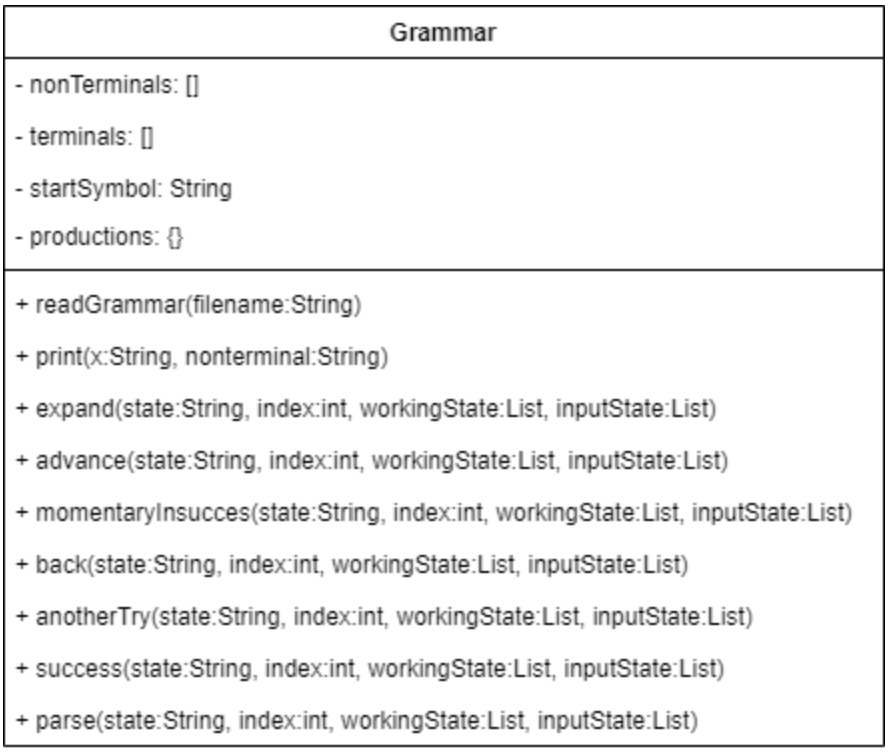
Lab 5 documentation

Link GitHub: https://github.com/timoteicopaciu/LFCD/tree/main/Lab%2005




```
class Grammar:

def readGrammar(self, filename):
    """
    Read a Grammar from a file
    :param filename: string, the name of the name where Grammar is stored
    :preconditions: filename must to be a string, representing a file name
    :postconditions: the Grammar object's attributes will be completed
    :return: none
    """
def print(self, x, nonterminal = None):
    """
    Print some attributes of Grammar
    :param x: char, representing an option in order to know what to return
    :param nonterminal: string, a nonterminal, production starting from it
will be printed, is optional
    :preconditions: x must be a string, x is from A = {'1', '2', '3', '4'}
    :postconditions: a string is returned, representing an attribute as a
string, or '' if x is not in A
    :return: a string
    """
def expand(self, state, index, workingStack, inputStack):
    """
    Expand function
    :param state: a char
        :pre: 'q'
        :post: 'q'
```

```
    :param index: integer
        :pre: some integer i
        :post: same i
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: some list W
        :post: list W U A_0, where A_0 is a name for first prod of non-
terminal A
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: A U I, where A is a non-terminal and I is a list
        :post: first prod of A U I
    :return: a configuration, all input params but updated
    """
def advance(self, state, index, workingStack, inputStack):
    """
    Advance function
    :param state: a char
        :pre: 'q'
        :post: 'q'
    :param index: integer
        :pre: some integer i
        :post: i + 1
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: some list W
        :post: list W U a, where a is the first terminal from the input stack
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: a U I, where a is a terminal, a terminal = current symbol from
input, and I is a list
        :post: I
    :return: a configuration, all input params but updated
    """
def momentaryInsuccess(self, state, index, workingStack, inputStack):
    """
    MomentaryInsuccess function
    :param state: a char
        :pre: 'q'
        :post: 'b' back/previous state
    :param index: integer
        :pre: some integer i
        :post: same i
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: some list W
        :post: same W
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: a U I, where a is a terminal and I is a list, a is different
from current symbol from input
        :post: same a U I
    :return: a configuration, all input params but updated
    """
def back(self, state, index, workingStack, inputStack):
    """
   Back function
```

```
    :param state: a char
        :pre: 'b'
        :post: 'b'
    :param index: integer
        :pre: some integer i
        :post: i - 1
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: some list W U a, where a is a terminal
        :post: list W, without terminal a
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: I
        :post: a U I, where a is a terminal and I is a list
    :return: a configuration, all input params but updated
    """
def anotherTry(self, state, index, workingStack, inputStack):
    """
    AntoherTry function
    :param state: a char
        :pre: 'b'
        :post: 'q' or 'b' or 'e'
    :param index: integer
        :pre: some integer i
        :post: same i
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: list W U A_j, where A_j is a name for the j-th prod of non-
terminal A
        :post: list W U A_j+1 or just W
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: j-th prod of A U I
        :post: j+1-th prod of A U I or A or just I
    :return: a configuration, all input params but updated
    """

def success(self, state, index, workingStack, inputStack):
    """
    Success function
    :param state: a char
        :pre: 'q'
        :post: 'f'
    :param index: integer
        :pre: some integer (n + 1)
        :post: same integer (n + 1)
    :param workingStack: a list, representing the working stack, stores the
way the parse is built
        :pre: some list W
        :post: same list W
    :param inputStack: a list, representing  input stack, part of the tree to
be built
        :pre: empty list
        :post: empty list
    :return: a configuration, all input params but updated
    """
```

```
def parse(self, state, index, workingStack, inputStack):
    """
     This function parse the word set in self.__word item and check if is a
accepted sequence using recursion
     :param state: a char
         :pre: 'q'
         :post: any state
     :param index: integer
         :pre: 0
         :post: some integer
     :param workingStack: a list, representing the working stack, stores the
way the parse is built
         :pre: []
         :post: same list W
     :param inputStack: a list, representing  input stack, part of the tree
to be built
         :pre: a list containing just starting symbol
         :post: empty list
     :return: a configuration, all input params but updated
     """
```