```
/*
 * Programming Challenge 6
 */
#include <cassert>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

/*
 * Create a string containing the contents of an array, each element
 * separated by a specified character. For example, if the array contents
 * are {1, 2, 3} and the separator character is ':', the string returned
 * will be "1:2:3"
 * @param values an integer array
 * @param size the size of the integer array values
 * @param separator the character to use as the separator character in
 *        the returned string
 * @return a string containing the contents of values separated by the
 *         specified separator character
 */
string prepareForDisplay (int values[], int size, char separator = ',');

/*
 * Test to see if an array contains a specified value.
 * @param values an integer array
 * @param size the size of the integer array values
 * @param value the value to search for within the array values
 * @return true if value is found in values, else false
 */
bool hasValue (int values[], int size, int value);

/*
 * Return the value from an array at a specified index.
 * @param values an integer array
 * @param size the size of the integer array values
 * @param index the position in the array from which to return a value
 * @param error a bool flag that will be set to true if index
 *        is invalid for the array, else it will be set to false
 * @return an integer containing the value at the specified index in the
 *         array and error set to false; if index is invalid, returns 0 and
 *         sets error to true
 */
int valueAt (int values[], int size, int index, bool& error);

/*
 * Return the sum of the values in an integer array.
 * @param values an integer array
 * @param size the size of the integer array values
 * @return an integer containing a sum of the values in the array
```

```
 */
int sum (int values[], 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 ();

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

    return 0;
}

// CODE HERE -- FUNCTION DEFINITIONS

string prepareForDisplay( int values[], int size, char separator )  {

    stringstream ss;

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

        ss << values[count];

        if( (count + 1) != size )   {

            ss << separator;

        }

    }

    return ss.str();

}

bool hasValue ( int values[], int size, int value ) {

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

```

```
        if( values[count] == value )    {   return true;    }

    }

    return false;

}

int valueAt( int values[], int size, int index, bool& error )   {

    if( index >= size || index < 0 )    {

        error   = true;
        return 0;

    }

    error       = false;
    return values[index];

}

int sum( int values[], int size )   {

    int intSumOfArrayNumbers    = 0;

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

        intSumOfArrayNumbers += values[count];

    }

    return intSumOfArrayNumbers;

}

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

    int intTempValue    = 0;

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

}

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

```

```
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;
```

```
    try {
        btassert<bool>(valueAt(values, size, 0, error) == 3 && error == false);
        cout << "Passed TEST 7: valueAt(values, size, 0, error)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 valueAt(values, size, 0, error) #\n";
    }

    try {
        btassert<bool>(valueAt(values, size, 5, error) == 0 && error == true);
        cout << "Passed TEST 8: valueAt(values, size, 5, error)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 valueAt(values, size, 5, error) #\n";
    }

    try {
        btassert<bool>(valueAt(values, size, 4, error) == 11 && error == false);
        cout << "Passed TEST 9: valueAt(values, size, 4, error)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 valueAt(values, size, 4, error) #\n";
    }

    try {
        btassert<bool>(sum(values, size) == 35);
        cout << "Passed TEST 10: sum(values, size)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 sum(values, size) #\n";
    }

    try {
        btassert<bool>(sum(values, 1) == 3);
        cout << "Passed TEST 11: sum(values, 1)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 11 sum(values, 1) #\n";
    }

    try {
        btassert<bool>(sum(values, 0) == 0);
        cout << "Passed TEST 12: sum(values, 0)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 12 sum(values, 0) #\n";
    }

    try {
        swapValues(values, 0, 4);
        btassert<bool>(values[0] == 11 && values[4] == 3);
        cout << "Passed TEST 13: swapValues(values, 0, 4)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 13 swapValues(values, 0, 4) #\n";
    }
```

```
    try {
        swapValues(values, 1, 3);
        btassert<bool>(values[1] == 9 && values[3] == 5);
        cout << "Passed TEST 14: swapValues(values, 1, 3)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 14 swapValues(values, 1, 3) #\n";
    }

    try {
        swapValues(values, 2, 2);
        btassert<bool>(values[2] == 7);
        cout << "Passed TEST 15: swapValues(values, 2, 2)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 15 swapValues(values, 2, 2) #\n";
    }

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

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