1. Class Overview

I will be designing 1 class called Trie. The Trie class will store the head of the trie which will be a struct, the number of nodes currently in the trie. The Trie class will also house the recursive implementations for insert, delete, search, print, autocomplete and clear. The node struct for the tree will have member variables for 26 possible children, as well as the current value of the node and its parent.

1. UML Diagram

Graphical user interface, text, application

Description automatically generated

1. Details on design decisions

**Node Struct instead of class**: I chose to use a node struct since I needed all member variables of the node to be easily mutable and there are no functions that need to be implemented within a node object.

**Trie Constructor**: This constructor will be left empty.

**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.

**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.

1. Test cases

Since searching is not case sensitive, I will need to consider the case where the search term differs from the URL name by capital letters.

I will need to test the edge cases for all the pop and push functions. These edge cases include when there are no elements in the list and when there is 1 element in the list.

I will test when there are empty strings and strings with only spaces to ensure my searching works well.

1. Performance considerations

It is expected that all my functions except find(), clear() and print() run in O(1) time. To do this, I have implemented a length counter in the URL\_History 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.

Find(), clear() and print() require an O(n) runtime. This can be achieved by traversing the list and performing the necessary operations to fulfill the functions duties on each iteration. Traversing a list of size n will have an asymptotic time bound of O(n).