https://github.com/fiamardar/FLCD -> GitHub Repository Link

Lab 3

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

Input: Programs p1.txt/p2.txt/p3.txt/p1err.txt and token.txt (see Lab 1a)

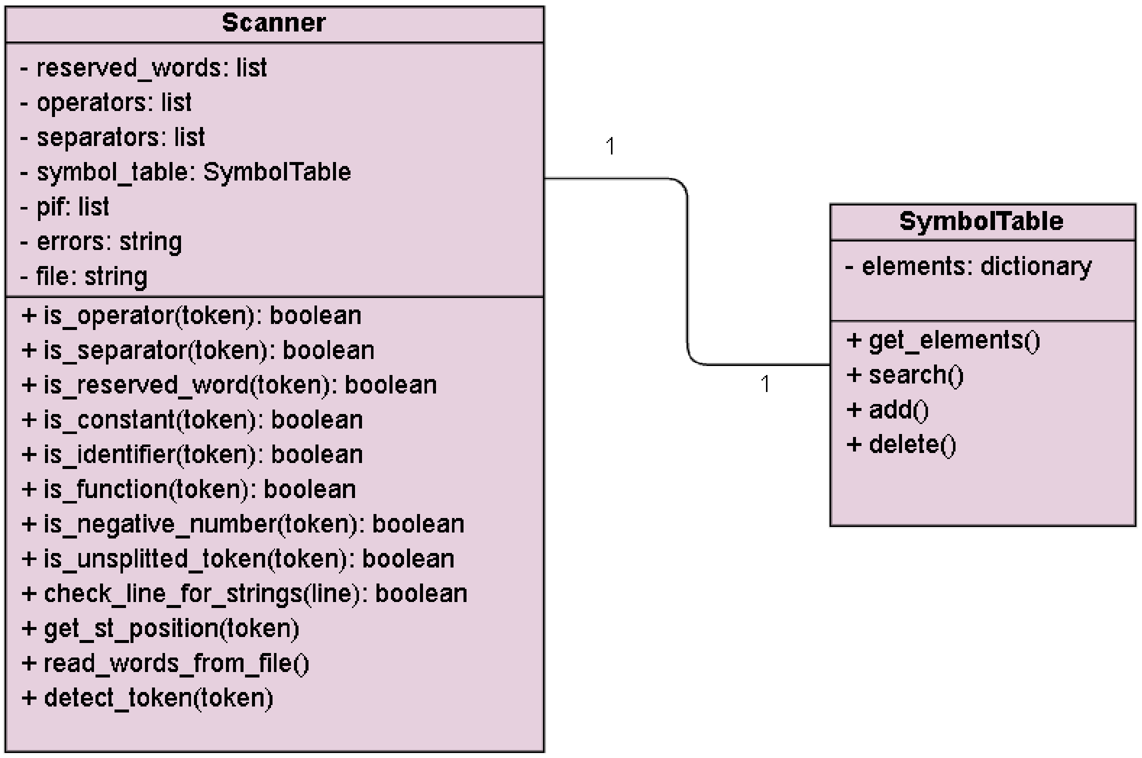
Output: PIF.out, ST.out, message "lexically correct" or "lexical error + location" for each program

Deliverables: input, output, source code, documentation

The Scanner class has an instance of Symbol Table as an attribute.

I chose to implement the Symbol Table class as a **Hash Table**.

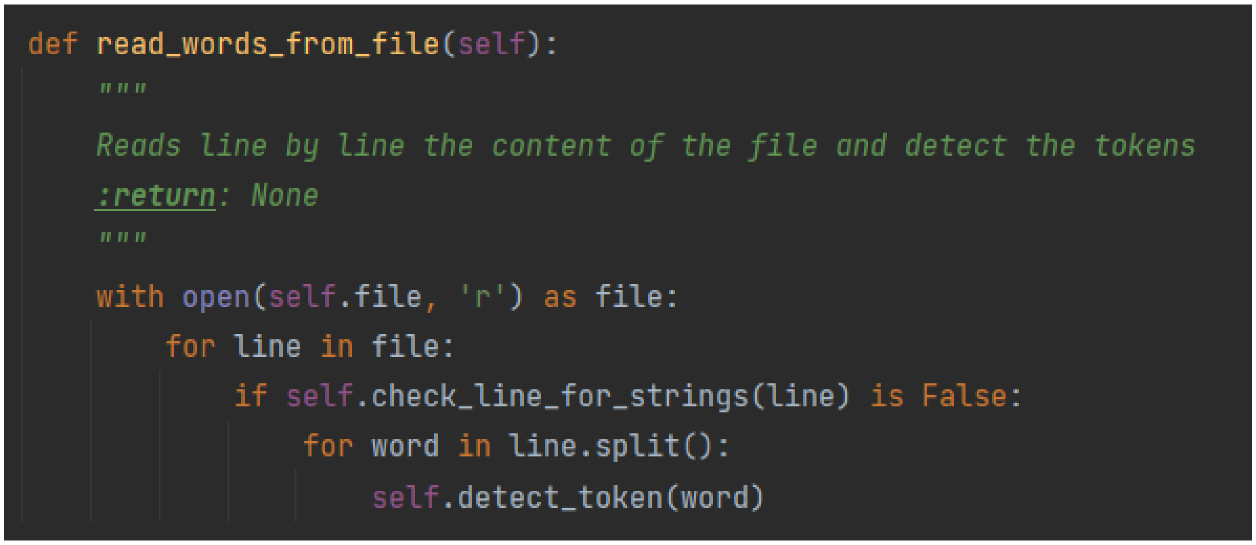
CLASS DIAGRAM


Scanner
- reserved_words: list
- operators: list
- separators: list
- symbol_table: SymbolTable
- pif: list
- errors: string
- file: string
+ is_operator(token): boolean
+ is_separator(token): boolean
+ is_reserved_word(token): boolean
+ is_constant(token): boolean
+ is_identifier(token): boolean
+ is_function(token): boolean
+ is_negative_number(token): boolean
+ is_unsplitted_token(token): boolean
+ check_line_for_strings(line): boolean
+ get_st_position(token)
+ read_words_from_file()
+ detect_token(token)
1
1
SymbolTable
- elements: dictionary
+ get_elements()
+ search()
+ add()
+ delete()

Scanner class has a function which reads the program from a file and parses it line by line.

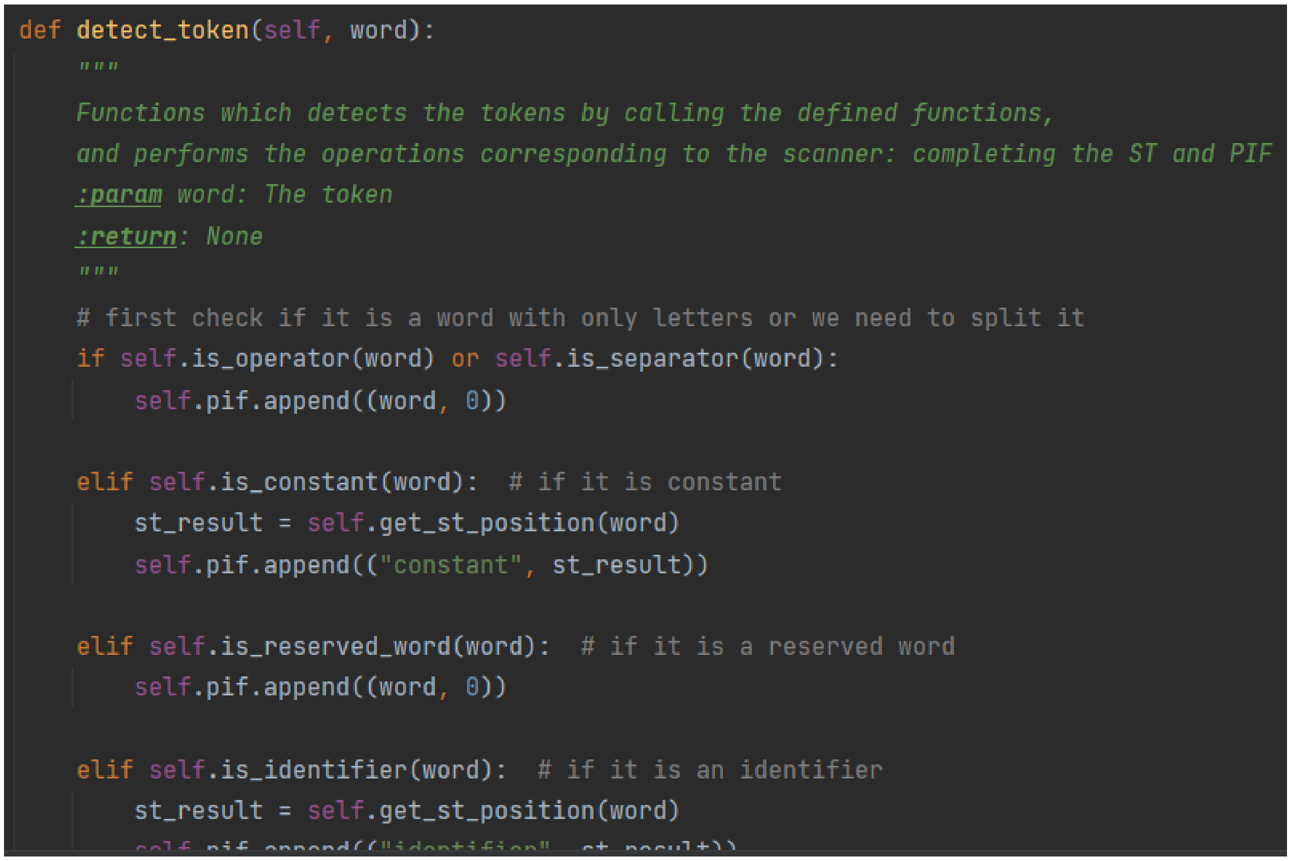
Each line it splitted in tokens and for each token we call method *detect_token()*.

We have a special case when the line read from the file contains a string, and in that case, we will call another function which will perform a special type of parsing tokens.



```
def read_words_from_file(self):
    """
    Reads line by line the content of the file and detect the tokens
    :return: None
    """
    with open(self.file, 'r') as file:
        for line in file:
            if self.check_line_for_strings(line) is False:
                for word in line.split():
                    self.detect_token(word)
```

Detect_token() method receives the token we want to classify and it goes through all the additional methods which checks the nature of the token using **regex**.



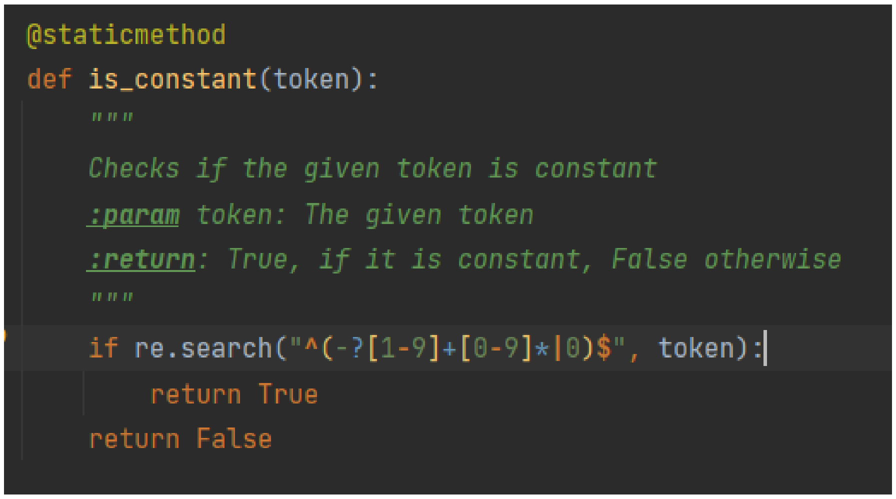
```
def detect_token(self, word):
    """
    Functions which detects the tokens by calling the defined functions,
    and performs the operations corresponding to the scanner: completing the ST and PIF
    :param word: The token
    :return: None
    """
    # first check if it is a word with only letters or we need to split it
    if self.is_operator(word) or self.is_separator(word):
        self.pif.append((word, 0))

    elif self.is_constant(word):  # if it is constant
        st_result = self.get_st_position(word)
        self.pif.append(("constant", st_result))

    elif self.is_reserved_word(word):  # if it is a reserved word
        self.pif.append((word, 0))

    elif self.is_identifier(word):  # if it is an identifier
        st_result = self.get_st_position(word)
        self.pif.append(("identifier", st_result))
```

An example of the additional method is the one which checks if a token is a constant:

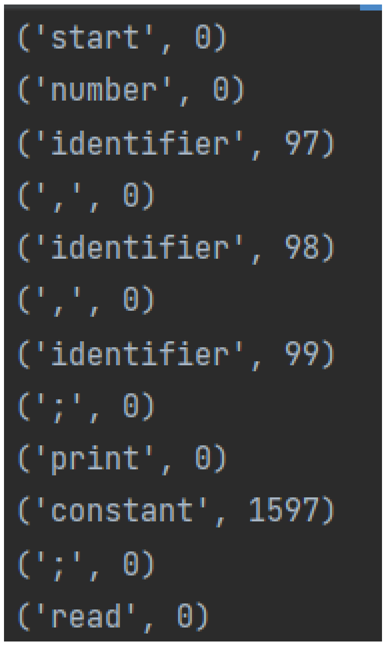


```
@staticmethod
def is_constant(token):
    """
    Checks if the given token is constant
    :param token: The given token
    :return: True, if it is constant, False otherwise
    """
    if re.search("^(-?[1-9]+[0-9]*|0)$", token):
        return True
    return False
```

It searches for a sequence which begins with – or a digit from 1 to 9, followed by any number of digits from 0 to 9 OR it could be only 0. In case this sequence is found, we return True, else return False.

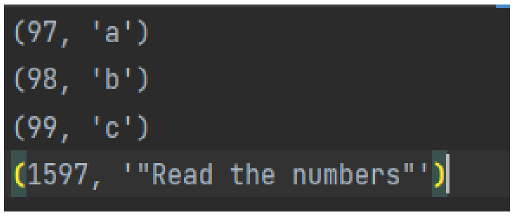
At the end of the execution, the program will write in 3 different files:

- PIF.out will contain the content of the PIF, each element on a different line
- ST.out will contain the content of the SymbolTable, each key-value pair written on a different line
- message.out will contain the message "Lexically correct" if we didn't find any errors, or the list of errors and the location otherwise



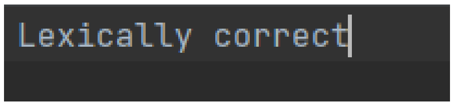
```
('start', 0)
('number', 0)
('identifier', 97)
(',', 0)
('identifier', 98)
(',', 0)
('identifier', 99)
(';', 0)
('print', 0)
('constant', 1597)
(';', 0)
('read', 0)
```

PIF.out example



```
(97, 'a')
(98, 'b')
(99, 'c')
(1597, '"Read the numbers"')
```

ST.out example



```
Lexically correct
```

message.out example