GitHub link: https://github.com/RazvanAndreiLazar/FLCD
* works for non-space cases (a=2+1)

**HashTable**

Representation:

- Array with 509 buckets
- Collision resolution
  closed addressing with dynamic array
- keys are strings

hash function - rolling hash function for strings

p = 51

hash = a[0] p^0 + a[1] p^1 + ... + a[N] p^N % M

search:

compute hash value

search element linearly in the corresponding bucket

- return **value** if the key exists, return **None** otherwise

insert:

compute hash value

search element linearly in the corresponding bucket

if key exists update the value

otherwise add a new element

**SymbolTable → HashTable**

index - static field, increasing on each insert, used as a value for insert

**LinkedNode:**

- value: value of the node
- next: LinkedNode / null

**LinkedList:**

- head: LinkedNode / null
- tail: LinkedNode / null

insert(value): inserts the value as a new tail

**PIF (Program Internal Form) → LinkedList:**

- value: pair(token, pair(token_id, id_from_symbol_table))

id_from_symbol_table is '-1' if the token is not a constant/identifier

each token is inserted in pif while parsing the program

**Scanner**

Fields

file - input file path of the program file

tokens_file - input file path with the programming language tokens

tokens - dictionary token, index with every token from the input file

Methods

set_file - set the file path for the input program

read_tokens - read the tokens from the input file

scan - scan the input program file line by line and parse every one

__parse_line – parse a line of the program

__parse_elem - parse an element word by word and if needed character by character in search for tokens, identifiers and constants

__is_identifier - check if the word is an identifier

__is_constant - check if the word is a constant

__add_to_pif – add the element to the PIF with the right ids

__separate_from_strings – returns the a list of the main program separated from the string constants, the constants will be on even positions, the main program on odd ones