```
/*
 * Programming Challenge 14
 */
#include <cassert>
#include <iostream>
#include <string>
using namespace std;

/*
 * Allocate memory for a dynamic string with specified contents.
 * @param contents the desired contents of the dynamic string
 * @return a pointer to the newly allocated string
 */
string* makeDynoString (string contents);

/*
 * Free the memory associated with a dynamic string and NULL out its pointer.
 * @param theString a pointer (passed by reference) to a dynamic string
 */
void clearDynoString (string*& theString);

/*
 * Count the number of alphabetic and numeric characters in a string and return its length.
 * @param theString a pointer to the string in which characters will be counted
 * @param alpha an unsigned int (passed by reference) that will contain the count
 *        of alphabetic characters in the string on return
 * @param num an unsigned int (passed by reference) that will contain the count
 *        of numeric characters in the string on return
 * @return an unsigned integer containined the length of theString
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
unsigned int countChars (string* theString, unsigned int& alpha, unsigned int& num);

/*
 * Find a word inside of a string.
 * @param theString the string in which the search for a word will take place
 * @param theWord the word to look for inside of theString
 * @return true if theWord is found in theString, else false
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
bool findWord (string* theString, string theWord);

/*
 * Replace one word in a string with another word.
 * @param theString a pointer to the string that will have a word replaced
 * @param oldWord the word inside of theString being replaced
 * @param newWord the word that will be used to replace oldWord in theString
 * @return true if oldWord was found and replaced, else return false
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
```

```
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
```

```
 *        of alphabetic characters in the string on return
 * @param num an unsigned int (passed by reference) that will contain the count
 *        of numeric characters in the string on return
 * @return an unsigned integer containined the length of theString
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
unsigned int countChars (string* theString, unsigned int& alpha, unsigned int& num)
{
    if( theString == NULL )
        throw ArrayException( "NULL STRING REFERENCE" );

    int intAlphaTotal = 0, intNumTotal = 0;

    for( int count = 0; count < (*theString).length(); count++ )
    {
        if( isalpha( (*theString).at(count) ) )
            intAlphaTotal++;
        else if( isdigit( (*theString).at(count) ) )
            intNumTotal++;
    }

    alpha = intAlphaTotal;
    num = intNumTotal;

    return (*theString).length();
}

/*
 * Find a word inside of a string.
 * @param theString the string in which the search for a word will take place
 * @param theWord the word to look for inside of theString
 * @return true if theWord is found in theString, else false
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
bool findWord (string* theString, string theWord)
{
    if( theString == NULL )
        throw ArrayException( "NULL STRING REFERENCE" );

    for( int count01 = 0; count01 < (*theString).length(); count01++ )
    {
        if( (*theString).at( count01 ) == theWord.at( 0 ) )
        {
            int count02 = 1;
            while( (count01 + count02) < (*theString).length() && count02 < theWord.length() && (*theString).at(count01 + count02
            ) == theWord[count02] )
            {
                count02++;
            }

```

```
            if( count02 == theWord.length() )
                return true;
        }
    }

    return false;
}

/*
 * Replace one word in a string with another word.
 * @param theString a pointer to the string that will have a word replaced
 * @param oldWord the word inside of theString being replaced
 * @param newWord the word that will be used to replace oldWord in theString
 * @return true if oldWord was found and replaced, else return false
 * @throw ArrayException with the message "NULL STRING REFERENCE" if theString is NULL
 */
bool replaceWord (string* theString, string oldWord, string newWord)
{
    if( theString == NULL )
        throw ArrayException( "NULL STRING REFERENCE" );

    string* strTemp = NULL;
    bool boolReplacementOperation = false;

    for( int count01 = 0; count01 < (*theString).length(); count01++ )
    {
        if( (*theString).at( count01 ) == oldWord.at( 0 ) )
        {
            int count02 = 1;
            while( (count01 + count02) < (*theString).length() && count02 < oldWord.length() && (*theString).at(count01 + count02
            ) == oldWord[count02] )
            {
                count02++;
            }
            //"123, abc; 456: hello. 0!"
            if( count02 == oldWord.length() )
            {
                strTemp = new string( (*theString).substr( 0, count01 ) + newWord + (*theString).substr( count01 + oldWord.length
                (), (*theString).length() ) );
                // Step #01:  pointer myString from outside of the function assigns its memory address local pointer theString
                // Step #02:  the local pointer theString can be assigned a different memory address that does not affect the
                //      memory address assigned to pointer myString; all "new" strings are local in scope and are destroyed
                //      once the function ends
                // Step #03:  any "new" string that needs to persist outside of the function has to be assigned to a memory
                //      address that exists outside of the function:  so long as theString does not have its memory address
                //      changed the value at that location can be refreshed by assigning a value with *theString
                // Step #04:  pointer strTemp is assigned the memory address holding the "new" string; this memory address
                //      is local to the function.  by using *theString = *strTemp the value at one memory address that is local
                //      to the function is passed to a different memory address that is local to either main() or the calling
                //      function (in this case unitTest()).
```

```
                //delete theString;
                // do not set point value to NULL because it breaks the connection with the pointer originally passed to the
                //      function; this is why clearDynoString should not be used here because it will sever the connection
                //      between the two pointers
                *theString = *strTemp;
                delete strTemp;
                strTemp = NULL;

                boolReplacementOperation = true;
                count01 += count02;
            }
        }
    }

    if( boolReplacementOperation )
        return true;
    else
        return false;
}

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

void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    string* myString = 0;
    unsigned int alpha, num;

    try {
        countChars(myString, alpha, num);
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "NULL STRING REFERENCE");
            cout << "Passed TEST 1: countChars EXCEPTION HANDLING (STRING*) () \n";
        } catch (bool b) {
            cout << "# FAILED TEST 1: countChars EXCEPTION HANDLING (STRING*) () #\n";
        }
    }

    try {
        findWord(myString, "hello");
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "NULL STRING REFERENCE");
            cout << "Passed TEST 2: findWord EXCEPTION HANDLING (STRING*) () \n";
        } catch (bool b) {
            cout << "# FAILED TEST 2: findWord EXCEPTION HANDLING (STRING*) () #\n";
```

```
        }
    }

    try {
        replaceWord(myString, "hello", "goodbye");
    } catch (ArrayException e) {
        try {
            btassert<bool>(e.message == "NULL STRING REFERENCE");
            cout << "Passed TEST 3: replaceWord EXCEPTION HANDLING (STRING*) () \n";
        } catch (bool b) {
            cout << "# FAILED TEST 3: replaceWord EXCEPTION HANDLING (STRING*) () #\n";
        }
    }

    myString = makeDynoString("123, abc; 456: hello. 0!");

    try {
        btassert<bool>(myString != 0);
        cout << "Passed TEST 4: STRING INITIALIZATION () \n";
    } catch (bool b) {
        cout << "# FAILED TEST 4: STRING INITIALIZATION () #\n";
    }

    try {
        btassert<bool>(countChars(myString, alpha, num) == 24);
        cout << "Passed TEST 5: countChars (myString) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 5: countChars (myString) #\n";
    }

    try {
        btassert<bool>(alpha == 8);
        cout << "Passed TEST 6: countChars (alphabetic) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 6: countChars (alphabetic) #\n";
    }

    try {
        btassert<bool>(num == 7);
        cout << "Passed TEST 7: countChars (numeric) \n";
    } catch (bool b) {
        cout << "# FAILED TEST 7: countChars (numeric) #\n";
    }

    try {
        btassert<bool>(findWord(myString, "HELLO") == false);
        cout << "Passed TEST 8: findWord (\"HELLO\") \n";
    } catch (bool b) {
        cout << "# FAILED TEST 8: findWord (\"HELLO\") #\n";
    }
```

```
    try {
        btassert<bool>(findWord(myString, "abc") == true);
        cout << "Passed TEST 9: findWord (\"abc\") \n";
    } catch (bool b) {
        cout << "# FAILED TEST 9: findWord (\"abc\") #\n";
    }

    replaceWord(myString, "hello", "goodbye");
    try {
        btassert<bool>(*myString == "123, abc; 456: goodbye. 0!");
        cout << "Passed TEST 10: replaceWord (myString, \"hello\", \"goodbye\") \n";
    } catch (bool b) {
        cout << "# FAILED TEST 10: replaceWord (myString, \"hello\", \"goodbye\") #\n";
    }

    try {
        btassert<bool>(replaceWord(myString, "HELLO", "GOODBYE") == false);
        cout << "Passed TEST 11: replaceWord (myString, \"HELLO\", \"GOODBYE\") \n";
    } catch (bool b) {
        cout << "# FAILED TEST 11: replaceWord (myString, \"HELLO\", \"GOODBYE\") #\n";
    }

    clearDynoString(myString);

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

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

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