**Scanner**

**Goian Tudor – 933/2**

<https://github.com/VaruTudor/Formal-Languages-and-Compiler-Design>

Requirement - Implement the Symbol Table (ST) as the specified data structure, with the corresponding operations

Implementation

* I decided to use a Hash Table as the data structure for the Symbol Table. The main advantage of hash tables over other data structures is speed . The access time of an element is on average O(1), therefore lookup could be performed very fast.
* For each entry in the ST a [Deque](https://docs.python.org/3/library/collections.html#collections.deque) (which under the hood is a doubly-linked list) will be used. It ensures that there will be no conflicts – when elements hash to the same position, they will be part of the same deque.
* I use modular hashing (the hash function is simply h(k) = k mod m for some m - size. The value k is an integer hash code generated from the key. If m is a) where for strings I add the ASCII ([American Standard Code for Information Interchange](https://www.asciitable.com/)) codes of each composing character.
* Supported operations
* add(key)

input: key – the token (string)

output: getPosition(key)

* remove(key)

input: key - the token (string)

output: -

* contains(key)

input: key - the token (string)

output: true/false

* getPosition(key)

input: key - the token (string)

output: (listPosition, listIndex) – listPosition is the position in the outer list and listIndex is the position in the current deque

Requirement - Implement a scanner (lexical analyzer): Implement the scanning algorithm and use ST from lab 2 for the symbol table.

Implementation

* Supported operations
* getStringToken(line, index)

input: line – a line read from the file (string), index – the current position

output: (token(string), index(int))

* isPartOfOperator(char)

input: char – a character (string)

output: true/false

* getOperatorToken(line, index)

input: line – a line read from the file (string), index – the current position

output: (token(string), index(int))

* tokenize(line)

input: line – a line read from the file (string)

output: tokens(list(string)) – the tokens identified in the given line

* Tokenizing

For a line, character by character, check if the current one is an operator, separator, string or belongs in the ST (meaning it is constant or identifier) .

* Scanning

Line by line performs tokenizing, adds each identifier or constant to the ST and the keywords, separators and operators /w (-1,-1). For the constants the code will be “const” and for identifiers “id”



