Design document

Overview of Class:

1. class illegal_exception:
   This class throws an error message of "illegal argument" when the input word has other alphabets other than lowercase alphabets.
   - Public:
     - function: void printException(); //print out an error message: illegal argument
   - Private:
     No private argument
2. class Node:
   This class creates nodes for Trie.
   - Public:
     - Function:
       Node(); ~Node(); initialize all variables and destruct the node to avoid memory leaks
       Variables:
       Node* next[26]; represents 26 children of each node
       bool isalphabet; shows whether that node/leaf is the end of a word (1: true; 0: false)
       string s; the word stored in that node
   - Private:
     No private argument
3. class Trie:
   The class implement a trie
   - Public:
     - Function:
       Trie(); ~Trie();
       used to initialize variables in the class, and delete the node root to avoid memory leak
       std::string insert(Node* &root, std::string word);
       Used to insert the word that is written in lowercase, and is not inserted before into the trie.
       std::string search(Node* root, std::string word);
       Find the word that is written in lowercase by traversing the trie by the alphabet of the word one by one.
       std::string erase(Node* root, std::string word);
       Delete the word that is written in lowercase if the input is legal, the trie is not empty, and the word exists in the trie by setting the isalphabet into false and delete s.
       void count();
       Print out the number of words in the trie
       std::string empty();
       Check if the trie is empty by checking if the number of words is 0.
       void print(Node* root);
       Print all the words alphabetically in the trie using depth-first traversal using iteration.
       void spellcheck(Node* root, std::string word);
       Store all the words into a vector, and print out the words that contain the most prefix with the word inputted. If the first alphabet in the word is not in the trie, there will be no output.
       std::string clear(Node* root);
       The function deletes call words by setting the isalphabet to false.
   - Private:
     - Function:
       void helper_check(Node* a); used to traverse all the words in the trie and store them. Used in spellcheck
       int get_size(); used to get the number of words in the trie
       Variables:
       Node* root; node of the root in the trie
       int num; number of words in the trie
       std::vector<std::string> words; used to store all the words in the trie

Detail on design decision:

Class illegal_exception:
No constructor or destructor: only used to throw error and give error message
Class Node
One constructor and one destructor is created. initialize the node next[26] to NULL, isalphabet to false, and s to "".
Free the node after use to avoid memory leaks
Class Trie
One constructor and one destructor is created. used to set the root of the trie, and initialize the variables in the class; free the root to avoid memory leaks.
No operator being override

Function parameter:

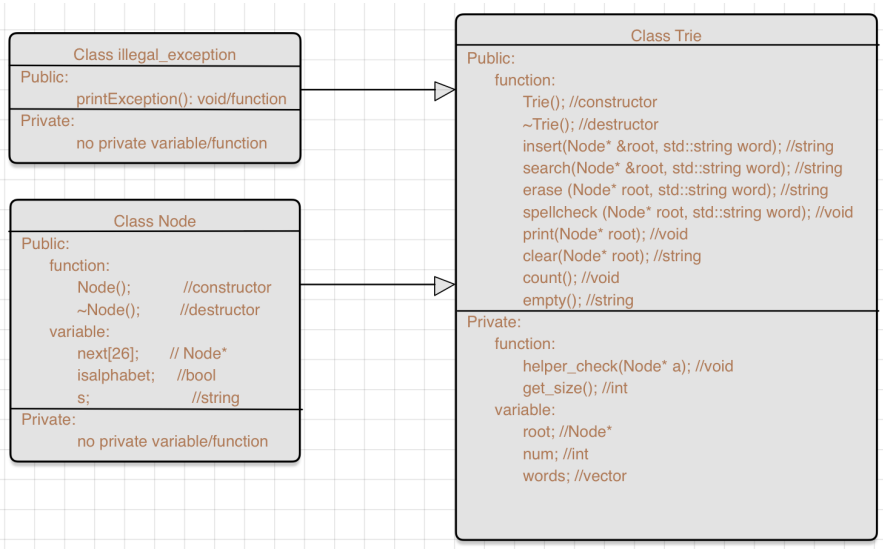
Function without parameter:
Class illegal_exception: printException();

Class Node: Node(); ~Node();
Class Trie: Trie(); ~Trie(); count(); empty(); get_size()

Function with parameter:

insert(Node* &root, std::string word);
the parameter root and word are passed by reference to insert the word into the trie
search(Node* &root, std::string word);
the parameter root and word are passed by reference to search the word into the trie
erase (Node* root, std::string word);
the parameter root and word are passed by reference to erase the word into the trie
spellcheck (Node* root, std::string word);
the parameter root and word are passed by reference to check if the word is spelled correctly
print(Node* root);
the parameter root is passed by reference to print all the words in the trie
clear(Node* root);
the parameter root is passed by reference to delete all the words in the trie

UML diagram



Test cases:

1. Check if the words in the corpus.txt can be stored into the trie correctly. **2.** Check the trie will store duplicate words illegal words. **3.** Search if the word already inserted can be found. **4.** Check if the word in the trie can be erased. **5.** Check if the spellcheck can print out all the words with the same prefix as the word inputted. **6.** Check if all the words can be print out. **7.** Check if all the words can be deleted using clear(), and empty() functions. **8.** Check if the number of words is correct.

Performance consideration:

insert(Node* &root, std::string word);
search(Node* &root, std::string word);
erase (Node* root, std::string word);
} O(n): insert/search/erase the word by traversing each alphabet in the word using for loop.

spellcheck (Node* root, std::string word);
print(Node* root);
clear(Node* root);
helper_check(Node* a);
} O(N): spellcheck/print/clear/helper_check the word by traversing each word using for loop.

count(); print the private variable num
empty(); check the private variable num to see if the trie is empty using if/else
get_size(); return the private variable num
} O(1)

} Trie