https://github.com/AliceHincu/Formal-Languages-and-Compiler-Design

Implementation - HashTable

- The HashTable contains an ArrayList of HashNodes and implements the separate chaining collision resolution strategy. Each HashNode inside this list represents the head of another array, array in which the elements have the same hashcode as the head.

- The HashNode represents an element in the HashTable, having the key which represents the identifier/constant, the value which represent the position from the ST, and the reference to the next HashNode.

- As the function to get the hashcode, I used modular hashing: h(k) = k%capacity, where k is the sum of all ASCII codes of each composing character from the value.

- Supported operations:

- put(key, value): Maps the specified key to the specified value in this hashtable. Returns the old value if the key already exists, else returns the new value.
    * key - the identifier / constant
    * value - the position in the Symbol Table

- get(key): Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
    * key - the identifier / constant

-hashCode(Object key): Returns the hashcode
    * key - the value that will be transformed into a hashcode

Implementation - SymbolTable

- The SymbolTable contains a HashTable and the current available position. The key represents the SYMBOL, and the value is the POSITION.

- Supported operations:

- add(key): Adds the key to the symbol table. If the key is new, adds it to a new position and increments the new position with 1 unit. Returns the position of the key in the SymbolTable.
    * key - the identifier / constant

- get(key): Returns the position in the symbol table.
    * key - the identifier / constant

Implementation - ProgramInternalForm

- The ProgramInternalForm contains a List of Pairs, representing the TOKEN and the POSITION of the token in the Symbol Table.
- Pair is an implemented class with the purpose of saving in the list and returning from a function a set of 2 values.
- Supported operations:

- add(token, position): Adds the token and its position from the symbol table in the list.

* token - a token from the program. In the case of identifiers, it is "id" and the position from ST. For constants, it is "const" and the position from ST. In the case of keywords/operators/separators, space is ignored, the token will be the token itself, and the position will be -1.

* position - the position in the ST.

Implementation - Scanner

- The Scanner contains the Symbol Table, the Program Internal Form, and 3 lists that contain: the keywords, the operators and the separators. For identification of identifiers and constants, I used REGEX.
- Supported operations:

- scan(filename): Scans the given file which represents the program, and applies the Scanning Algorithm to it. If an exception message needs to be shown, the contents of the PIF.out and ST.out will be deleted. If the program is lexically correct, then write the two tables in their respective files.

-tokenize(line): For a given line from a file, get the tokens from it and return them.