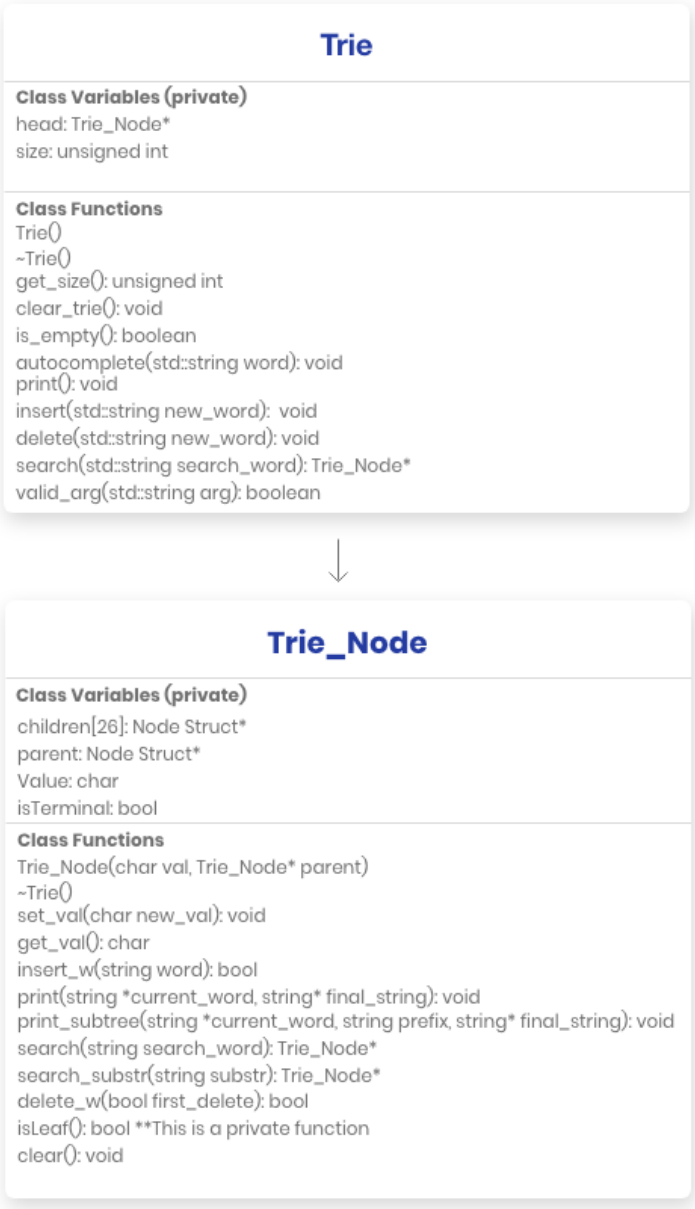
Course: ECE250
Name: Karim Al-Atrash
Student ID/Number: kalatras/20824606

1. A Class Overview

I will be designing 1 class called Trie and another called Trie_Node. The Trie class will store the head of the trie which will be a struct, the number of nodes currently in the trie and will call the recursive functions that are located in Trie_Node when necessary as well as print any results. The Trie_Node class will house the recursive implementations for insert, delete, search, print, autocomplete and clear. Each Trie_Node will have an array of 26 possible children Trie_Nodes, a parent node, a value and a flag to signify whether this node is the end of a word or not.

2. UML Diagram



Course: ECE250
Name: Karim Al-Atrash
Student ID/Number: kalatras/20824606

3. Details on design decisions

Trie Constructor: This constructor will instantiate the root node of the trie.
Trie Destructor: The clear() function will be called to delete the nodes of the Trie.
Search(): This function will return a pointer to the node which denotes the end of the word that has been searched for. This allows for easy access to nodes for auto complete.
Print() and print_subtree(): These functions will take in 2 string arguments, one for keeping track of the current word while recursing and another to keep track of all the words that the function has traversed through. This allows for mutation of the final string in order to remove any extra spaces included.
When to use const: All strings passed through all functions should be immutable since we are assuming all inputs will be of the correct format. Thus, const will be used whenever passing parameters by value to prevent accidental mutations.

4. Test cases

Testing to make sure no illegal words are being stored (no strings with spaces or special characters).

Testing to make sure duplicate words are not stored is necessary.

Testing to make sure no output is created if the prefix provided for autocomplete is not in the trie.

Testing to make sure all words print in alphabetical order.

Testing to make sure insertion and deletion of subwords and superwords (a word containing a subword) function correctly.

Testing to make sure deletion of words that are members of a branch functions correctly.

5. Performance considerations

It is expected that empty(), and size() run in O(1) time. To do this, I have implemented a length counter in the trie class to return the length without having to iterate through the whole list on each call. This makes my size() function run in O(1) time on call.

search() is required to run in O(n) time. I will do this by traversing the tree letter by letter until I reach the end of the word. Autocomplete(), delete() will use the searching algorithm to achieve O(n) runtime.