**Spell checker**

**Refer to the documenting and submitting homework from** [**Programming Guidelines**](https://ucdenver.instructure.com/courses/347316/pages/programming-guidelines)**.**

This programming homework is to develop a simple spelling checker. The file [**dict.dat**](https://ucdenver.instructure.com/courses/347316/files/4667972/download?wrap=1)**[Preview the documentView in a new window](https://ucdenver.instructure.com/courses/347316/files/4667972/download?wrap=1)** contains 25,021 frequently used words, each on a separate line in lowercase. Read the file, and insert the words into a hash table with 1373 buckets.  Remember to move dict.dat to the csegrid.  Then run the command dos2unix dict.dat (to remove those pesky /r's created by Windows)

Prompt for the name of an input text file to check. This file will contain a number of words.

For this assignment a word is any sequence of one or more characters separated by one or more Spaces or newlines.  You could be reading text from a book, so you have to delete starting and ending quotations, and delete periods, question marks, exclamation marks and semi-colons from the back of the string (Conveniently strings have a .back member function, and the first character is at [0]). If you include cctype, you will be able to ask if isalnum.  Also remember the purpose of << operators.

Read the document, and separate it into a sequence of words converted to lowercase.  Use [http://www.cplusplus.com/reference/locale/tolower/ (Links to an external site.)Links to an external site.](http://www.cplusplus.com/reference/locale/tolower/) as an example way to convert. A for loop could be useful in converting all characters.

Print out the words that could be mispelled, then print out the # of Words in the Dictionary, # of Words in the File, # of words not in the dictionary.

Here are two files to check against (you may be a few off depending on how you coded):

[check.txt Preview the documentView in a new window](https://ucdenver.instructure.com/courses/347316/files/4667954/download?wrap=1)

25021 dictionary words, 29 words in file, 4 misspelled

[Potter.txtPreview the documentView in a new window](https://ucdenver.instructure.com/courses/347316/files/4667953/download?wrap=1)

25021 dictionary words, 78452 words in file, 19531 misspelled

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

HASH TABLES

A hash table contains buckets into which an object (data item) can be placed. When a hash function is applied to an object, a hash value is generated. The hash value is used to determine which bucket the object is assigned to.  
  
A bucket is a cluster (or a sub container) that holds a set of data items that hash to the same table location. Obviously, you can not store 25K words in 1373 slots and you need to use some kind of chaining schemes such as linear probing or the second hashing. The size of a bucket is independent from the number of data items you put into the hash. So if you have too many buckets, the hash will not have many collisions but you may waste the storage and you may have to deal with a rather complex hash function and longer keys. If you have too small number of buckets, then you have to deal with frequent collisions. Finding a good bucket number would play an important role in reducing collisions. That's why we usually pick a prime number for the number of bucket. We picked 1373 for the bucket number.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IMPLEMENTATION

Implement with an array (SIZE=1373) of linked lists. Your linked lists should contain the word that was hashed to that array. When you land on a particular array cell (equal to the hash of the word), traverse the linked list until you either find the word, or the nullptr...then add the word.  (You can use the STL list if you choose). For your hash function, you will be hashing strings.

To get a hash string you should declare a variable something like this:  
**hash<string> hashStr;**

then when you want to hash a particular string (let's say called string1)

**hashStr(string1);**

This "function" will produce a long data type (which you should mod by 1373)

You don't have to interpret verb tense, plurals, conjugations etc. All you have to do is to check with each word against the dictionary.