CSCI 235 – Software Design & Analysis II
Assignment 4

Introduction

Before starting this assignment, read the following programming rules:
http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci235/programming_rules.pdf

This assignment tests your ability to design and implement a tree-like data structure called a trie. The name "trie" is short for "retrieve," so is properly pronounced the same as "tree," but many computer scientists pronounced it the same as "try." A trie is often called a "prefix tree" because the descendants of a node share a common prefix. For example, in the figure below (from Wikipedia), the nodes "tea," "ted" and "ten" all share a common prefix, so all have the node "te" as a common parent.

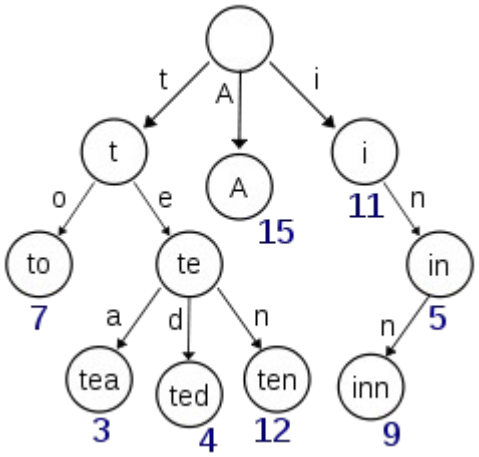

Figure 1: A sample trie from http://en.wikipedia.org/wiki/File:Trie_example.svg

The program must compile and run on the computers in 1000G HN (G-lab). You may work on this program by yourself or in a group. (If you choose to work in a group, each member receives the same grade. The maximum group size is 3 and must be composed of other students taking CSCI 235 this semester.)

Assignment

Create a spell-check program. Your program must read the attached file, create a trie from the data and respond to user queries.

The attached file "sae-sorted.dic" contains the correct spelling for several thousand words in the Standard American English. This is a data file was cobbled together from several available as part of the Jazzy Spell Checker (http://jazzy.sourceforge.net/). This file has one valid word per line in Unix format. Your program must import data from this file and create a trie from the data. If the file contained the following eight entries, the resulting trie would look like Figure 1:

```
a
i
in
```

```
inn
tea
ted
ten
to
```

Note two important details. Firstly, a word (such as “i” or “in”) may be a prefix of one or more other valid words. Secondly, your trie, unlike Figure 1, will probably need to be insensitive to case. Your trie must conform to and implement the attached (“trie.hpp”) class.

Once your program has read the file and imported the data, it should respond to user queries. When your program has determined that the user has misspelled a word, your program must give the user alternatives based on the last valid prefix. For example, if the user types:

```
thisr
```

… your program must provide the following as alternatives:

```
this
thistle
thisledown
thistles
thistly
```

In addition to the alternatives as above, the program must also give the user the opportunity to add the word to the spell-check dictionary. (Your program must do this with the `operator+=` function.)

Your program must be interactive – i.e. must not wait until the user has pressed the [Enter] key to determine whether a word has been misspelled. However, your program may wait until the first whitespace character is pressed. You may use ncurses, specifically the “`getch()`” function for this part of the implementation. You can read more about ncurses here:
http://en.tldp.org/HOWTO/NCURSES-Programming-HOWTO/
If you choose to use ncurses, you can compile it on the lab machines by explicitly linking the ncurses library, for example:

```
g++ -lncurses dictionary.cpp
```

Also indicate that you are using ncurses in the Blackboard submission comments.

Submission

Submit your source code on Blackboard. In the Blackboard submission comments, list all people who worked on the code. If you work in a group, only one team member needs to submit for the group.

Submit all your code and documentation as one “tar.gz” file. A tar file concatenates a bunch of different files (without compressing them). A gz (gzip) file compresses a single file. You can create a tar file (named "a1.tar") in the same directory from three files (main.cpp This.hpp This.cpp) with the following command:

```
tar -cvf ./a1.tar main.cpp This.hpp This.cpp
```

You can gzip the tar file above with the following command:

```
gzip a1.tar
```

This creates a file in the same directory called "a1.tar.gz", which is what you should submit on Blackboard.

Grading

Your grade will be based on the following:

50% = Correctness

The program must compile and run on one of the G-lab machines. In addition, it must perform the functions outlined in the "Assignment" section.

20% = Design

The program must show a reasonable object-oriented decomposition of the assignment into classes. The UML must match the implemented program.

20% = Performance

The implementation must be as efficient as possible in terms of the amount of memory used and in terms of the amount of computational cycles used.

10% = Documentation and style

The implementation must have good comments; variables must have reasonable names, and the submission must have instructions with respect to compilation and execution. The package must be placed on Blackboard as outlined in the "Submission" section.

There is a 10% late penalty per day after the first day.