Programming Assignment #6

This assignment will give you some practice with C++ file I/O as well as practice with std::strings and std::vectors. The main purpose of the program is work with C++ strings and implement some useful functionality. One of the tasks is to implement a spell-checker class. Besides checking words for correct spelling, you will also implement a few other functions. This is a partial interface for the class:

```
class SpellChecker
{
  public:
    enum SCResult { scrFILE_OK = -1,       /* File was opened successfully */
                    scrFILE_ERR_OPEN = -2, /* File was unable to be opened */
                    scrWORD_OK = 1,        /* Found in the dictionary      */
                    scrWORD_BAD = 2        /* Not found in the dictionary  */
                  };
    SpellChecker(const std::string &dictionary);
    // Other public methods (see website)
  private:
    std::string dictionary_;
};
```

Method	Description
`SpellChecker(const std::string &dictionary);`	Constructor. A dictionary (filename) must be supplied.
`SCResult WordsStartingWith(char letter, int& count) const;`	Count the number of words that start with `letter`. If the file can't be opened, return `scrFILE_ERR_OPEN`. If successful, return `scrFILE_OK`. The count is returned in the reference parameter, `count`.
`SCResult WordLengths(std::vector<int>& lengths, size_t count) const;`	Count the number of words that have length 1 to `count` and store them in the vector `lengths` at the appropriate index. If the file can't be opened, return `scrFILE_ERR_OPEN`, otherwise return `scrFILE_OK`.
`SCResult GetInfo(DictionaryInfo &info) const;`	Return some information about the dictionary using the reference parameter. If the file can't be opened, return `scrFILE_ERR_OPEN`, otherwise return `scrFILE_OK`.
`SCResult SpellCheck(const std::string& word) const;`	Lookup the word in the dictionary. If the word was found, return `scrWORD_OK`. If the word was not found, return `scrWORD_BAD`. If the dictionary file can't be opened, return `scrFILE_ERR_OPEN`.
`SCResult AcronymToWord(const std::string& acronym, std::vector<std::string> &words, size_t maxlen = 0) const;`	Given a string, find words in the dictionary that are composed of the letters in the same order. This will be explained in detail in class. If the dictionary can't be opened, return `scrFILE_ERR_OPEN`. If successful, return `scrFILE_OK`.

Other string Operations
In addition to the spell-checking functionality, there are 3 other methods that you need to implement. They are all static methods in a class called *StringUtils.* See the driver and header file for details.

Methods (static)	Description
`std::string& UpperCase(std::string& string);`	Simple function to convert a string to upper case. The conversion happens in-place, so no copy is made in the function. (**This is provided for you.**)
`std::vector<std::string> Tokenize(const std::string& words);`	Split a string into words. A word is any character but a space. Return the words in a vector.
`std::vector<std::string> WordWrap(const std::string& words, size_t line_length);`	Given a string, break it into lines less than or equal to line_length. The sample driver has numerous examples and tests for this.

Notes:

1. All of the methods in the *SpellChecker* class (except the constructor) need to open the dictionary. If it fails when attempting to open the file, you must simply return `scrFILE_ERR_OPEN`.
2. There is exactly one word per line in the dictionaries. All words end with a newline character which is not to be included in the word. Since you're using *string* and *ifstream,* this will be stripped off for you.
3. You are not to include any other header files than those that are already included. Including other files will cause you to receive a 0 on the assignment.
4. The methods `WordsStartingWith` and `Spellcheck` are case-insensitive when looking for characters or words. You need to account for this. One way to do this is to make everything uppercase or lowercase before comparing. The test driver shows examples of how your code should function.
5. You should work on the SpellChecker methods first (except for `AcronymToWord`), as they are all very simple. Then work on `Tokenize`, `AcronymToWord`, and `WordWrap`, in that order. Two of these functions are static methods in a class named *StringUtils.*
6. Do not call `Tokenize` from the `WordWrap` function. You are to implement the algorithm that I discussed in class.
7. Do not use **new** or **delete** anywhere. You don't need them. The STL takes care of all memory for you.
8. There are additional tips listed on the web page for this assignment. **PLEASE READ THEM.**
9. **PLEASE USE diff** or something like it**.** *Don't be a fool, use a diff tool.* ™

Details
Because you are using the std::string, std::vector and std::ifstream classes for much of the hard work, the amount of code for this assignment is much less than the equivalent C-style version. There are many tests in the driver to get you started. This assignment will only require you to include the three system header files mentioned. No other system header files should be included. Don't forget to put a comment next to each included file that indicates the reasons for including it. Make sure you can write the comments before you start to write the code. These comments (pseudo code) will guide you when writing C++ code.

Deliverables
You must submit the header file (FileStrings.h), the implementation file (FileStrings.cpp), and the Doxygen-generated index.chm file. These file must be zipped up and uploaded to appropriate submission place on Moodle.

Refer to the web page for this assignment for any additional details on how to submit this assignment. **Do not submit any other files than the ones listed.**

Source Files	Description
FileStrings.cpp	The implementation file. All implementation for the functions goes here. You must document the file (file header comments) and all functions (function header comments) using the appropriate Doxygen tags.
FileStrings.h	The header file. Except for adding a file header comment, you don't need to modify this file.

Command line
```
g++ -o gnu driver-sample.cpp FileStrings.cpp -g -O2 -Wall -Wextra -ansi -pedantic -
Wconversion -Wold-style-cast
```

Usual stuff
Your code must compile (using the compilers specified in the syllabus) to receive credit. You should do a “test run” by extracting the files into a folder and verifying that you can compile and execute what you have submitted (because that's what I'm going to do.) Details about how to submit are posted on the course website and in the syllabus.

Make sure your name and other info is on all documents.