https://github.com/taniasasaran/UBB-Computer-Science/tree/main/Semester-5/FLCD

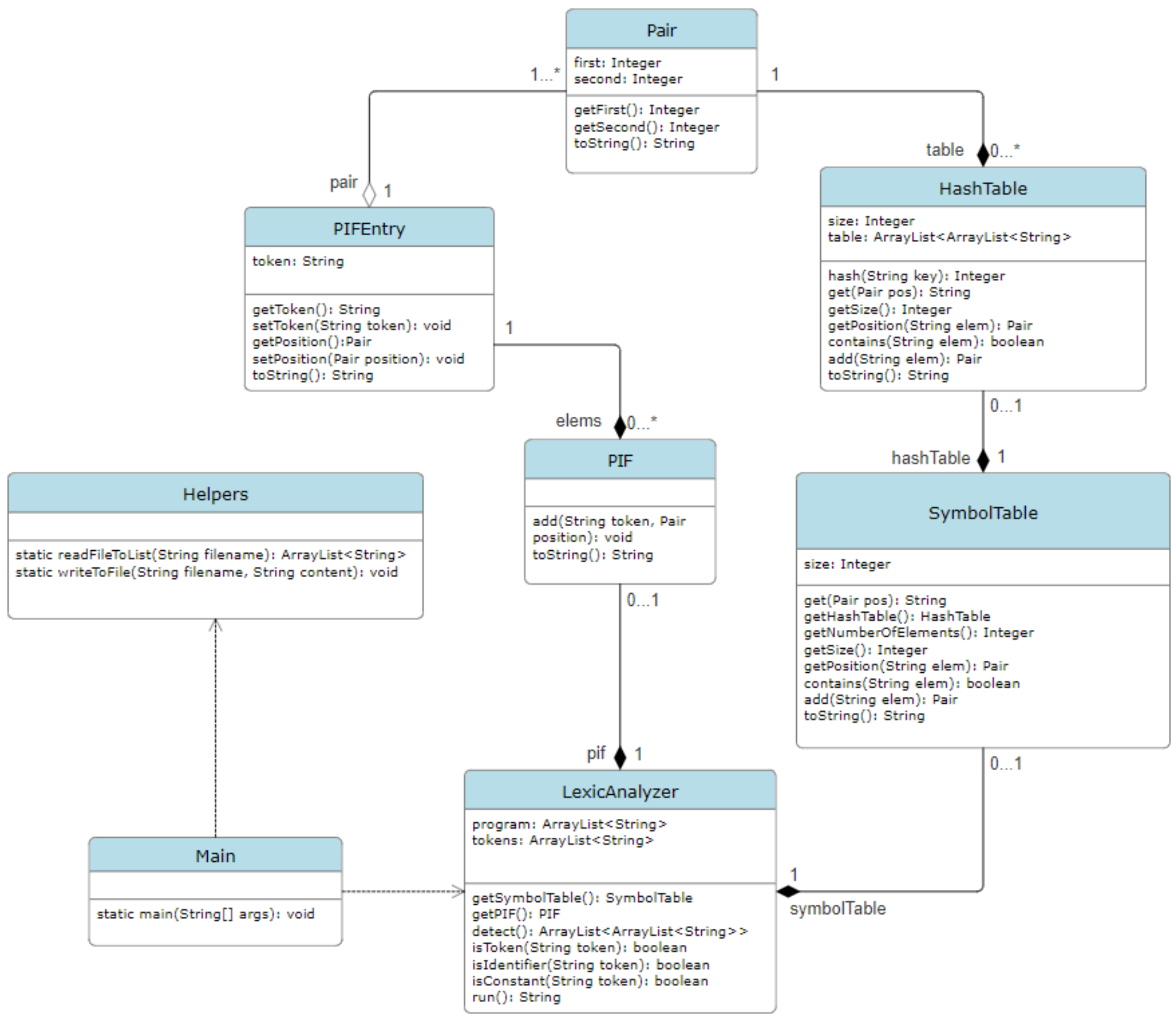
For my Symbol Table, I chose to implement one Hash Table that stores both constants and identifiers. The collision technique used for my hash table is separate chaining. The Hash Table uses two types of lists: one which stores all elements which have the same hash value and one which stores all such lists. I have also defined the Pair as a data structure, composed of 2 integer numbers, representing the pair of positions an element is on (hash value first, position in the hash list second).
getSize() method returns the size of the Hash Table
getNumberOfElements() method returns the number of elements in the table, regardless of its size.
getPosition(String elem) method returns the position of the element, which is computed the following way: we compute the hash for that element, we check whether the list from that is not empty, if it is empty we return null, otherwise we take the list from that position, parse it and when we reach our terminal, we create a Pair with the hash and terminal's index from the list.
get(Pair pos) method will return the terminal from the Symbol Table based on its position.
hash(String key) – represents the hash function and the approach used is to compute the sum of the characters modulo size.
contains (String elem) – returns true or false, depending if the element belongs to the Symbol Table or not
add(String elem) – adds an element to the Symbol Table and returns its position(when adding an existing element it doesn't add it again and it returns the position of the existing element).

PIF Table contains an array of PIFEntries. A PIFEntry is a tuple composed from a String, together with a Pair of integers(defined above), representing the position of that constant/identifier in the Symbol Table, otherwise (if it's a reserved work, operator or delimiter) it is bound to (-1, -1).
add(String token, Pair position) – creates and then adds an entry in the table.

The LexicalAnalyzer class is the one which contains the implementation of the scanning algorithm. It receives the list of tokens, a program(in the form of a list of strings, one string for each line of the file). It also has, as fields, a Symbol Table and a PIF Table which are initialized in the constructor.
detect() – this method detects the tokens in the program; it splits the line by space, retrieves an initial list of tokens; then it removes the comments, keeping the start comment symbol; after this, it uses a RegEx to allow the use of ' or , after one element (such as in lists, in declarations etc), adding both the element and the symbol in the final tokens list; the regex used is "^.*[',]$" which chooses sequences of characters that end with " ' " or " , "; finally, it returns a list of lists of tokens, a list of tokens for each line in the program;
isToken(String token) – checks if the input token is one of the delimiters, operators or reserved words, searching in the tokensList (from tokens.in) for a match; returns true if it is, false otherwise.
isIdentifier(String token) – checks if the input token is an identifier based on the language rules; it tries to match against a RegEx "^[A-Z_][A-Z_0-9]*$" that allows only words starting with "_" or capital letters and containing only "_", digits or capital letters
isConstant(String token) – checks if the input token is a constant, trying to match one of the 3 RegExes:
"^ 0|-?[1-9][0-9]*$" – to check if the token is a valid number (integer, with or without "-" sign)
"^\"[a-zA-Z0-9_]*\"$" – to check if the token is a valid string containing only the letters, digits and "_"
"^true|false$" – to check if the token is a valid bool.

run() – this method represents the implementation of the algorithm presented; if the token is a delimiter, operator or reserved word, it adds it to the PIF with the value (-1,-1), otherwise, it adds the meaning of it(“const” or “id”) to the PIF together with the position of it in the Symbol Table, after trying to add it there; if the token is neither of those 5 categories, returns a message, notifying that there occurred a lexical error on line N, token number M, otherwise it returns a message stating that the program is lexically correct.

Finally, the Helpers class is designed to keep functions that may help throughout the execution of the program (like reading from a file).
static readFileToList(String filename) – returns an ArrayList of String that contains each line of the file as a string.
static writeToFile(String filename, String content) – writes the content into a file.


Pair
first: Integer
second: Integer
getFirst(): Integer
getSecond(): Integer
toString(): String
1...*
1
table
0...*
pair
1
HashTable
size: Integer
table: ArrayList<ArrayList<String>
hash(String key): Integer
get(Pair pos): String
getSize(): Integer
getPosition(String elem): Pair
contains(String elem): boolean
add(String elem): Pair
toString(): String
PIFEntry
token: String
getToken(): String
setToken(String token): void
getPosition():Pair
setPosition(Pair position): void
toString(): String
1
elems
0...*
0...1
hashTable
1
PIF
add(String token, Pair position): void
toString(): String
Helpers
static readFileToList(String filename): ArrayList<String>
static writeToFile(String filename, String content): void
SymbolTable
size: Integer
get(Pair pos): String
getHashTable(): HashTable
getNumberOfElements(): Integer
getSize(): Integer
getPosition(String elem): Pair
contains(String elem): boolean
add(String elem): Pair
toString(): String
0...1
pif
1
0...1
LexicAnalyzer
program: ArrayList<String>
tokens: ArrayList<String>
Main
static main(String[] args): void
1
symbolTable
getSymbolTable(): SymbolTable
getPIF(): PIF
detect(): ArrayList<ArrayList<String>>
isToken(String token): boolean
isIdentifier(String token): boolean
isConstant(String token): boolean
run(): String