Design Document
Jonathan Lam

I designed a Trie class for the trie data structure, a TrieNode class for each node of the trie, and a IllegalException class for handling exceptions.
trie.hpp is the header file containing the class declarations.
trie.cpp contains the class definitions.
trietest.cpp contains main() and success(), failure() helper functions.

The member variables for the IllegalException class are:

- *string msg;* Stores string "illegal argument" for illegal exception.

Member variables are private to avoid unintentional assignment to variables within main().

The member functions for the IllegalException class are:

- *IllegalException;* Default constructor. Assigned msg with "illegal argument". Has a runtime of O(1).
- *~IllegalException;* Default destructor. Has a runtime of O(1).
- *string what();* returns msg. Has a runtime of O(1).

The member variables for the TrieNode class are:

- *bool end;* Boolean value for end of word.
- *TrieNode *parent;* Trienode* pointer to the parent of the node.
- *TrieNode *children[ALPHABET];* Array of TrieNode* pointers that stores the pointers to the child nodes. ALPHABET is a constant value of 26.

end and **parent* member variables are private to avoid unintentional assignment to variables within main(). **children[]* is public as it has to be accessed and changed in Trie often.

The member functions for the TrieNode class are:

- *TrieNode(TrieNode *trienode);* Constructor. Takes TrieNode* pointer and assign to *parent* variable. Set *end* to false. Initialize *children[]* with nullptr's.
- *~TrieNode();* Default destructor. No deallocation needed in TrieNode in my implementation.
- *bool getend();* Returns *end* boolean. Has a runtime of O(1).
- *void setend(bool end);* Sets *end* boolean with end parameter. Has a runtime of O(1).
- *TrieNode *getparent();* Returns *parent* TrieNode* pointer. Has a runtime of O(1).
- *void setparent(TrieNode *parent);* Sets *parent* TrieNode* pointer with parent parameter. Has a runtime of O(1).

The member variables for the Trie class are:

- *TrieNode *root;* TrieNode* pointer for the root of the trie.
- *unsigned int words;* unsigned int of number of words in the trie.

Member variables are private to avoid unintentional assignment to variables within main().

The member functions for Trie class are:

- *Trie();* Constructor. Initializes *words* and assigns to 0. Set *root* to dynamically allocated new TrieNode object with nullptr in parameter as parent.
- *~Trie();* Deconstructor. Sets *words* to 0. Calls clear(*root*) function to deallocate the whole trie from *root*.
- *TrieNode *getroot();* Returns TrieNode* pointer *root*. Has a runtime of O(1).
- *void setroot(TrieNode* root);* Assigns TrieNode* pointer parameter root to *root* of trie. Has a runtime of O(1).
- *unsigned int getwords();* Returns unsigned int words. Has a runtime of O(1).
- *void setwords(unsigned int words);* Assigns unsigned int parameter words to *words* of trie. Has a runtime of O(1).
- *bool insert(string string);* Boolean function that returns true if the word does not exist and false if the word exists in trie already. Called in main for i command to print success or failure depending on the boolean return of the insert function. Set temporary boolean exist variable to true and temporary TrieNode* current pointer to the root of the trie. For every character in string parameter, get the index by subtracting character with 'A' for the ASCII value. If current->children[index] is nullptr, which means the character does not exist at that node in the trie yet, assign that node with a dynamically allocated new TrieNode object with a parent parameter passed with current pointer for a new node in the corresponding index. Set exist to false since the word does not exist as a new node needs to be created. Set current to current->children[index] to traverse through the trie. Once the for loop has gone through all characters in the string, if exist is still true and the current pointer's end boolean is true, this means the word passed into insert already exists in the trie and the function returns false. Otherwise, set the current pointer's end boolean to true to signify end of word, increment the word count, and return true. Has a runtime of O(n) where n is the number of characters in the word. For loop loops through at most all characters in the word.
- *TrieNode *search(string string, bool helper);* TrieNode* pointer function that returns nullptr or pointer to relevant TrieNode pointer that also serves as a helper function determined by the boolean helper parameter. Assign temporary TrieNode* current pointer to root. For every character in the string parameter, get the index by 'A' similar to the insert function. If the current->children[index] is nullptr, that means the current character does not exist in the trie and the function returns nullptr. current = current0>children[index] to increment to the next node. Once the for loop has gone through all of the characters in the string, if the function is used as a helper function (helper boolean is true), the function returns the current pointer. Otherwise, if current is the end of word, return the current pointer since the word exists, otherwise return nullptr as the word does not exist. Has a runtime of O(n) where n is the number of characters in the word. For loop loops through at most all characters in the word.

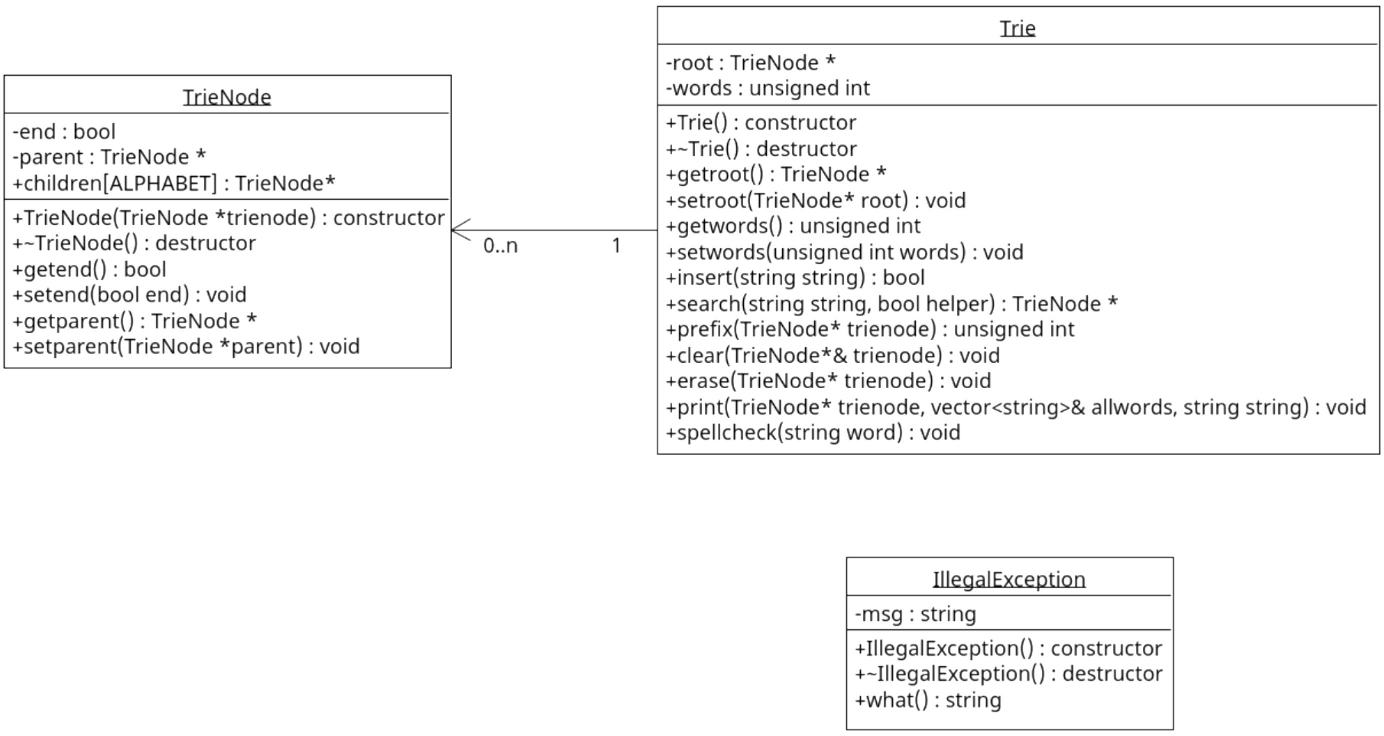
- *unsigned int prefix(TrieNode* trienode);* Returns the number of words (end of words) that start with the TrieNode* pointer parameter. Command c calls prefix function with the TrieNode* pointer passed as the TrieNode* pointer from the search function for the prefix as a helper function. For loop that goes through all indices of the children[] array of trienode. If children[i] is not nullptr, N increments with the result of a recursive call of the prefix function. Outside of the for loop, if the trienode is an end of word, N is incremented and returned. From all of the recursive calls of the function, all nodes child of the original trienode pointer passed that is an end of word are added together to N and returned at the end. Has a runtime of O(N) where N is the number of words in the trie. Recursive calls for at most the number of end-of-words found in the trie.
- *void clear(TrieNode*& trienode);* Clear function is used in the destructor of the trie and for erasing words. Recursively deallocates TrieNode* pointer passed by reference and all children nodes of the pointer passed in the parameter. For loop to find trienode->children[i] that is not nullptr, recursive call the clear function. Outside of the for loop, deallocate trienode and set to nullptr. Has a runtime of O(n) where n is the number of nodes in the trie. Recursive calls for the number of nodes found after the node in parameter, at most the number of nodes in the trie.
- *void erase(TrieNode* trienode);* 4 cases for erase. 1st case is if the word to erase is in the middle of another word. 2nd case is if the ending character(s) of the word branches off of another word. 3rd case is if the trie only contains the word to be erased. 4th case is part of the word to be erased is another word. Command e calls erase function with the TrieNode* pointer of the result of the search function of the word, which is the pointer to the last node of the word.
  For the 1st case: if there are more than 0 child nodes of trienode, set the end of word boolean to false, decrement words, and return.
  For the 2nd case: set temporary TrieNode* pointers for parent and current nodes. While parent is not nullptr, which means current is not the root of the trie (3rd case), and parent is not an end of word (4th case), for loop to find if there is a child node of parent that is not the current node. If that is true, it is the 2nd case. The child node of the parent that points to the current node is set to nullptr, call the clear function for the current node, and decrement words. Otherwise set current to parent, and parent to the parent of parent. Continue the while loop.
  For the 3rd case: while loop condition failed. If parent is the nullptr, which means current is the root of the trie, clear root and set root to a new dynamically allocated TrieNode with nullptr parent (new root). Words = 0 and return.
  For the 4th case: while loop conditioned failed and current is not root, then parent is an end of word. The child node of the parent that points to the current node is set to nullptr, call the clear function for the current node, and decrement words.
  Has a runtime of O(n) where n is the number of characters in the word. While loop loops through at most the number of characters in the word.
- *void print(TrieNode* trienode, vector<string>& allwords, string string);* Print function prints all words from the TrieNode* pointer passed in the parameter. Uses depth-first traversal. Used in command p by passing the root of the trie into the function, and used in spellcheck to print words with specific prefixes. Assign temporary TrieNode* pointer current to trienode. If current is the end of a word, append *string* into *allwords* vector of strings. For every child node of current, recursively call print with parameters, trienode->children[i], the same *allwords* vector of strings passed by reference, and *string* concatenated with the character that corresponds to the child node. Once all recursive calls are complete, the vector of strings will have a vector of words stemming from the trienode pointer parameter. Has a runtime of O(N) where N is the number of words in the trie. Recursive calls for at most the number of end-of-words found in the trie.
- *void spellcheck(string word);* spellcheck function takes a string *word* input. Spellcheck has 4 cases. 1st case is input string exists as a word in the trie. 2nd case is the input string is part of existing word(s) in the trie. 3rd case is part of the input string is the prefix of existing word(s). 4th case is there is no matching first character of the input string in the trie.
  For the 1st case: call search function for *word*. If it returns a not a nullptr, that means *word* matches with a word in the trie. Print “correct”.
  For the 2nd case: call search function for *word* as a helper function. If the function returns not a nullptr for *input*, it is the 2nd case described above. In the if condition, initialize temporary vector of strings *str* and call the print function on *input* and use a for loop to print out the vector of strings from the result of the print function with *word* concatenated at the front.
  For the 3rd case: *input* returned from the result of the search function as a helper function will be nullptr for the 3rd case. In the while loop, it will keep popping back characters and calling search function as a helper function to check until it does not return a nullptr, indicating it is a word or prefix of a/multiple word(s). Similar to the 2nd case, print function will be called with the prefix appropriate for spellcheck and all appropriate words for the input for spellcheck are printed out.
  For the 4th case: once *word* has all its characters removed in the while loop, meaning there is no match for the first character of the input to the words in the trie, it is the 4th case. Then an end of line character is printed. Has a runtime of O(N) where N is the number of words in the trie. Prints at most the number of words in the trie.

In main():

- *load command;* loads corpus into trie by calling insert for each word in corpus.
- *i command;* try-catches illegal arguments. Calls insert function for word parameter. Prints successful if insertion was successful. Prints failure if the word already exists in the trie. Has a runtime of O(n) where n is the number of characters in the word.
- *c command;* try-catches illegal arguments. Calls prefix function with the result of the search function as a helper function with word parameter. Prints the number of words with prefix if prefix is found. Prints not found if no words with the prefix is in the trie. Has a runtime of O(N) where N is the number of words in the trie.

- *e command;* try-catches illegal arguments. Calls erase function with the result of the search function with word parameter. Prints failure if the word does not exist in the trie. Prints success if the word is erased. Has a runtime of O(n) where n is the number of characters in the word.
- *p command;* Calls print function with a temporary vector of strings passed by reference, root of the trie, and an empty string passed as parameters. Prints out the strings in the vector with an end of line character. Has a runtime of O(N) where N is the number of words in the trie.
- *spellcheck command;* Calls spellcheck function with word parameter. Prints correct if the input exists as a word in the trie. Prints all words that start from the first letter with an error if the word is not in the trie. Prints an empty line if the first character does not exist in the trie. Has a runtime of O(N) where N is the number of words in the trie.
- *empty command;* calls getwords function to get the number of words in the trie. Print empty 1 if the trie is empty, print empty 0 if the trie is not empty. Has a runtime of O(1).
- *clear command;* Set word count to 0. Calls clear function with the root of the trie. Set root to a new TrieNode pointer as the root of the trie. Prints success. Has a runtime of O(N) where N is the number of words in the trie.
- *size command;* Prints the number of words of the trie. Has a runtime of O(1).
- *exit command;* Breaks the while loop and ends the program.

UML Diagram:


Trie
-root : TrieNode *
-words : unsigned int
+Trie() : constructor
+~Trie() : destructor
+getroot() : TrieNode *
+setroot(TrieNode* root) : void
+getwords() : unsigned int
+setwords(unsigned int words) : void
+insert(string string) : bool
+search(string string, bool helper) : TrieNode *
+prefix(TrieNode* trienode) : unsigned int
+clear(TrieNode*& trienode) : void
+erase(TrieNode* trienode) : void
+print(TrieNode* trienode, vector<string>& allwords, string string) : void
+spellcheck(string word) : void
TrieNode
-end : bool
-parent : TrieNode *
+children[ALPHABET] : TrieNode*
+TrieNode(TrieNode *trienode) : constructor
+~TrieNode() : destructor
+getend() : bool
+setend(bool end) : void
+getparent() : TrieNode *
+setparent(TrieNode *parent) : void
0..n
1
IllegalException
-msg : string
+IllegalException() : constructor
+~IllegalException() : destructor
+what() : string

Test Cases:

Tested normal operations of all commands. Tested for correct insertion. Tested for the correct number of prefixes. Tested for illegal arguments try-catch. Tested all 4 cases of erase and the robustness of each case. Made sure that nodes are deallocated and children pointers are set to nullptr accordingly to avoid memory leaks and segmentation faults. Tested printing all words in trie with empty trie, trie with one word, and trie with multiple branches. Tested spellcheck for all 4 cases and spellcheck in conjunction with erase function for robustness. Tested clear command in series and multiple clear and loads. Tested for the correct number of nodes after multiple inserts and erases.