Assignment 2: Word Completion

Out: 10th February 2014 **Due: 24th February 2014 at 11:55pm**

In most word processors and browsers, automatic word completion is a common feature. This works by using a dictionary to determine which words start with the set of letters typed by the user. In this assignment you will write a program that uses a linked data structure to implement such word completion.

A single node of this class contains an array of length 26, one element corresponding to each letter of the English alphabet. Every entry in this array contains a list of words that contain the prefix ending at this node, and a pointer to another such node. The actual linked data structure remembers simply the first of such nodes, called the "root".

To add a new word "apple", we start at the root and look at the entry corresponding to 'a' (which will be at index 0). We travel to this node and add the word "apple" in it. At this node we look at the entry corresponding to 'p' (at index 15). We travel to that node and add the word "apple" to it as well, and follow its link to the next node. We continue for the remaining letters 'p', 'l', 'e' in this fashion, adding the word "apple" in each visited node. At any point if a link points to NULL, we create a new node and add it there before visiting it.

When the user types some letters, say "app". We again start at the "root" and look for 'a', following its link and looking for the entry 'p', following its link and looking for 'p'. As the prefix has now ended, the words at the current node should contain all words added to the data structure that begin with the prefix "app" (e.g. apple, application, apply, etc.). Printing these words will complete the word-completion operation.

1. Write the class "PrefixNode" that represents one node of this data structure. It should contain the array of 26 objects, each of which consists of a bunch of words and a link to another PrefixNode object. Keep in mind that a node may contain an arbitrary number of words, so use the appropriate data structure. You are permitted to use STL classes only for this purpose. Initially each of the 26 links should be NULL, and all bunches of words should be empty.
2. Write the class "WordCompleter" that keeps track of just one PrefixNode object, its root. This class should have the following functions:
   a. A default constructor that initializes the word completer by making its root equal to a blank PrefixNode object.
   b. "add(string s)": this adds the given word 's' to the word completer, creating/filling in the nodes as necessary. **It should not add the word if it is already present.**
   c. "search(string prefix)": this searches for the prefix "prefix" in the word completer, and return a vector of strings corresponding to all words that start with this prefix.
   d. "find(string word)": this function returns true if it found the given word in the word completer, false otherwise.

   You are allowed to add any other private functions as you see fit, but the above should be the only public functions this class offers. Also remember to manage memory properly!
3. Write a main function that reads in the provided dictionary file, <u>removes any punctuation marks</u> from it, converts it to lower case and adds the words in an initially empty WordCompleter object. It should then prompt the user to enter a word and print (a) whether the word was found, and ONLY IF NOT (b) the number of possible words beginning with that prefix (c) the words themselves. The search should be "case-insensitive", i.e. it should not matter whether the user typed in capitals or not. It should continue prompting the user to enter words until the user enters a blank word (i.e. directly presses the "Enter" key without typing in any letters).

Please make sure that the program you submit asks for and prints exactly what is specified, nothing else (so no debugging error messages).

What to submit

Submit all .h and .cpp files, the provided dictionary file and a Makefile as a single zipped file on Reggienet. Please make sure that your code is properly indented and commented (you will lose points for submitting unreadable code, which includes non-existent indentation). **Declaring any public variables in a class will also cost you points!**