Autocomplete

Background Information

Most phones and devices will offer suggestions for possible words that you mean to write, even before you finish typing! In this assignment, you will build an autocomplete predictor using the **Trie** data structure. The user will be able to enter an incomplete word, and your autocomplete predictor will offer up to **N suggestions** for "close" words. Close words are legal words in requiring the fewest number of additional characters, sorted first by length, and then alphabetically.

Efficiency of space and runtime are of extreme importance to us for this project. Take very careful consideration in your design choices. **Correctness** for this project is clearly and objectively defined. Please bear this mind with respect to the portion of your grade defined by correctness. It is encouraged that you communicate with your fellow students to judge each other's correctness.

Trie Requirements

You must implement a class called **Trie** that establishes a tree of **TrieNode**s. The goal of the trie class is to minimize the space requirements of storing an entire dictionary of the English language by exploiting the fact that many words will contain the same prefix. The words "apple", "apply", "applied", and "application" all contain the same prefix, "app", so it would be redundant to store each word in full.

For more information about the Trie data structure see the links below. Some online resources may suggest using a symbol node to indicate the end of a string in a trie. Our goal is to minimize space of our trie, so it is recommended to avoid this method.
https://en.wikipedia.org/wiki/Trie
https://medium.com/algorithms/trie-prefix-tree-algorithm-ee7ab3fe3413 - .lnz2pc73h
http://theoryofprogramming.com/2015/01/16/trie-tree-implementation/

The Trie class interface must provide the following functionality (as well as **default constructor** and **destructor**). Additional methods may be constructed (where logical and efficient) to assist any of the required functions.

1. One parameter constructor — Constructs the trie based on a file stream to a text file.
2. load — Creates a trie based on the given dictionary file. If the current trie is already loaded, then the trie will be cleared before loaded.
3. insert — Inserts the given string into the trie. Returns **true** if the string is inserted; otherwise return **false**.
4. contains — If the trie contains the given string, returns **true**; otherwise, returns **false**.
5. remove — If the given string is in the trie and the string is a word, *remove* the word from the trie and return **true**; otherwise, return **false**.
6. clear — Empties the Trie of all nodes (except the dummy root) and sets the word count to zero.
7. numWords — Returns the number of words stored in the trie.
8. countNodes — Returns the number of nodes that have been created in this trie (excluding the root). *This function must be calculated on demand. It must not return a saved number.*

Dictionary Requirements

The Dictionary class is a wrapper class on top of the trie with a small extension. Ensure that your trie is complete before moving on to the Dictionary class.

The Dictionary class interface must provide the following functionality (as well as **default constructor** and **destructor**). Additional methods may be constructed (where logical and efficient) to assist any of the required functions.

1. One parameter constructor — Constructs the dictionary based on a file stream to a text file.
2. load — Creates a new Dictionary given a dictionary file. If the Dictionary is already loaded, then the Dictionary will be cleared before being loaded.
3. isLegalWord — If the dictionary recognizes the given string, returns **true**; otherwise, returns **false**.
4. numWords — Returns the number of words stored in this dictionary.
5. suggest — Given a string, s, and a number, n, return a vector containing up to n strings. The words within the vector are "close", meaning that they are legal words that require the fewest additional characters. The words should be organized by length, then alphabetically. *If implemented properly, no additional sorting is necessary.*

Example output for Dictionary::suggest(s, n)

Note that "philodina" should come before "philodox" lexicographically, but is not so in this project. In our definition of "close", shorter words should be suggested first.

`suggest("car", 10)`	`car`
	`cara`
	`card`
	`care`
	`cark`
	`carl`
	`caro`
	`carp`
	`carr`
	`cart`

`suggest("philo", 10)`	`philodox`
	`philomel`
	`philonic`
	`philopig`
	`philocaly`
	`philocyny`
	`philodina`
	`philogyny`
	`philohela`
	`philology`

Provided Files

You will be given a dictionary file called *words.txt*. It is a copy of the file found on most Unix based machines at location /usr/share/dict/words. It is formatted file with one word per line. Although some words in the file may not be recognizable, you can be assured that any word in the file provided to you is a legitimate word, and not merely a prefix. Keep this in mind in the previous example of the suggest method.

Neither starter code nor tester code will be provided. It is encouraged that you develop tests on your own. You will be given a working executable for you to query more example outputs.

User Interface

Once your classes are functional, build a main that receives the name of a text file containing words in identical format to the words file provided (i.e. new line delimited words) as a **command line argument**, and constructs a Dictionary object based on the argument. This user interface will continuously request a word and number, and print the resulting suggestions based on those arguments. Due to the simplistic nature of this UI, you may assume error-free input.

Libraries

For this assignment, you are free to use (or not use) any library included in the Standard Template Library (STL) **that we have covered in class**. Your choice in data structures will affect your design and efficiency grade. Choose wisely.

Things To Think About

- The words.txt file has some uppercase letters. How do you want to handle this?
- What does it mean to "remove" from this data structure? What are the implications of deleting nodes?
- Which of these classes need constructors and destructors? Which can be deferred to the compiler?
- In suggest, when you are given a partial word, how can we explore the trie beginning where the partial word ends?

Submission Details

Your submission must include all of the header/source files (*.h/*.cpp) required for your program to properly compile and run. You must include a makefile that can build, run, and clean your project. The name of the generated executable must be **Autocomplete**. Feel free to adapt a makefile that you have used in a previous project. Do not submit any generated executables or object files (*.o). You must also create a README file according to the specifications of the programming rubric. Confirm that your code successfully compiles and runs on the Linux lab machines. See the Programming Rubric for more details.

Due Date

This project must be submitted before 11:55pm, Sunday, May 7th.
Late assignments will receive a zero. No exceptions.