**Lab Exercise: Tries**

In this exercise you will get acquainted with the Trie data structure. Tries are extremely efficient in storing strings, i.e. sequences of digits of characters from a limited alphabet. In this sense, a Trie may store regular text strings, but also e.g. DNA sequences (consisting of the nucleotides C, T, A and G), long integers (consisting of digits 0-9), etc.

Without loss of generality, this exercise will limit its scope to working with regular text strings.

You are provided with an implementation of a TrieNode and test suites for each set of exercises.

The overall goal for this exercise is to implement a class Trie with the by-now common operations insert(), search() and remove(). The UML class diagram for this class is given below:



(you may of course add private methods, member variables, etc., as you wish)

**Exercise 1:**

Investigate the provided implementation of a Trie and the (inner) class TrieNode. Ensure that you understand…

* Why the class Trie has the constructor it does
* What a TrieNode consists of, specifically what the typedef’s TrieMap and TrieMapIter represent and why they are there.

**Exercise 2:**

Investigate the provided implementation of the private member function

unsigned int findPrefixEnd(string str, TrieNode\*& end).

This method sets end to the last node in the prefix of str in the Trie and returns the length of this prefix.

Ensure that you understand how this method works, what the purpose of the pointer-reference end is, and what it returns.

**Exercise 3:**

Design the method search() in accordance with your design. Hint: Can you use findPrefixEnd() to make this a whole lot easier?

**Exercise 4:**

Design the method insert()in accordance with your design. Hint: Can you use findPrefixEnd() to make this a whole lot easier?

**Exercise 5:**

Implement your designs for insert() and search(). Use the relevant test suite to verify that your implementation works.

*The following exercises are advanced, and you do not need to do them – but hey, now that we’re add it, right? Pour some coffee and get crackin’ ☺.*

**Exercise 6:**

Design the method remove(). Remember to handle all the cases that we have discussed in class.

**Exercise 7:**

Implement your design of remove() in accordance with your design. Use the relevant test suite to verify that your implementation works.

**Exercise 8 (advanced, but fun!):**

Design a new method for the Trie class findAllWithPrefix() that returns all strings with a given prefix. I.e., for the Trie given in the Pair & Share-questions for this lesson, findAllWithPrefix(“an”) should return the strings “ant”, “antibody” and “antilope”.

Implement findAllWithPrefix() and test it using the relevant test suite.