=50, master[2]=50, master[3]=50, master[4]=50;
200 tester[0]=50, tester[1]=50, tester[2]=50, tester[3]=50, tester[4]=50;

```

```

201
202     selectionSort(tester, 5);
203     try {
204         btassert<bool>(compareArrays(master, tester, 5));
205         cout << "Passed TEST 10: selection sort (50,50,50,50,50) \n";
206     } catch (bool b) {
207         cout << "# FAILED TEST 10: selection sort (50,50,50,50,50) #\n";
208     }
209
210     cout << "\nUNIT TEST COMPLETE\n\n";
211 }
212
213 bool compareArrays (int arrayOne[], int arrayTwo[], unsigned int size)
214 {
215     for (unsigned int i=0; i<size; i++)
216         if (arrayOne[i] != arrayTwo[i])
217             return false;
218
219     return true;
220 }
221
222 template <typename X, typename A>
223 void btassert (A assertion)
224 {
225     if (!assertion)
226         throw X();
227 }

```