

```

1  /*
2   * Programming Challenge 14
3   */
4  #include <cassert>
5  #include <iostream>
6  #include <string>
7  using namespace std;
8
9  /*
10   * Allocate memory for a dynamic string with specified contents.
11   * @param contents the desired contents of the dynamic string
12   * @return a pointer to the newly allocated string
13   */
14  string* makeDynoString (string contents);
15
16  /*
17   * Free the memory associated with a dynamic string and NULL out its pointer.
18   * @param theString a pointer (passed by reference) to a dynamic string
19   */
20  void clearDynoString (string*& theString);
21
22  /*
23   * Count the number of alphabetic and numeric characters in a string and return its length.
24   * @param theString a pointer to the string in which characters will be counted
25   * @param alpha an unsigned int (passed by reference) that will contain the count
26   *             of alphabetic characters in the string on return
27   * @param num an unsigned int (passed by reference) that will contain the count
28   *            of numeric characters in the string on return
29   * @return an unsigned integer contained the length of theString
30   * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
31   */
32  unsigned int countChars (string* theString, unsigned int& alpha, unsigned int& num);
33
34  /*
35   * Find a word inside of a string.
36   * @param theString the string in which the search for a word will take place
37   * @param theWord the word to look for inside of theString
38   * @return true if theWord is found in theString, else false
39   * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
40   */
41  bool findWord (string* theString, string theWord);
42
43  /*
44   * Replace one word in a string with another word.
45   * @param theString a pointer to the string that will have a word replaced
46   * @param oldWord the word inside of theString being replaced
47   * @param newWord the word that will be used to replace oldWord in theString
48   * @return true if oldWord was found and replaced, else return false
49   * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
50   */

```

```

51  bool replaceWord (string* theString, string oldWord, string newWord);
52
53  /* for unit testing -- do not alter */
54  struct ArrayException
55  {
56      ArrayException (string newMessage="error")
57          : message(newMessage)
58      {
59      }
60
61      string message;
62  };
63
64  template <typename X, typename A>
65  void btassert(A assertion);
66  void unittest ();
67
68  int main (int argc, char* argv[])
69  {
70      unittest();
71
72      return 0;
73  }
74
75  // CODE HERE -- FUNCTION DEFINITIONS
76
77  /*
78   * Allocate memory for a dynamic string with specified contents.
79   * @param contents the desired contents of the dynamic string
80   * @return a pointer to the newly allocated string
81   */
82  string* makeDynoString (string contents)
83  {
84      return new string( contents );
85  }
86
87  /*
88   * Free the memory associated with a dynamic string and NULL out its pointer.
89   * @param theString a pointer (passed by reference) to a dynamic string
90   */
91  void clearDynoString (string*& theString)
92  {
93      delete theString;
94      theString = NULL;
95  }
96
97  /*
98   * Count the number of alphabetic and numeric characters in a string and return its length.
99   * @param theString a pointer to the string in which characters will be counted
100  * @param alpha an unsigned int (passed by reference) that will contain the count

```

```

101  *      of alphabetic characters in the string on return
102  * @param num an unsigned int (passed by reference) that will contain the count
103  *      of numeric characters in the string on return
104  * @return an unsigned integer contained the length of theString
105  * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
106  */
107  unsigned int countChars (string* theString, unsigned int& alpha, unsigned int& num)
108  {
109      if( theString == NULL )
110          throw ArrayException( "NULL STRING REFERENCE" );
111
112      int intAlphaTotal = 0, intNumTotal = 0;
113
114      for( int count = 0; count < (*theString).length(); count++ )
115      {
116          if( isalpha( (*theString).at(count) ) )
117              intAlphaTotal++;
118          else if( isdigit( (*theString).at(count) ) )
119              intNumTotal++;
120      }
121
122      alpha = intAlphaTotal;
123      num = intNumTotal;
124
125      return (*theString).length();
126  }
127
128  /*
129  * Find a word inside of a string.
130  * @param theString the string in which the search for a word will take place
131  * @param theWord the word to look for inside of theString
132  * @return true if theWord is found in theString, else false
133  * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
134  */
135  bool findWord (string* theString, string theWord)
136  {
137      if( theString == NULL )
138          throw ArrayException( "NULL STRING REFERENCE" );
139
140      for( int count01 = 0; count01 < (*theString).length(); count01++ )
141      {
142          if( (*theString).at( count01 ) == theWord.at( 0 ) )
143          {
144              int count02 = 1;
145              while( (count01 + count02) < (*theString).length() && count02 < theWord.length() && (*theString).at(count01 + count02)
146                  == theWord[count02] )
147              {
148                  count02++;
149              }

```

```

150         if( count02 == theWord.length() )
151             return true;
152     }
153 }
154
155 return false;
156 }
157
158 /*
159  * Replace one word in a string with another word.
160  * @param theString a pointer to the string that will have a word replaced
161  * @param oldWord the word inside of theString being replaced
162  * @param newWord the word that will be used to replace oldWord in theString
163  * @return true if oldWord was found and replaced, else return false
164  * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
165  */
166 bool replaceWord (string* theString, string oldWord, string newWord)
167 {
168     if( theString == NULL )
169         throw ArrayException( "NULL STRING REFERENCE" );
170
171     string* strTemp = NULL;
172     bool boolReplacementOperation = false;
173
174     for( int count01 = 0; count01 < (*theString).length(); count01++ )
175     {
176         if( (*theString).at( count01 ) == oldWord.at( 0 ) )
177         {
178             int count02 = 1;
179             while( (count01 + count02) < (*theString).length() && count02 < oldWord.length() && (*theString).at(count01 + count02)
180                 == oldWord[count02] )
181             {
182                 count02++;
183             }
184             //"123, abc; 456: hello. 0!"
185             if( count02 == oldWord.length() )
186             {
187                 strTemp = new string( (*theString).substr( 0, count01 ) + newWord + (*theString).substr( count01 + oldWord.length
188                 (), (*theString).length() ) );
189                 // Step #01: pointer myString from outside of the function assigns its memory address local pointer theString
190                 // Step #02: the local pointer theString can be assigned a different memory address that does not affect the
191                 // memory address assigned to pointer myString; all "new" strings are local in scope and are destroyed
192                 // once the function ends
193                 // Step #03: any "new" string that needs to persist outside of the function has to be assigned to a memory
194                 // address that exists outside of the function: so long as theString does not have its memory address
195                 // changed the value at that location can be refreshed by assigning a value with *theString
196                 // Step #04: pointer strTemp is assigned the memory address holding the "new" string; this memory address
197                 // is local to the function. by using *theString = *strTemp the value at one memory address that is local
198                 // to the function is passed to a different memory address that is local to either main() or the calling
199                 // function (in this case unitTest()).

```

```

198         //delete theString;
199         // do not set point value to NULL because it breaks the connection with the pointer originally passed to the
200         //      function; this is why clearDynoString should not be used here because it will sever the connection
201         //      between the two pointers
202         *theString = *strTemp;
203         delete strTemp;
204         strTemp = NULL;
205
206         boolReplacementOperation = true;
207         count01 += count02;
208     }
209 }
210 }
211
212 if( boolReplacementOperation )
213     return true;
214 else
215     return false;
216 }
217
218 /*
219  * Unit testing functions. Do not alter.
220  */
221
222 void unittest ()
223 {
224     cout << "\nSTARTING UNIT TEST\n\n";
225
226     string* myString = 0;
227     unsigned int alpha, num;
228
229     try {
230         countChars(myString, alpha, num);
231     } catch (ArrayException e) {
232         try {
233             btassert<bool>(e.message == "NULL STRING REFERENCE");
234             cout << "Passed TEST 1: countChars EXCEPTION HANDLING (STRING*) () \n";
235         } catch (bool b) {
236             cout << "# FAILED TEST 1: countChars EXCEPTION HANDLING (STRING*) () #\n";
237         }
238     }
239
240     try {
241         findWord(myString, "hello");
242     } catch (ArrayException e) {
243         try {
244             btassert<bool>(e.message == "NULL STRING REFERENCE");
245             cout << "Passed TEST 2: findWord EXCEPTION HANDLING (STRING*) () \n";
246         } catch (bool b) {
247             cout << "# FAILED TEST 2: findWord EXCEPTION HANDLING (STRING*) () #\n";

```

```

248     }
249 }
250
251 try {
252     replaceWord(myString, "hello", "goodbye");
253 } catch (ArrayException e) {
254     try {
255         btassert<bool>(e.message == "NULL STRING REFERENCE");
256         cout << "Passed TEST 3: replaceWord EXCEPTION HANDLING (STRING*) () \n";
257     } catch (bool b) {
258         cout << "# FAILED TEST 3: replaceWord EXCEPTION HANDLING (STRING*) () #\n";
259     }
260 }
261
262 myString = makeDynoString("123, abc; 456: hello. 0!");
263
264 try {
265     btassert<bool>(myString != 0);
266     cout << "Passed TEST 4: STRING INITIALIZATION () \n";
267 } catch (bool b) {
268     cout << "# FAILED TEST 4: STRING INITIALIZATION () #\n";
269 }
270
271 try {
272     btassert<bool>(countChars(myString, alpha, num) == 24);
273     cout << "Passed TEST 5: countChars (myString) \n";
274 } catch (bool b) {
275     cout << "# FAILED TEST 5: countChars (myString) #\n";
276 }
277
278 try {
279     btassert<bool>(alpha == 8);
280     cout << "Passed TEST 6: countChars (alphabetic) \n";
281 } catch (bool b) {
282     cout << "# FAILED TEST 6: countChars (alphabetic) #\n";
283 }
284
285 try {
286     btassert<bool>(num == 7);
287     cout << "Passed TEST 7: countChars (numeric) \n";
288 } catch (bool b) {
289     cout << "# FAILED TEST 7: countChars (numeric) #\n";
290 }
291
292 try {
293     btassert<bool>(findWord(myString, "HELLO") == false);
294     cout << "Passed TEST 8: findWord (\\"HELLO\\") \n";
295 } catch (bool b) {
296     cout << "# FAILED TEST 8: findWord (\\"HELLO\\") #\n";
297 }

```

```

298
299     try {
300         btassert<bool>(findWord(myString, "abc") == true);
301         cout << "Passed TEST 9: findWord (\"abc\") \n";
302     } catch (bool b) {
303         cout << "# FAILED TEST 9: findWord (\"abc\") #\n";
304     }
305
306     replaceWord(myString, "hello", "goodbye");
307     try {
308         btassert<bool>(*myString == "123, abc; 456: goodbye. 0!");
309         cout << "Passed TEST 10: replaceWord (myString, \"hello\", \"goodbye\") \n";
310     } catch (bool b) {
311         cout << "# FAILED TEST 10: replaceWord (myString, \"hello\", \"goodbye\") #\n";
312     }
313
314     try {
315         btassert<bool>(replaceWord(myString, "HELLO", "GOODBYE") == false);
316         cout << "Passed TEST 11: replaceWord (myString, \"HELLO\", \"GOODBYE\") \n";
317     } catch (bool b) {
318         cout << "# FAILED TEST 11: replaceWord (myString, \"HELLO\", \"GOODBYE\") #\n";
319     }
320
321     clearDynoString(myString);
322
323     try {
324         btassert<bool>(myString == 0);
325         cout << "Passed TEST 12: clearDynoString () \n";
326     } catch (bool b) {
327         cout << "# FAILED TEST 12: clearDynoString () #\n";
328     }
329
330     cout << "\nUNIT TEST COMPLETE\n\n";
331 }
332
333 template <typename X, typename A>
334 void btassert (A assertion)
335 {
336     if (!assertion)
337         throw X();
338 }

```