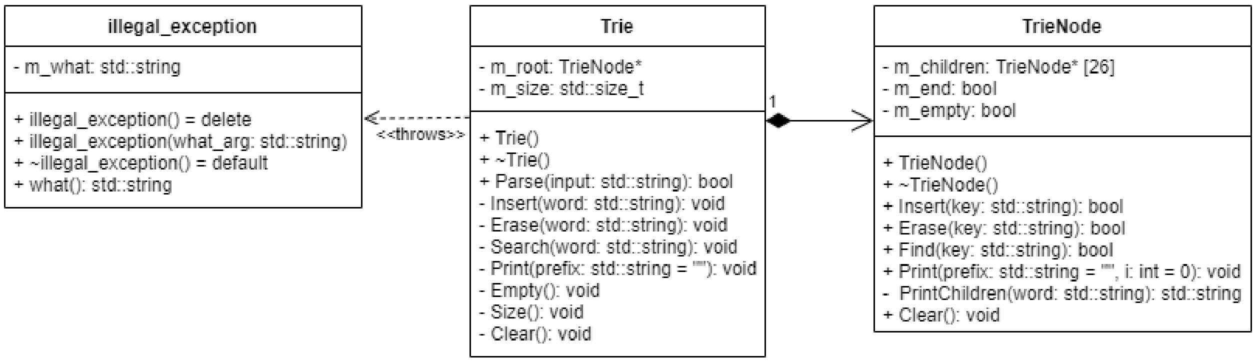
Design Document

Overview
For this project, I designed 3 classes:

Trie
Trie is a class that contains a root TrieNode and keeps track of the number of words in the trie. It also acts as a driver for trie related functions, parsing the input and printing the required output.

TrieNode
TrieNode is a recursive node containing a trie data structure, each node can be considered the root of a Trie, methods called on the root get recursively called down the branches. The class contains pointers to child nodes.

illegal_exception
illegal_exception is an exception class thrown by Trie when there are illegal arguments in the input.

UML Class Diagram

illegal_exception
- m_what: std::string
+ illegal_exception() = delete
+ illegal_exception(what_arg: std::string)
+ ~illegal_exception() = default
+ what(): std::string
<<throws>>
Trie
- m_root: TrieNode*
- m_size: std::size_t
+ Trie()
+ ~Trie()
+ Parse(input: std::string): bool
- Insert(word: std::string): void
- Erase(word: std::string): void
- Search(word: std::string): void
- Print(prefix: std::string = ""): void
- Empty(): void
- Size(): void
- Clear(): void
1
TrieNode
- m_children: TrieNode* [26]
- m_end: bool
- m_empty: bool
+ TrieNode()
+ ~TrieNode()
+ Insert(key: std::string): bool
+ Erase(key: std::string): bool
+ Find(key: std::string): bool
+ Print(prefix: std::string = "", i: int = 0): void
- PrintChildren(word: std::string): std::string
+ Clear(): void

Design Decisions

Trie
This class initializes member variables and dynamically allocates a root node in the constructor, and deallocates the root node in the destructor. No operator overloads were necessary.

bool Parse(std::string input) void Insert(std::string word) void Erase(std::string word)

No const keywords are used for these functions because using a value parameter as input allows std::move(), and they all modify members.

void Search(std::string word) const void Print(std::string prefix = “”) const

These functions also use a value parameter to allow std::move(), but they do not modify members, so a const is added.

void Empty() const void Size() const

These functions have no parameters and do not modify members, so a const is added.

void Clear()

This function has no parameters and modifies members, const is not used.

TrieNode

This class initializes member variables in the constructor and calls the clear function in the destructor to deallocate branch nodes. No operator overloads were necessary.

bool Insert(std::string key) bool Erase(std::string key)

No const keywords are used for these functions because using a value parameter as input allows std::move(), and they all modify members.

bool Find(std::string key) const void Print(std::string prefix = “”, int i = 0) const

These functions also use a value parameter to allow std::move(), but they do not modify members, so a const is added.

std::string PrintChildren(std::string word) const

This function recurses onto itself while using std::move(), effectively passing the same string all the way down and back up, thus both parameter and return value are passed by value. It does not modify any members, so a const is added.

void Clear()

This function has no parameters and modifies members, const is not used.

illegal_exception

This class forbids the use of the default constructor because there should always be an error message when an exception is thrown, a constructor is created with an std::string parameter containing the message. The default destructor is used since no dynamic allocation was used. No operator overloads were necessary.

const std::string& what() const

This function returns the error message, the error message should never be modified, so a const reference is returned. The function will not modify the error message, so const is added.

Test Cases

- There is not a requirement to handle invalid commands, so all invalid commands should be ignored.
- The parameters used for the ***i***, ***e***, and ***s***, functions should check and throw an illegal_exception. The trie should not be modified if the exception is thrown.
- The parameter used for ***autocomplete*** can contain illegal arguments, in such a scenario, the function should stop with no error, and no messages should be printed.
- The ***print*** and ***autocomplete*** functions should not print any extra spaces, and if nothing is printed, should not print an end line.
- When ***e*** is called, unneeded nodes should be erased, and nodes that are still needed should remain.
- When ***i*** is called with an illegal input, no new node should be created.
- When ***size*** is called, the return value should be the number of words, not the number of nodes.

Performance Considerations

Let n = the number of characters in a word, and N = the number of nodes in the trie.

size empty

These two commands have a runtime of O(1). In my implementation, these two commands simply return a stored value, which is constant time.

i e s

These commands have a runtime of O(n). In my implementation, all of these commands each call on a recursive function. The functions recurse n times, once for each character in the word, and each recursion does constant time operations such as creating a new node or deleting a node. Thus, the functions are O(n).

print autocomplete clear

These commands have a runtime of O(N) since the traversal of all nodes is O(N). In my implementation, *clear* traverses all nodes in order to deallocate every node, the deallocation of a node is constant time; *print* traverses every node in order to find print every word in the trie; and in the worst case scenario *autocomplete* starts printing at the root node, which is O(N).