```
/*
 * Programming Challenge 17
 */
#include <cassert>
#include <iostream>
using namespace std;

/*
 * Apply the insertion sort algorithm to sort an array of integers.
 * @param theArray the integer array to be sorted
 * @param size an unsigned integer containing the size of theArray
 */
void insertionSort (int theArray[], unsigned int size);

/*
 * Apply the shell sort algorithm to sort an array of integers.
 * @param theArray the integer array to be sorted
 * @param size an unsigned integer containing the size of theArray
 */
void shellSort (int theArray[], unsigned int size);

/*
 * Swap the positions of two values in an integer array. The two
 * index values must be valid for the array.
 * @param values an integer array
 * @param index1 the position of the first value to be swapped
 * @param index2 the position of the second value to be swapped
 */
void swapValues (int values[], int index1, int index2);

/* for unit testing -- do not alter */
template <typename X, typename A>
void btassert(A assertion);
void unittest ();
bool compareArrays (int arrayOne[], int arrayTwo[], unsigned int size);

int main (int argc, char* argv[])
{
    unittest();

    return 0;
}

// CODE HERE -- FUNCTION DEFINITIONS

void insertionSort (int theArray[], unsigned int size)
{
    int intPosition     = 0;

    for( int count = 0; count < size; count++ )
```

```
    {
        intPosition = count;

        while( ( intPosition > 0 ) && ( theArray[intPosition] < theArray[(intPosition - 1)] ) )
        {
            swapValues( theArray, intPosition, (intPosition - 1) );
            intPosition--;
        }
    }
}

void shellSort (int theArray[], unsigned int size)
{
    int intPosition     = 0;
    int intGap          = (size / 2);
    int intTempValue    = 0;

    while( intGap > 0 )
    {
        for( int count = intGap; count < size; count++ )
        {
            intTempValue    = theArray[count];
            intPosition     = count;

            while( ( intPosition >= intGap ) && ( theArray[(intPosition - intGap)] > intTempValue ) )
            {
                theArray[intPosition]   = theArray[(intPosition - intGap)];
                intPosition             = (intPosition - intGap);
            }

            theArray[intPosition]   = intTempValue;
        }

        intGap /= 2;
    }

}

void swapValues (int values[], int index1, int index2)
{
    int intTempValue    = values[index1];

    values[index1]      = values[index2];
    values[index2]      = intTempValue;
}


/*
 * Unit testing functions. Do not alter.
 */
```

```
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 {
```

```
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 5: insertion sort (50,50,50,50,50) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 5: insertion sort (50,50,50,50,50) #\n";
    }

    master[0]=10, master[1]=20, master[2]=30, master[3]=40, master[4]=50;
    tester[0]=50, tester[1]=40, tester[2]=30, tester[3]=20, tester[4]=10;

    shellSort(tester, 5);
    try {
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 6: shell sort (50,40,30,20,10) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 6: shell sort (50,40,30,20,10) #\n";
    }

    shellSort(tester, 5);
    try {
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 7: shell sort (10,20,30,40,50) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 7: shell sort (10,20,30,40,50) #\n";
    }

    tester[0]=10, tester[1]=30, tester[2]=20, tester[3]=50, tester[4]=40;

    shellSort(tester, 5);
    try {
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 8: shell sort (10,30,20,50,40) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 8: shell sort (10,30,20,50,40) #\n";
    }

    master[0]=10, master[1]=30, master[2]=30, master[3]=50, master[4]=50;
    tester[0]=50, tester[1]=30, tester[2]=10, tester[3]=30, tester[4]=50;

    shellSort(tester, 5);
    try {
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 9: shell sort (50,30,10,30,50) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 9: shell sort (50,30,10,30,50) #\n";
    }

    master[0]=50, master[1]=50, master[2]=50, master[3]=50, master[4]=50;
    tester[0]=50, tester[1]=50, tester[2]=50, tester[3]=50, tester[4]=50;

    shellSort(tester, 5);
```

```
    try {
        btassert<bool>(compareArrays(master, tester, 5));
        cout << "Passed TEST 10: shell sort (50,50,50,50,50) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 10: shell sort (50,50,50,50,50) #\n";
    }

    cout << "\nUNIT TEST COMPLETE\n\n";
}

bool compareArrays (int arrayOne[], int arrayTwo[], unsigned int size)
{
    for (unsigned int i=0; i<size; i++)
        if (arrayOne[i] != arrayTwo[i])
            return false;

    return true;
}

template <typename X, typename A>
void btassert (A assertion)
{
    if (!assertion)
        throw X();
}
```