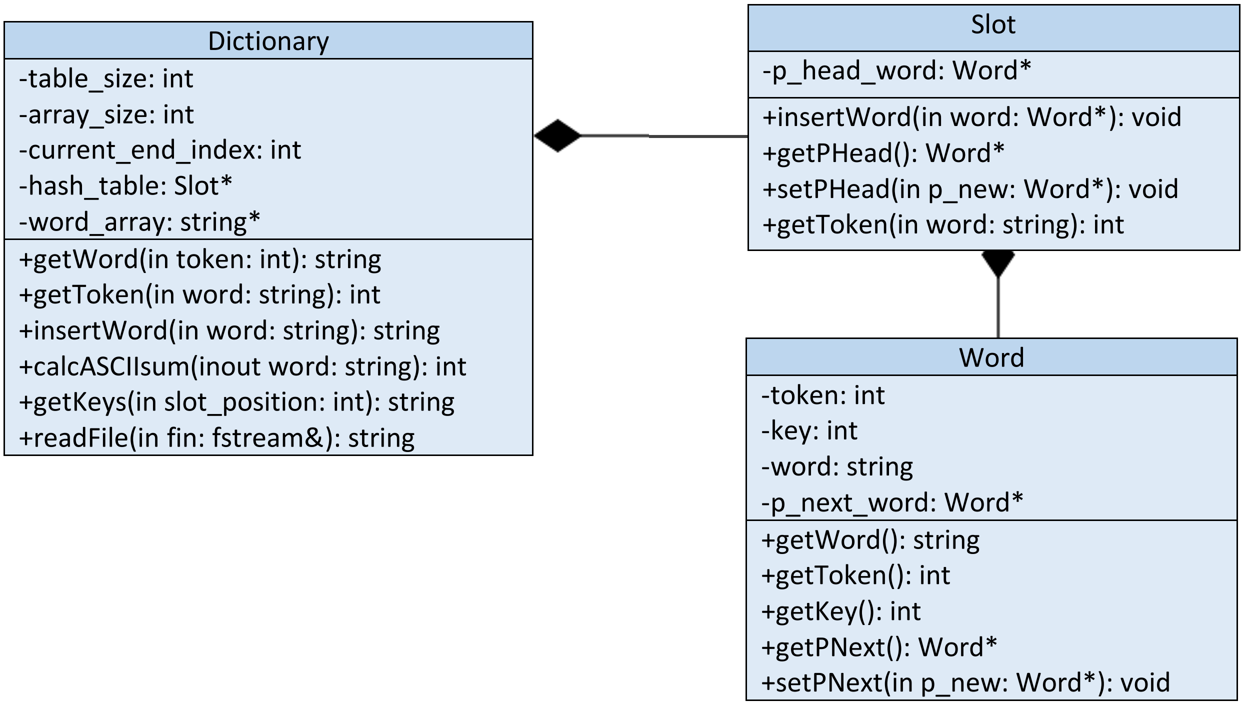
LLM Tokenization (Project 2)

20992999 Kaixuan Chen

UML Diagram


Dictionary
-table_size: int
-array_size: int
-current_end_index: int
-hash_table: Slot*
-word_array: string*
+getWord(in token: int): string
+getToken(in word: string): int
+insertWord(in word: string): string
+calcASCIIsum(inout word: string): int
+getKeys(in slot_position: int): string
+readFile(in fin: fstream&): string
Slot
-p_head_word: Word*
+insertWord(in word: Word*): void
+getPHead(): Word*
+setPHead(in p_new: Word*): void
+getToken(in word: string): int
Word
-token: int
-key: int
-word: string
-p_next_word: Word*
+getWord(): string
+getToken(): int
+getKey(): int
+getPNext(): Word*
+setPNext(in p_new: Word*): void

Class Functions

Word Class

Each object is a word stored in the hash table. Since collision is resolved using chaining, each Word object is also an element in the linked list.

The functions are self-explanatory, i.e. getWord() returns the "word" attribute, setPNext() sets the "p_next_word" attribute, etc. Every attribute is initialized in the constructor.

Slot Class

Each object represents a slot in the hash table. Collision is resolved using chaining, so each Slot object is a linked list.

Function	Behavior
Word* getPHead()	Return the p_head_word attribute.
void setPHead(Word* p_new)	Set p_head_word to p_new, this is a helper function for insertWord().
void insertWord(Word* word)	Insert the word object at the start of the linked list, setting p_head_word and word.p_next_word accordingly.
int getToken(string word)	Iterate through the linked list, compare each element's "word" attribute with the function parameter. If there is a match, return the token of that element. Otherwise, return 0 to indicate the given word is not stored in this linked list.
Slot()	In the constructor, set p_head_word to nullptr.
~Slot()	Since each Word object is created dynamically, in the destructor go through the linked list and delete each element.

Dictionary Class

This is the storage container of unique words. It contains a string array "word_array" that stores the words, as well as a hash table "hash_table" that stores Word objects, using the Slot class as table slots. For each Word, the key is the ASCII character sum, and the token is their respective index in the string array.

Function	Behavior
string getWord(int token)	This function is called to find the word with the given token. It is called to answer the “RETRIEVE” input command, and called continuously to answer “TOKS”. If the token is not valid, return “UNKNOWN”. Otherwise, return the string at index token of word_array.
int getToken(string word)	This function is called to find the token of the given word. It is called to answer the “TOKENIZE” input command, and called continuously to answer “STOK”. Call calcASCIIsum(word) to get the key of the word, and use the given hash function to get an index in the hash_table. Call getToken(word) of the Slot object at the index, and return that value.
string insertWord(string word)	This function is called to insert a word into the dictionary. It is used to answer the “INSERT” input command. Call calcASCIIsum(word) to get the key, return “failure” if the key is 0 (non-alphabetic) or if getToken(word) is non-zero (already stored). If word_array is full, create a new string array with array_size + 100, copy over all elements, and set this as word_array, deleting the old array. Increase current_end_index by one, then create a new Word object with token equal to current_end_index. Finally, insert this object into a slot in the hash_table, onto the end of word_array, and return “success”.
string getKeys(int slot_position)	This function is called to print all keys of a slot in the hash table. It is used to answer the “PRINT” command. Use slot_position as index in the hash_table to reach a Slot object. Iterate through the linked list and append each element’s key to the end of a string. If the string is empty at the end, return “chain is empty”. Else, return the string.
string readFile(fstream& fin)	This function is called to insert the words from a file. It is used to answer the “READ” input command. Using <fstream>, read the file and call insertWord() on each word encountered. If any insertWord() returns “success”, return “success”. Return “failure” otherwise.
int calcASCIIsum(string word)	Helper function to sum the ASCII value of each character. Iterate through the word character by character. If a non-alphabetic character is encountered, return 0. Otherwise, add the character’s ASCII value to a sum and return the sum once the whole word is finished.
Dictionary(int size)	Initialize array_size to a predetermined value, 100 in this project, and table_size to size. Create a new Slot array of size table_size and a new string array of size array_size, and assign them to identifiers hash_table and word_array, respectively.
~Dictionary()	Delete the two dynamically allocated arrays, hash_table and word_array.

Runtime Requirements

RETRIEVE command calls the getWord() function of Dictionary, and it has runtime O(1) since retrieving from index in an array is always constant.
TOKENIZE command calls the getToken() function of Dictionary, and it has runtime O(1) on average. Assuming uniform hashing, on average, for input size n and table size m, n/m operations are performed each call, i.e. constant operation time for each function call.
INSERT command calls the insertWord() function of Dictionary class, and it has runtime O(1) on average. Constant operations are performed to add the word to word_array, and assuming uniform hashing, constant operations (n/m) are performed to add the word to hash_table. The worst case occurs when word_array is full, in which case each element has to be copied once before the insertion can happen, i.e. O(n) runtime.