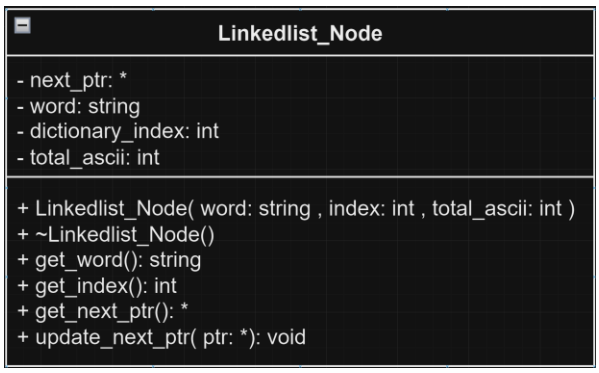
ECE250 LAB2 Design Document

Author: Bowen Zheng, student #: 20949303, student ID: b57zheng

Linkedlist_Node Class:

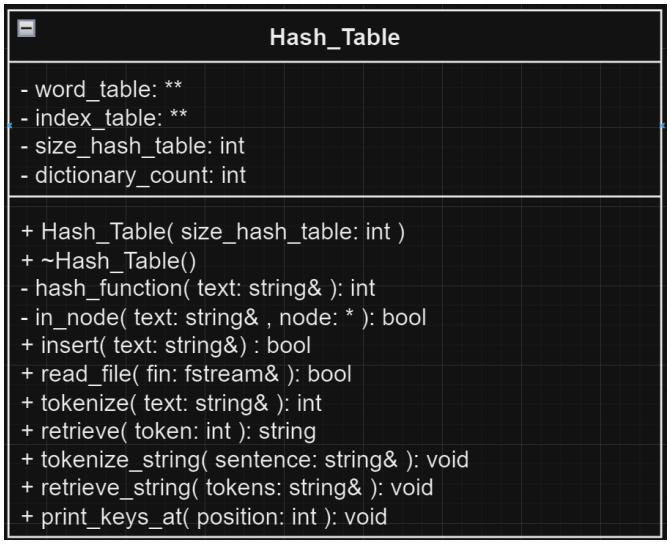
- Class UML:



- Constructor:
  Initialize word, token and the total ascii number of the word in the node. These elements will be saved in the node for future use.
- Destructor:
  We will use the default destructor for this class as there is no dynamically allocated memory in this class needs to be released at the end of the programs.
- Overall:
  The functions get_word(), get_index(), get_next_ptr() are implemented to help me access the corresponding private variables in this class
  The function update_next_ptr was implemented to set the next node in the linked list for separate chaining during collision.

Hash_Table Class:

- Class UML:



- Hashing functions:

  word_table hashing function: h(k) = total ascii of the word % size of hash table

  index_table hashing function: h(t) = token of the word % size of hash table

- Constructor:
  The constructor initializes all private member variables, initialize word hash table to null pointer and initialize index hash table to null pointer.

- Destructor:
  The destructor first releases the memory allocated to the word hash table, then releases the memory allocated to the index hash table. This will avoid memory leaks.
- Function: hash function:
  This is a helper function that calculates the word_table hashing function.
- Function: in_node
  This is a boolean helper function that determines if a word is already existed in the dictionary.
- Function: insert
  This function is implemented for command “INSERT”, it returns an boolean value and insert word into the dictionary.
    - Run time analysis:
        - The runtime for calling the hash function, calling the in node function is O(1).
        - Assume uniform hashing, we already knew where the word will be inserted by calculating the hash function; If there is collision, we will always chain the new word to the front of the linked list. The time to insert a word is independent of the total number of input words n.
        - Hence, the average run time to insert a word into a table is O(1).
- Function: read_file
  This is a function for command "READ", it returns a boolean value and insert words from a text file into the dictionary.
- Function: tokenize
  This is a function for command "TOKENIZE", it returns the corresponding token of the input word.
    - Run time analysis:
        - The runtime for calling the hash function, and checking if hash value is valid is O(1).
        - we already knew where the token will be in the hash table by calculating its hash value. Assume uniform hashing, if there is collision, it will never take n times to find the token stored in the linked list. The time to find the token is independent from the total number of words.
        - Hence the average run time to tokenize a word is O(1).
- Function: retrieve
  This is a function for command "RETRIEVE", it returns the corresponding word of the input token.
    - Run time analysis:
        - The runtime for calculating the h(t), and check if the input token is valid takes O(1) time.
        - We already knew where the word will be in the hash table by calculating the index hash function. Assume uniform hashing, if there is collision, it will never take n times to find the word stored in the linked list. The time to find the word is independent from the total number of words.
        - Hence the average run time to tokenize a word is O(1).
- Function: tokenize_string
  This is a function for command "STOK", it returns the corresponding string of token of the input string of words.
- Function: retrieve_string
  This is a function for command "TOKS", it returns the corresponding string of words of the input string of tokens.
- Function: print_keys_at
  This is a function for command "PRINT", it returns the key of total ascii number for the words stored in the input position of the hash table.