Dictionary - Trie

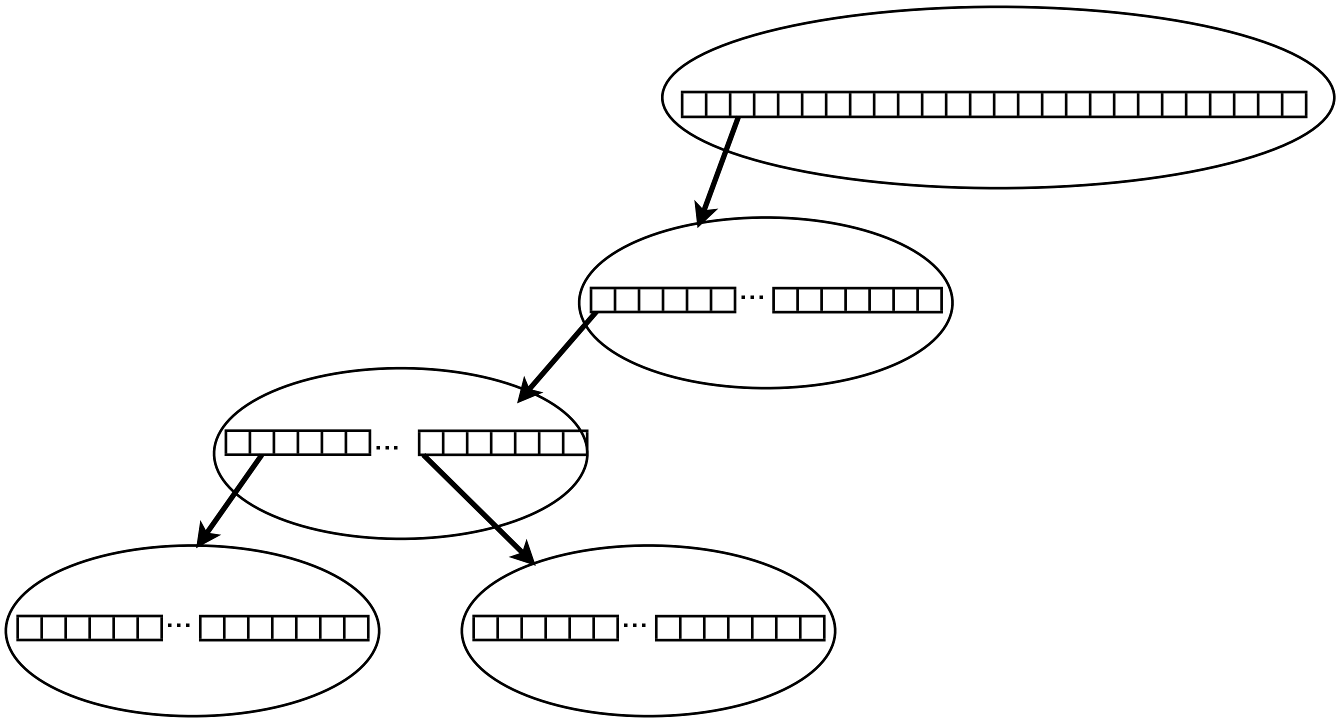
In the first part of project 2 we will implement a dictionary supporting two lookup functions, one to determine if a given string is a word in our dictionary, another to determine if a given string is a prefix of a word in our dictionary. Because we wish to answer the prefix problem quickly we will use a trie data structure to store our dictionary.

Specifically, you will write one class called Dictionary. You will submit both a header and cpp file for this class. You will not submit a main function or any other files. (The class must be named Dictionary). The class will have the following public functions (they must be named exactly as below):

```
bool isPrefix(string s)
bool isWord(string s)
bool insert(string s)
```

It will also have a default constructor defined. You will probably have additional functions in your design, but they will be private.

Recall, that the trie is different than a standard tree. It leverages the fixed character set to store a word. Namely, there are only 26 possible choices for the first character in the string. Similarly, there are 26 possible choices for the second, 26 for the third, and so on. We can leverage this when storing the word. Instead of just left and right pointers we can have 26 pointers. The first pointer would store the subtree containing all words beginning with ‘a’, the second pointer all words that start with ‘b’, the third, words starting with ‘c’, and so on. Moreover, we can do this in a recursive fashion. The nodes in the third level of the tree would store the information regarding the second character in the word, and so on for all characters in the word.



The above nodes depict what the words “cab” and “cat” would look like stored in a trie. Before implementing you may want draw many pictures, for instance, consider what the above would look like if we added the words “cable”, “candy”, and “catenary”.

Case Insensitive:

All of the above public functions are case insensitive, “Cat”, “CAT”, “cat”, “cAT”, are all the same word. In addition, punctuation and numbers are ignored, “c.a.t”, “C/At”, “c8At” are all the same word --- cat. Sorry, Will.i.am.

CCTYPE:

There is a library cctype which contains the following functions you may find useful:

```
bool isalpha(char c) //character is alphabetic

bool isdigit(char c) //character is a decimal digit

bool ispunct(char c) //is punctuation

char tolower(char c) //converts letter to lowercase
```

ASCII:

Our project will assume that strings are encoded using ASCII (sorry IBM). You can use this function to convert a character to an integer. ‘a’ will map to 0, ‘b’ will map to 1, etc. It only works on lowercase alphabetic characters.

```
//lower case alphabetic characters only,
//behavior is undefined for other inputs

int charToInt(char c)
{
      return (c - ‘a’);
}
```

Empty String:

The empty string is a special case. The empty string is a prefix of every word, even if it has not been added to the dictionary using insert. The only way that the empty string is a word is if someone explicitly adds it to the dictionary using insert.

Submissions:

provide comp15 proj2part1 dictionary.h dictionary.cpp