

```

1  /*
2  * Programming Project 1
3  * Programmer:  Chad Philip Johnson
4  * Submission Date:  February 15th, 2013
5  * Course:  CSCI21 - Programs & Algorithms II
6  */
7
8  #include <cassert>
9  #include <climits>
10 #include <iostream>
11 #include <string>
12 using namespace std;
13
14 /*
15 * Evaluates the number of alphabetical characters and numerical
16 * characters within a string.
17 * @param theString String object to be evaluated; passed by value.
18 * @param alpha Integer value to contain the counted number of
19 *       alphabetical characters within the string; passed by
20 *       reference.
21 * @param num Integer value to contain the counted number of
22 *       numerical characters within the string; passed by reference.
23 */
24 void countCharacters( string theString, int& alpha, int& num );
25
26 /*
27 * Changes the characters of a string by alternating between uppercase
28 *       and lowercase.
29 * @param theString String object to be evaluated; passed by value.
30 * @return The modified string object.
31 */
32 string upAndDown( string theString );
33
34 /*
35 * Counts the number of words contained within a string.  Words are
36 *       distinguished within the string by the appearance of a space
37 *       character:  ' '.
38 * @param theString String object to be evaluated; passed by value.
39 * @return Integer value representing the number of words contained
40 *       within the string object.
41 */
42 int countWords( string theString );
43
44 /*
45 * Computes the average value (rounded down) of the set of integer
46 *       values contained within an array.
47 * @param values [] Array containing a set of integer values.
48 * @param arraySize Integer value representing the size of the array;
49 *       passed by value.
50 * @return Integer value representing the rounded down average value

```

```

51     *         of the set of numbers contained within the array.
52     */
53     int computeAverage( int values [], int arraySize );
54
55     /*
56     * Finds the smallest value within an array containing a set of integer
57     * values.
58     * @param values [] Array containing a set of integer values.
59     * @param arraySize Integer value representing the size of the array;
60     * passed by value.
61     * @return Integer value representing the smallest value found within
62     * the array.
63     */
64     int findMinValue( int values [], int arraySize );
65
66     /*
67     * Finds the greatest value within an array containing a set of integer
68     * values.
69     * @param values [] Array containing a set of integer values.
70     * @param arraySize Integer value representing the size of the array;
71     * passed by value.
72     * @return Integer value representing the greatest value found within
73     * the array.
74     */
75     int findMaxValue( int values [], int arraySize );
76
77     /* for unit testing -- do not alter */
78     template <typename X, typename A>
79     void btassert(A assertion);
80     void unittest ();
81
82     int main (int argc, char* argv[])
83     {
84         unittest();
85
86         return 0;
87     }
88
89     void countCharacters( string theString, int& alpha, int& num )
90     {
91         int intAlphaCount = 0, intNumCount = 0;
92
93         for( int count = 0; count < theString.length(); count++ )
94         {
95             if( isalpha( theString[count] ) )
96                 intAlphaCount++;
97             else if( isdigit( theString[count] ) )
98                 intNumCount++;
99         }
100

```

```

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 }

```

```

151
152 int findMinValue( int values [], int arraySize )
153 {
154     int intMinimumValue = values[0];
155
156     for( int count = 0; count < (arraySize - 1); count++ )
157     {
158         if( values[(count + 1)] < intMinimumValue )
159             intMinimumValue = values[(count + 1)];
160     }
161
162     return intMinimumValue;
163 }
164
165 int findMaxValue( int values [], int arraySize )
166 {
167     int intMaximumValue = values[0];
168
169     for( int count = 0; count < (arraySize - 1); count++ )
170     {
171         if( values[(count + 1)] > intMaximumValue )
172             intMaximumValue = values[(count + 1)];
173     }
174
175     return intMaximumValue;
176 }
177
178 /*
179  * Unit testing functions. Do not alter.
180  */
181 void unittest ()
182 {
183     cout << "\nSTARTING UNIT TEST\n\n";
184
185     int n1=0, n2=0;
186
187     try {
188         countCharacters("", n1, n2);
189         btassert<bool>((n1 == 0) && (n2 == 0));
190         cout << "Passed TEST 1: countCharacters(empty string)\n";
191     } catch (bool b) {
192         cout << "# FAILED TEST 1 #\n";
193     }
194
195     try {
196         countCharacters("hello", n1, n2);
197         btassert<bool>((n1 == 5) && (n2 == 0));
198         cout << "Passed TEST 2: countCharacters(\"hello\")\n";
199     } catch (bool b) {
200         cout << "# FAILED TEST 2 #\n";

```

```

201     }
202
203     try {
204         countCharacters("12345", n1, n2);
205         btassert<bool>((n1 == 0) && (n2 == 5));
206         cout << "Passed TEST 3: countCharacters(\"12345\")\n";
207     } catch (bool b) {
208         cout << "# FAILED TEST 3 #\n";
209     }
210
211     try {
212         countCharacters("hello 12345", n1, n2);
213         btassert<bool>((n1 == 5) && (n2 == 5));
214         cout << "Passed TEST 4: countCharacters(\"hello 12345\")\n";
215     } catch (bool b) {
216         cout << "# FAILED TEST 4 #\n";
217     }
218
219     try {
220         countCharacters("&.", n1, n2);
221         btassert<bool>((n1 == 0) && (n2 == 0));
222         cout << "Passed TEST 5: countCharacters(\"&.\")\n";
223     } catch (bool b) {
224         cout << "# FAILED TEST 5 #\n";
225     }
226
227     string s;
228
229     try {
230         s = upAndDown("hello");
231         btassert<bool>(s == "HeLl0");
232         cout << "Passed TEST 6: upAndDown(\"hello\")\n";
233     } catch (bool b) {
234         cout << "# FAILED TEST 6 #\n";
235     }
236
237     try {
238         s = upAndDown("HeLl0");
239         btassert<bool>(s == "HeLl0");
240         cout << "Passed TEST 7: upAndDown(\"HeLl0\")\n";
241     } catch (bool b) {
242         cout << "# FAILED TEST 7 #\n";
243     }
244
245     try {
246         s = upAndDown("hElLo");
247         btassert<bool>(s == "HeLl0");
248         cout << "Passed TEST 8: upAndDown(\"hElLo\")\n";
249     } catch (bool b) {
250         cout << "# FAILED TEST 8 #\n";

```

```
251     }
252
253     try {
254         s = upAndDown("");
255         btassert<bool>(s == "");
256         cout << "Passed TEST 9: upAndDown(empty string)\n";
257     } catch (bool b) {
258         cout << "# FAILED TEST 9 #\n";
259     }
260
261     try {
262         s = upAndDown("a");
263         btassert<bool>(s == "A");
264         cout << "Passed TEST 10: upAndDown(\"a\")\n";
265     } catch (bool b) {
266         cout << "# FAILED TEST 10 #\n";
267     }
268
269     try {
270         btassert<bool>(countWords("") == 0);
271         cout << "Passed TEST 11: countWords(empty string)\n";
272     } catch (bool b) {
273         cout << "# FAILED TEST 11 #\n";
274     }
275
276     try {
277         btassert<bool>(countWords("hello") == 1);
278         cout << "Passed TEST 12: countWords(\"hello\")\n";
279     } catch (bool b) {
280         cout << "# FAILED TEST 12 #\n";
281     }
282
283     try {
284         btassert<bool>(countWords("hello,world") == 1);
285         cout << "Passed TEST 13: countWords(\"hello world\")\n";
286     } catch (bool b) {
287         cout << "# FAILED TEST 13 #\n";
288     }
289
290     try {
291         btassert<bool>(countWords("hello world") == 2);
292         cout << "Passed TEST 14: countWords(\"hello world\")\n";
293     } catch (bool b) {
294         cout << "# FAILED TEST 14 #\n";
295     }
296
297     try {
298         btassert<bool>(countWords("hello, world") == 2);
299         cout << "Passed TEST 15: countWords(\"hello, world\")\n";
300     } catch (bool b) {
```

```

301     cout << "# FAILED TEST 15 #\n";
302 }
303
304 int values [] = {10, 20, 30};
305 try {
306     btassert<bool>(computeAverage(values, 3) == 20);
307     cout << "Passed TEST 16: computeAverage([10,20,30])\n";
308 } catch (bool b) {
309     cout << "# FAILED TEST 16 #\n";
310 }
311
312 values[0] = 0, values[1] = 0, values[2] = 0;
313 try {
314     btassert<bool>(computeAverage(values, 3) == 0);
315     cout << "Passed TEST 17: computeAverage([0,0,0])\n";
316 } catch (bool b) {
317     cout << "# FAILED TEST 17 #\n";
318 }
319
320 values[0] = 5, values[1] = 7, values[2] = 11;
321 try {
322     btassert<bool>(computeAverage(values, 3) == 7);
323     cout << "Passed TEST 18: computeAverages([5,7,11])\n";
324 } catch (bool b) {
325     cout << "# FAILED TEST 18 #\n";
326 }
327
328 values[0] = -10, values[1] = -20, values[2] = -30;
329 try {
330     btassert<bool>(computeAverage(values, 3) == -20);
331     cout << "Passed TEST 19: computeAverages([-10,-20,-30])\n";
332 } catch (bool b) {
333     cout << "# FAILED TEST 19 #\n";
334 }
335
336 values[0] = -5, values[1] = 0, values[2] = 5;
337 try {
338     btassert<bool>(computeAverage(values, 3) == 0);
339     cout << "Passed TEST 20: computeAverages([-5,0,5])\n";
340 } catch (bool b) {
341     cout << "# FAILED TEST 20 #\n";
342 }
343
344 values[0] = -1, values[1] = 0, values[2] = 1;
345 try {
346     btassert<bool>(findMinValue(values, 3) == -1);
347     cout << "Passed TEST 21: findMinValue([-1,0,1])\n";
348 } catch (bool b) {
349     cout << "# FAILED TEST 21 #\n";
350 }

```

```

351
352 values[0] = -3, values[1] = -2, values[2] = -1;
353 try {
354     btassert<bool>(findMinValue(values, 3) == -3);
355     cout << "Passed TEST 22: findMinValue([-3,-2,-1])\n";
356 } catch (bool b) {
357     cout << "# FAILED TEST 22 #\n";
358 }
359
360 values[0] = 0, values[1] = 1, values[2] = 2;
361 try {
362     btassert<bool>(findMinValue(values, 3) == 0);
363     cout << "Passed TEST 23: findMinValue([0,1,2])\n";
364 } catch (bool b) {
365     cout << "# FAILED TEST 23 #\n";
366 }
367
368 values[0] = 1, values[1] = 1, values[2] = 1;
369 try {
370     btassert<bool>(findMinValue(values, 3) == 1);
371     cout << "Passed TEST 24: findMinValue([1,1,1])\n";
372 } catch (bool b) {
373     cout << "# FAILED TEST 24 #\n";
374 }
375
376 values[0] = INT_MAX, values[1] = INT_MAX, values[2] = INT_MAX;
377 try {
378     btassert<bool>(findMinValue(values, 3) == INT_MAX);
379     cout << "Passed TEST 25: findMinValue([INT_MAX,INT_MAX,INT_MAX])\n";
380 } catch (bool b) {
381     cout << "# FAILED TEST 25 #\n";
382 }
383
384 values[0] = -1, values[1] = 0, values[2] = 1;
385 try {
386     btassert<bool>(findMaxValue(values, 3) == 1);
387     cout << "Passed TEST 26: findMaxValue([-1,0,1])\n";
388 } catch (bool b) {
389     cout << "# FAILED TEST 26 #\n";
390 }
391
392 values[0] = -3, values[1] = -2, values[2] = -1;
393 try {
394     btassert<bool>(findMaxValue(values, 3) == -1);
395     cout << "Passed TEST 27: findMaxValue([-3,-2,-1])\n";
396 } catch (bool b) {
397     cout << "# FAILED TEST 27 #\n";
398 }
399
400 values[0] = 0, values[1] = 1, values[2] = 2;

```



```

401     try {
402         btassert<bool>(findMaxValue(values, 3) == 2);
403         cout << "Passed TEST 28: findMaxValue([0,1,2])\n";
404     } catch (bool b) {
405         cout << "# FAILED TEST 28 #\n";
406     }
407
408     values[0] = 1, values[1] = 1, values[2] = 1;
409     try {
410         btassert<bool>(findMaxValue(values, 3) == 1);
411         cout << "Passed TEST 29: findMaxValue([1,1,1])\n";
412     } catch (bool b) {
413         cout << "# FAILED TEST 29 #\n";
414     }
415
416     values[0] = INT_MIN, values[1] = INT_MIN, values[2] = INT_MIN;
417     try {
418         btassert<bool>(findMaxValue(values, 3) == INT_MIN);
419         cout << "Passed TEST 30: findMaxValue([INT_MIN,INT_MIN,INT_MIN])\n";
420     } catch (bool b) {
421         cout << "# FAILED TEST 30 #\n";
422     }
423
424     cout << "\nUNIT TEST COMPLETE\n\n";
425 }
426
427 template <typename X, typename A>
428 void btassert (A assertion)
429 {
430     if (!assertion)
431         throw X();
432 }
433

```