https://github.com/Gabarsolon/FLCD/tree/main/Lab2/src

The Symbol Table which I've implemented uses a hash table with separate chaining for collisions.

An element is inserted to a bucket using this hash function: h(k) = hashCode(k) % numberOfBuckets, where the numberOfBuckets Is a prime number. When the ratio between the number of elements and the number of buckets is greater than 0.75 (threshold) we changed the capacity to the next prime number and rehash every key.

The add function puts an entry to its corresponding bucket according to its key if its not already existing and then returns its associated value.

The get function returns the value associated to a key by finding its bucket and then the node from linked list.

(optional) The remove function deletes an entry from the hash table and assures that the links between the elements are consistent

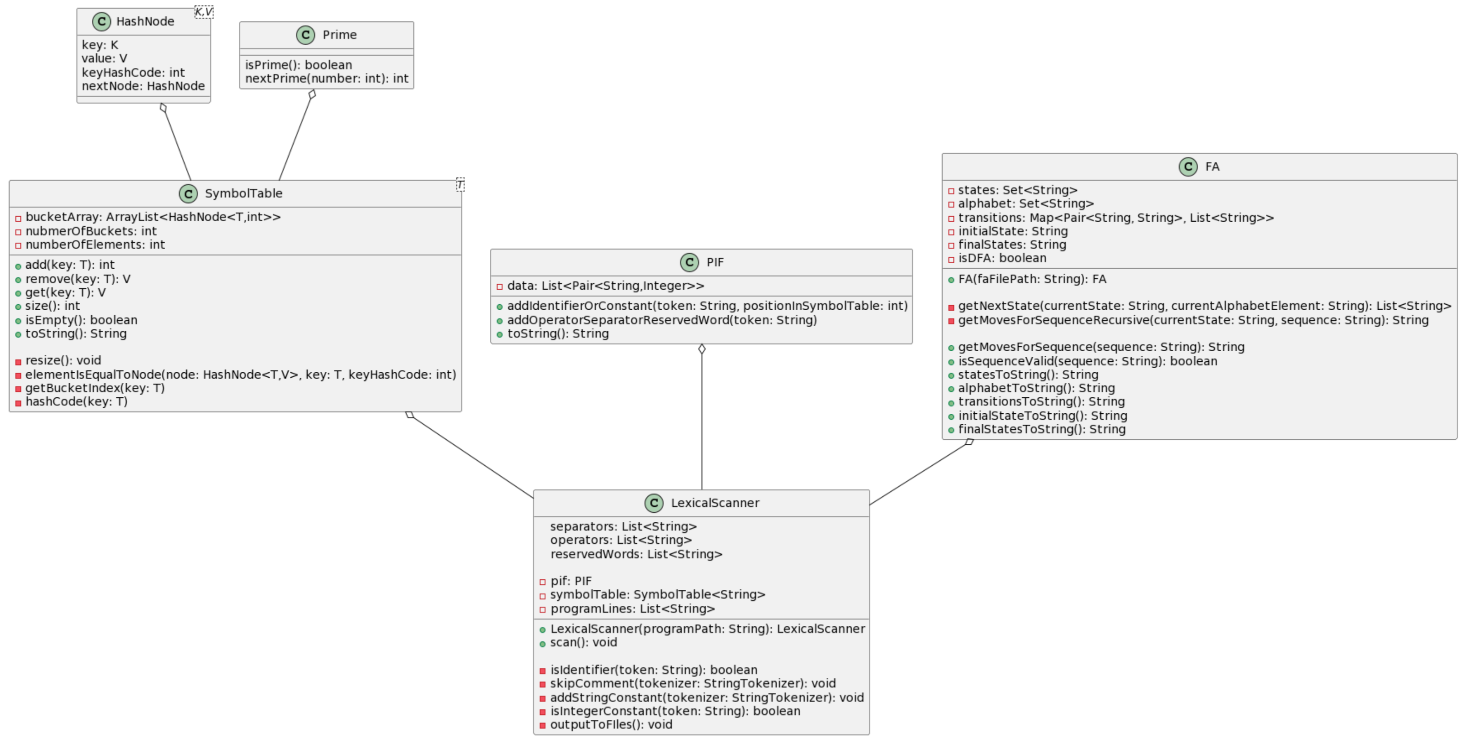
For the program internal form (PIF) class I used a list of (String, Integer) tuples which uses -1 as position for symbols that are operators, separators or reserved words

I'm using these regexes in my scanner class:

- for identifiers: ^[_a-zA-Z][_a-zA-Z0-9]*$
- for integer constants: ^0|(([+]|-)?[1-9][0-9]*)$

In order to skip comments I'm detecting the '#' character and then skip each token until the next '#'. I'm also doing something similar for string or char constants, but instead of ignoring the tokens, I'm adding them to a string that's delimited by " or ' respectively.

Class diagram:



HashNode K,V
key: K
value: V
keyHashCode: int
nextNode: HashNode
Prime
isPrime(): boolean
nextPrime(number: int): int
SymbolTable T
bucketArray: ArrayList<HashNode<T,int>>
nubmerOfBuckets: int
numberOfElements: int
add(key: T): int
remove(key: T): V
get(key: T): V
size(): int
isEmpty(): boolean
toString(): String
resize(): void
elementIsEqualToNode(node: HashNode<T,V>, key: T, keyHashCode: int)
getBucketIndex(key: T)
hashCode(key: T)
PIF
data: List<Pair<String,Integer>>
addIdentifierOrConstant(token: String, positionInSymbolTable: int)
addOperatorSeparatorReservedWord(token: String)
toString(): String
FA
states: Set<String>
alphabet: Set<String>
transitions: Map<Pair<String, String>, List<String>>
initialState: String
finalStates: String
isDFA: boolean
FA(faFilePath: String): FA
getNextState(currentState: String, currentAlphabetElement: String): List<String>
getMovesForSequenceRecursive(currentState: String, sequence: String): String
getMovesForSequence(sequence: String): String
isSequenceValid(sequence: String): boolean
statesToString(): String
alphabetToString(): String
transitionsToString(): String
initialStateToString(): String
finalStatesToString(): String
LexicalScanner
separators: List<String>
operators: List<String>
reservedWords: List<String>
pif: PIF
symbolTable: SymbolTable<String>
programLines: List<String>
LexicalScanner(programPath: String): LexicalScanner
scan(): void
isIdentifier(token: String): boolean
skipComment(tokenizer: StringTokenizer): void
addStringConstant(tokenizer: StringTokenizer): void
isIntegerConstant(token: String): boolean
outputToFIles(): void

For FA, I’m reading the 5-tuple M from a file, where on the first line is the set of states, on the next is alphabet and after that, on each line, until reaching the text “transitions_set_stop”, there are the transitions tuples consisting of state, terminal and next state. In the end, there are 2 lines reserved for initial state and the list of final states.

For storing transitions, I’m using a map where the key is a tuple containing the state and terminal character and the value can be a string or a list of strings representing the states. For the rest, I’m just using a set of strings for each one (except for the initial state, which is only a string).

For printing each element from M, I created 5 different toString function inside the FA class

In order to check validity for a sequence, first I’m verifying if the FA is DFA and then I’m parsing each character from the sequence and I’m checking if I have a pair with the current state inside the transitions map.

In order to use the FA class for integer constants or identifiers, I’m calling the validity check function recursively and I’m going through the set of all possible transitions for each (state, terminal) tuple (since in this case we have NFAs).

This is the BNF for FA.in:

<newline> ::= “\n”

<letter> ::= a|b|c|…|z|A|B|…|Z

<digit> ::= 0|1|2|…|9

<symbol> ::= <letter>|<digit>
<fa_in> ::= <states><alphabet><transitions><initial_state><states>

<states> ::= <symbol><newline>|<symbol>”,”<states>

<alphabet> ::= <symbol><newline>|<symbol>”,”<alphabet>

<transition> ::= <symbol>”,”<symbol>”,”<symbol>

<transitions> ::= <transition><newline>“transitions_set_stop”<newline>|<transition><newline><transitions>

<initial_state> ::= <symbol><newline>