

```

1  /*
2   * Programming Challenge 17
3   */
4  #include <cassert>
5  #include <iostream>
6  using namespace std;
7
8  /*
9   * Apply the insertion 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 insertionSort (int theArray[], unsigned int size);
14
15  /*
16   * Apply the shell 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 shellSort (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  void insertionSort (int theArray[], unsigned int size)
47  {
48      int intPosition      = 0;
49
50      for( int count = 0; count < size; count++ )

```

```

51     {
52         intPosition = count;
53
54         while( ( intPosition > 0 ) && ( theArray[intPosition] < theArray[(intPosition - 1)] ) )
55         {
56             swapValues( theArray, intPosition, (intPosition - 1) );
57             intPosition--;
58         }
59     }
60 }
61
62 void shellSort (int theArray[], unsigned int size)
63 {
64     int intPosition    = 0;
65     int intGap         = (size / 2);
66     int intTempValue   = 0;
67
68     while( intGap > 0 )
69     {
70         for( int count = intGap; count < size; count++ )
71         {
72             intTempValue   = theArray[count];
73             intPosition     = count;
74
75             while( ( intPosition >= intGap ) && ( theArray[(intPosition - intGap)] > intTempValue ) )
76             {
77                 theArray[intPosition]   = theArray[(intPosition - intGap)];
78                 intPosition              = (intPosition - intGap);
79             }
80
81             theArray[intPosition]   = intTempValue;
82         }
83
84         intGap /= 2;
85     }
86 }
87
88
89 void swapValues (int values[], int index1, int index2)
90 {
91     int intTempValue   = values[index1];
92
93     values[index1]     = values[index2];
94     values[index2]     = intTempValue;
95 }
96
97
98 /*
99  * Unit testing functions. Do not alter.
100 */

```

```

101
102 void unittest ()
103 {
104     cout << "\nSTARTING UNIT TEST\n\n";
105
106     int master[5] = {10, 20, 30, 40, 50};
107     int tester[5] = {50, 40, 30, 20, 10};
108
109     insertionSort(tester, 5);
110     try {
111         btassert<bool>(compareArrays(master, tester, 5));
112         cout << "Passed TEST 1: insertion sort (50,40,30,20,10) \n";
113     } catch (bool b) {
114         cout << "# FAILED TEST 1: insertion sort (50,40,30,20,10) #\n";
115     }
116
117     insertionSort(tester, 5);
118     try {
119         btassert<bool>(compareArrays(master, tester, 5));
120         cout << "Passed TEST 2: insertion sort (10,20,30,40,50) \n";
121     } catch (bool b) {
122         cout << "# FAILED TEST 2: insertion sort (10,20,30,40,50) #\n";
123     }
124
125     tester[0]=10, tester[1]=30, tester[2]=20, tester[3]=50, tester[4]=40;
126
127     insertionSort(tester, 5);
128     try {
129         btassert<bool>(compareArrays(master, tester, 5));
130         cout << "Passed TEST 3: insertion sort (10,30,20,50,40) \n";
131     } catch (bool b) {
132         cout << "# FAILED TEST 3: insertion sort (10,30,20,50,40) #\n";
133     }
134
135     master[0]=10, master[1]=30, master[2]=30, master[3]=50, master[4]=50;
136     tester[0]=50, tester[1]=30, tester[2]=10, tester[3]=30, tester[4]=50;
137
138     insertionSort(tester, 5);
139     try {
140         btassert<bool>(compareArrays(master, tester, 5));
141         cout << "Passed TEST 4: insertion sort (50,30,10,30,50) \n";
142     } catch (bool b) {
143         cout << "# FAILED TEST 4: insertion sort (50,30,10,30,50) #\n";
144     }
145
146     master[0]=50, master[1]=50, master[2]=50, master[3]=50, master[4]=50;
147     tester[0]=50, tester[1]=50, tester[2]=50, tester[3]=50, tester[4]=50;
148
149     insertionSort(tester, 5);
150     try {

```

```

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

```

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