CS 250 (Assignment 05)

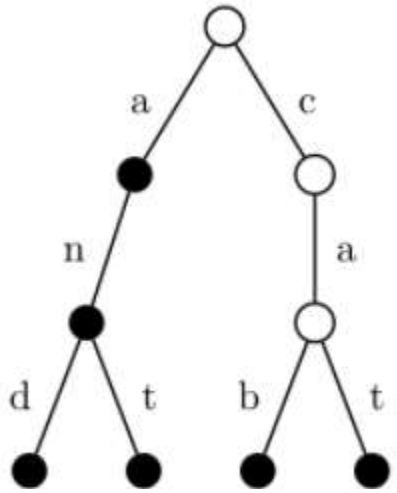
Due Date: On Friday, 15th December (11:55 pm) on LMS.

Note: Your Submissions should be a single zip file. It's a group assignment (max 2 in a group)

Your assignment is to implement a trie in C++. This is a data structure that we have not discussed in class, so this assignment is meant to simulate the sorts of problems you might encounter as a developer: implementing a new data structure for a particular task.

The program context is a spell checker. The need is to store a spelling dictionary so that it is as fast as possible to see whether a word is in the dictionary. The data structure we will use to store the dictionary is called a trie, which is derived from the middle letters of the word *retrieval*. Tries were invented many years ago to speed information retrieval tasks.

A trie is a tree whose edges are labeled with letters. Paths from the root of the trie spell out words. The image below shows a trie.



The black nodes indicate that the path to this point in the trie spells a word. The words represented by this trie are *a*, *an*, *and*, *ant*, *cab* and *cat*. It is very fast to check whether a word is in a trie: one need only start from the root and trace out the word along the edges; if one ends up at a word-terminating node (a black node), then the word is in the trie, otherwise it is not. A trie is very good for spell checking: simply construct a trie representing all the words in a dictionary, and then check each word in a document to see if it is in the trie. The words not in the trie are (presumably) misspelled.

Tries can also efficiently support auto-completion. Given the prefix-based structure of the trie, it is straightforward to enumerate all of the words that begin with a given letter sequence: these will be represented by nodes that are descendants of the node corresponding to that prefix.

You can use the following node definition to implement tries:

```
#define NO_OF_ALPHABETS 26
struct TrieNode
{
        TrieNode * child[NO_OF_ALPHABETS];
        bool isEndOfWord;
}
```

Your task is to implement the following:

- Insert the word in Trie
- Printing all the words in Trie
- Check if a word is present in the Trie or not

Note: Your Trie should contain minimum of 1000 words.