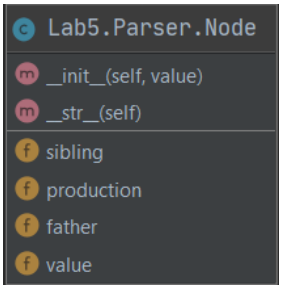
Lab 7

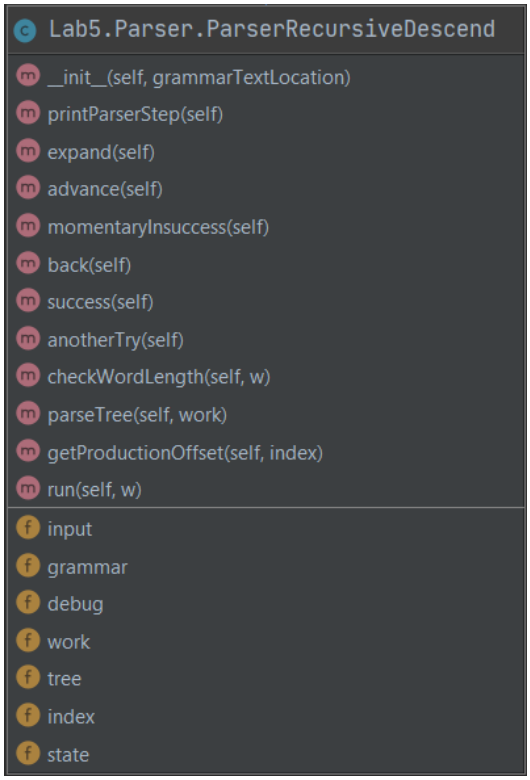
Recursive descent parser – implemented in team with Andrei Onița

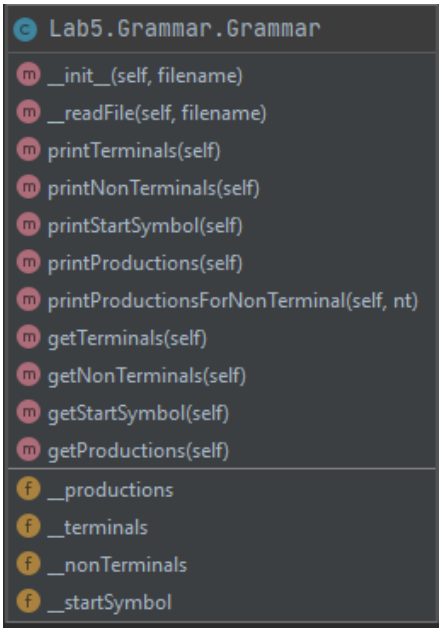
Github link: https://github.com/the-coding-cloud/FLCD/tree/main/Lab5

- Finished implementing the parsing strategy + bug fixes
- Implemented computation of the parse tree (parse tree is an arbitrary tree using the father sibling representation in a table)

UML Diagrams







Grammar

Data Structure

We created the class Grammar which stores the grammar from a text file

Constructor(file)

- Pre:
- In: file: string - location of Grammar definition
- Out: Grammar object
- Post: creates a Grammar object initialised with the terminals, nonTerminals, productions, startSymbol, File

The *terminals, nonTerminals, productions* fields are lists.

Operations:

Function readFile(filename)

- Pre: an existing fileName path
- In: fileName : String
- Out: -
- Post: the *terminals, nonTerminals, productions, startSymbol* fields are read from the file and initialised
- Desc: reads the Grammar and loads it in the memory

Parser - Recursive Descent

Data Structure

We created the class ParserRecursiveDescent

Constructor(file)

- Pre: file - location of Grammar definition
- In: file - string
- Out: a Parser object
- Post: creates a Parser object that initialises with the work and input stacks, index, state

Operations:

Function expand()

- Pre:
- In:
- Out: -
- Post:
- Desc: expands the non-terminal into its first production of terminals

Function advance()

- Pre:
- In:
- Out: -
- Post:
- Desc: puts one terminal in the work stack

Function momentaryInsuccess()

- Pre:
- In:
- Out: -
- Post:
- Desc: state is change to b

Function back()

- Pre:
- In:
- Out: -
- Post:
- Desc: goes back one index

Function success()

- Pre:
- In:
- Out: -
- Post:
- Desc: state set to s

Function anotherTry()

- Pre:
- In:
- Out: -
- Post:
- Desc: parses the last non terminal to the next set of terminals in its production, or pops the non terminal from work stack to input stack

Function checkWordLength()

- Pre:
- In: word to be checked
- Out: True or False
- Post:
- Desc: checks if the word length wasn't exceeded by the index and then checks the first letter in the stack with its word counterpart.

Function parseTree()

- Pre: work is the work stack
- In: work – list

- Out: prints tree
- Post: the table respecting the father sibling representation of the parse tree is printed
- Desc: the table respecting the father sibling representation of the parse tree is printed

Function getProductionOffset(index)

- Pre: index is a valid index from the work stack
- In: index - int
- Out: offset
- Post: offset - int
- Desc: returns an offset used in computing the index of a certain node in the parse tree represented using the table with father-sibling notation

Function run(w)

- Pre: w – sequence to parse
- In: w - list
- Out: True/False
- Post: function returns True and prints the parse tree if the sequence is accepted; returns False otherwise
- Desc: function returns True and prints the parse tree if the sequence is accepted; returns False otherwise

Node

Data Structure

We created the class Node, which stores the information of a node in the parse tree

Constructor(value)

- Pre:
- In: value - string
- Out: Node object
- Post: creates a Node object initialised with the value given in the constructor and the father, sibling and production fields are initialised with -1 (“invalid” value chosen to represent that no actual value had been assigned to them)

Code review – Lab 7

Istvan Olah & Razvan Neta – LR(0) Parser

- Algorithm works as expected
- The code looks quite neat, I like the specifications for each function – it makes it easier to understand what is happening at a quick glance
- The function names are suggestive
- I really like the nicely formatted output
- Some namings are slightly strange though – what is a “transaction” for example? (I am referring mainly to fields/variables that are not conventions from the lectures)
- The code is not split into more modules – the Grammar class is missing, it is integrated into the parser, even if the initial requirements were different

All in all, the algorithm can be followed by reading the code, the explanations are helpful, but the code was written in one big class and it could have helped to have more smaller classes, instead of a huge one.