```
/*
 * Programming Project 1
 * Programmer:  Chad Philip Johnson
 * Submission Date:  February 15th, 2013
 * Course:  CSCI21 - Programs & Algorithms II
 */

#include <cassert>
#include <climits>
#include <iostream>
#include <string>
using namespace std;

/*
 * Evaluates the number of alphabetical characters and numerical
 * characters within a string.
 * @param theString String object to be evaluated; passed by value.
 * @param alpha Integer value to contain the counted number of
 *      alphabetical characters within the string; passed by
 *      reference.
 * @param num Integer value to contain the counted number of
 *      numerical characters within the string; passed by reference.
 */
void countCharacters( string theString, int& alpha, int& num );

/*
 * Changes the characters of a string by alternating between uppercase
 *      and lowercase.
 * @param theString String object to be evaluated; passed by value.
 * @return The modified string object.
 */
string upAndDown( string theString );

/*
 * Counts the number of words contained within a string.  Words are
 *      distinguished within the string by the appearance of a space
 *      character:  ' '.
 * @param theString String object to be evaluated; passed by value.
 * @return Integer value representing the number of words contained
 *      within the string object.
 */
int countWords( string theString );

/*
 * Computes the average value (rounded down) of the set of integer
 *      values contained within an array.
 * @param values [] Array containing a set of integer values.
 * @param arraySize Integer value representing the size of the array;
 *      passed by value.
 * @return Integer value representing the rounded down average value
```

```
 *      of the set of numbers contained within the array.
 */
int computeAverage( int values [], int arraySize );

/*
 * Finds the smallest value within an array containing a set of integer
 *      values.
 * @param values [] Array containing a set of integer values.
 * @param arraySize Integer value representing the size of the array;
 *      passed by value.
 * @return Integer value representing the smallest value found within
 *      the array.
 */
int findMinValue( int values [], int arraySize );

/*
 * Finds the greatest value within an array containing a set of integer
 *      values.
 * @param values [] Array containing a set of integer values.
 * @param arraySize Integer value representing the size of the array;
 *      passed by value.
 * @return Integer value representing the greatest value found within
 *      the array.
 */
int findMaxValue( int values [], int arraySize );

/* 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;
}

void countCharacters( string theString, int& alpha, int& num )
{
    int intAlphaCount = 0, intNumCount = 0;

    for( int count = 0; count < theString.length(); count++ )
    {
        if( isalpha( theString[count] ) )
            intAlphaCount++;
        else if( isdigit( theString[count] ) )
            intNumCount++;
    }

```

```
101         alpha = intAlphaCount;
102         num = intNumCount;
103 
104     }
105 
106     string upAndDown( string theString )
107     {
108         for( int count = 0; count < theString.length(); count++ )
109         {
110             if( (count % 2) == 0 )
111                 theString[count] = toupper( theString[count] );
112             else
113                 theString[count] = tolower( theString[count] );
114         }
115 
116         return theString;
117 
118     }
119 
120     int countWords( string theString )
121     {
122 
123         int intWordCount = 0;
124 
125         if( theString.length() != 0 )
126         {
127             intWordCount++;
128 
129             for( int count = 0; count < theString.length(); count++ )
130             {
131                 if( theString[count] == ' ' )
132                     intWordCount++;
133             }
134         }
135 
136         return intWordCount;
137 
138     }
139 
140     int computeAverage( int values [], int arraySize )
141     {
142         int intAverage = 0;
143 
144         for( int count = 0; count < arraySize; count++ )
145         {
146             intAverage += values[count];
147         }
148 
149         return (intAverage / arraySize);
150     }
```

```
int findMinValue( int values [], int arraySize )
{
    int intMinimumValue = values[0];

    for( int count = 0; count < (arraySize - 1); count++ )
    {
        if( values[(count + 1)] < intMinimumValue )
            intMinimumValue = values[(count + 1)];
    }

    return intMinimumValue;
}

int findMaxValue( int values [], int arraySize )
{
    int intMaximumValue = values[0];

    for( int count = 0; count < (arraySize - 1); count++ )
    {
        if( values[(count + 1)] > intMaximumValue )
            intMaximumValue = values[(count + 1)];
    }

    return intMaximumValue;
}

/*
 * Unit testing functions. Do not alter.
 */
void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    int n1=0, n2=0;

    try {
        countCharacters("", n1, n2);
        btassert<bool>((n1 == 0) && (n2 == 0));
        cout << "Passed TEST 1: countCharacters(empty string)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 #\n";
    }

    try {
        countCharacters("hello", n1, n2);
        btassert<bool>((n1 == 5) && (n2 == 0));
        cout << "Passed TEST 2: countCharacters(\"hello\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 #\n";
```

```
    }

    try {
        countCharacters("12345", n1, n2);
        btassert<bool>((n1 == 0) && (n2 == 5));
        cout << "Passed TEST 3: countCharacters(\"12345\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 #\n";
    }

    try {
        countCharacters("hello 12345", n1, n2);
        btassert<bool>((n1 == 5) && (n2 == 5));
        cout << "Passed TEST 4: countCharacters(\"hello 12345\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 #\n";
    }

    try {
        countCharacters("&,.", n1, n2);
        btassert<bool>((n1 == 0) && (n2 == 0));
        cout << "Passed TEST 5: countCharacters(\"&,.\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 #\n";
    }

    string s;

    try {
        s = upAndDown("hello");
        btassert<bool>(s == "HeLlO");
        cout << "Passed TEST 6: upAndDown(\"hello\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 #\n";
    }

    try {
        s = upAndDown("HeLlO");
        btassert<bool>(s == "HeLlO");
        cout << "Passed TEST 7: upAndDown(\"HeLlO\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 #\n";
    }

    try {
        s = upAndDown("hElLo");
        btassert<bool>(s == "HeLlO");
        cout << "Passed TEST 8: upAndDown(\"hElLo\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 #\n";
```

```
    }

    try {
        s = upAndDown("");
        btassert<bool>(s == "");
        cout << "Passed TEST 9: upAndDown(empty string)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 #\n";
    }

    try {
        s = upAndDown("a");
        btassert<bool>(s == "A");
        cout << "Passed TEST 10: upAndDown(\"a\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 #\n";
    }

    try {
        btassert<bool>(countWords("") == 0);
        cout << "Passed TEST 11: countWords(empty string)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 11 #\n";
    }

    try {
        btassert<bool>(countWords("hello") == 1);
        cout << "Passed TEST 12: countWords(\"hello\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 12 #\n";
    }

    try {
        btassert<bool>(countWords("hello,world") == 1);
        cout << "Passed TEST 13: countWords(\"hello world\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 13 #\n";
    }

    try {
        btassert<bool>(countWords("hello world") == 2);
        cout << "Passed TEST 14: countWords(\"hello world\")\n";
    } catch (bool b) {
        cout << "# FAILED TEST 14 #\n";
    }

    try {
        btassert<bool>(countWords("hello, world") == 2);
        cout << "Passed TEST 15: countWords(\"hello, world\")\n";
    } catch (bool b) {
```

```
        cout << "# FAILED TEST 15 #\n";
    }

    int values [] = {10, 20, 30};
    try {
        btassert<bool>(computeAverage(values, 3) == 20);
        cout << "Passed TEST 16: computeAverage([10,20,30])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 16 #\n";
    }

    values[0] = 0, values[1] = 0, values[2] = 0;
    try {
        btassert<bool>(computeAverage(values, 3) == 0);
        cout << "Passed TEST 17: computeAverage([0,0,0])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 17 #\n";
    }

    values[0] = 5, values[1] = 7, values[2] = 11;
    try {
        btassert<bool>(computeAverage(values, 3) == 7);
        cout << "Passed TEST 18: computeAverages([5,7,11])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 18 #\n";
    }

    values[0] = -10, values[1] = -20, values[2] = -30;
    try {
        btassert<bool>(computeAverage(values, 3) == -20);
        cout << "Passed TEST 19: computeAverages([-10,-20,-30])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 19 #\n";
    }

    values[0] = -5, values[1] = 0, values[2] = 5;
    try {
        btassert<bool>(computeAverage(values, 3) == 0);
        cout << "Passed TEST 20: computeAverages([-5,0,5])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 20 #\n";
    }

    values[0] = -1, values[1] = 0, values[2] = 1;
    try {
        btassert<bool>(findMinValue(values, 3) == -1);
        cout << "Passed TEST 21: findMinValue([-1,0,1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 21 #\n";
    }
```

```
    values[0] = -3, values[1] = -2, values[2] = -1;
    try {
        btassert<bool>(findMinValue(values, 3) == -3);
        cout << "Passed TEST 22: findMinValue([-3,-2,-1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 22 #\n";
    }

    values[0] = 0, values[1] = 1, values[2] = 2;
    try {
        btassert<bool>(findMinValue(values, 3) == 0);
        cout << "Passed TEST 23: findMinValue([0,1,2])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 23 #\n";
    }

    values[0] = 1, values[1] = 1, values[2] = 1;
    try {
        btassert<bool>(findMinValue(values, 3) == 1);
        cout << "Passed TEST 24: findMinValue([1,1,1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 24 #\n";
    }

    values[0] = INT_MAX, values[1] = INT_MAX, values[2] = INT_MAX;
    try {
        btassert<bool>(findMinValue(values, 3) == INT_MAX);
        cout << "Passed TEST 25: findMinValue([INT_MAX,INT_MAX,INT_MAX])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 25 #\n";
    }

    values[0] = -1, values[1] = 0, values[2] = 1;
    try {
        btassert<bool>(findMaxValue(values, 3) == 1);
        cout << "Passed TEST 26: findMaxValue([-1,0,1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 26 #\n";
    }

    values[0] = -3, values[1] = -2, values[2] = -1;
    try {
        btassert<bool>(findMaxValue(values, 3) == -1);
        cout << "Passed TEST 27: findMaxValue([-3,-2,-1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 27 #\n";
    }

    values[0] = 0, values[1] = 1, values[2] = 2;
```

```
    try {
        btassert<bool>(findMaxValue(values, 3) == 2);
        cout << "Passed TEST 28: findMaxValue([0,1,2])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 28 #\n";
    }

    values[0] = 1, values[1] = 1, values[2] = 1;
    try {
        btassert<bool>(findMaxValue(values, 3) == 1);
        cout << "Passed TEST 29: findMaxValue([1,1,1])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 29 #\n";
    }

    values[0] = INT_MIN, values[1] = INT_MIN, values[2] = INT_MIN;
    try {
        btassert<bool>(findMaxValue(values, 3) == INT_MIN);
        cout << "Passed TEST 30: findMaxValue([INT_MIN,INT_MIN,INT_MIN])\n";
    } catch (bool b) {
        cout << "# FAILED TEST 30 #\n";
    }

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

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