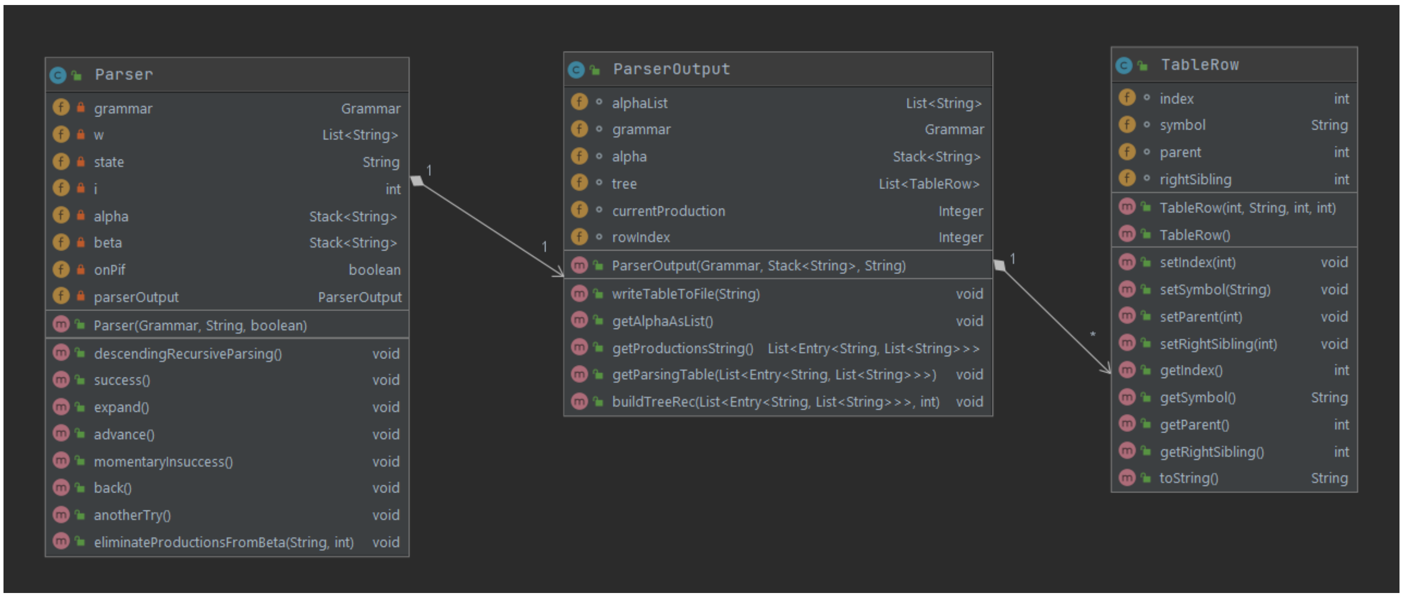
Requirement: Statement: Implement a parser algorithm (cont.) – Descendent Recursive Parser

PART 3: Deliverables

1. Algorithms corresponding to *parsing table* (if needed) and *parsing strategy*

2. Class *ParserOutput* - DS and operations corresponding to choice 2.a/2.b/2.c (Lab 5) (required operations: transform parsing tree into representation; print DS to screen and to file)

Analysis and Design:



The Parser Output is a class that generates the parser tree represented as a table (using father and sibling relation).

Its attributes are:

- alpha: Stack<String> – the working stack
- grammar: Grammar – the grammar
- alphaAsList: List<String> – the working stack represented as a list, with all the symbols in the right order
- tree: List<TableRow> – the table
- currentProduction: Int – the index of the current used production in the list of all the productions from the working stack
- rowIndex: Int – the index of the row that is currently created

Its methods are:

- writeTableToFile(filename: String) – writes the parsing tree in the file given as parameter
- getAlphaAsList() – transforms the working stack, from stack to a list
- getProductionsString(): List<Map.Entry<String, List<String>>
    - Post: a list with all the productions used in parsing, in the right order. A production is represented as a map entry, that maps a string (the lhs) to a list of strings (a list of all the symbols from the rhs)
- getParsingTree(usedProductions: List<Map.Entry<String, List<String>>) – constructs and displays the parsing tree represented as a table

- buildTreeRec(usedProductions: List<Map.Entry<String, List<String>>, parent: Int)
    - Pre: usedProductions – a list with all the productions used in parsing, in the right order; parent – the index in the parsing table of the parent of the elements that will be added in the current iteration
    - Post: adds to the table the rows corresponding to all the symbols from the current productions

The TableRow class represents a row in the parsing tree represented as a table.

Its attributes are:

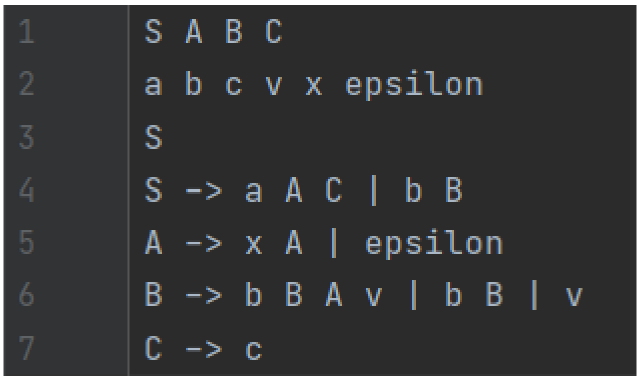
- index: Int – the index of the row in the table (the id)
- symbol: String – the symbol in the row
- parent: Int – the index of the parent in the table
- rightSibling: Int – the index of the rightSibling in the table

Implementation:

https://github.com/LaviniaGalan/FLCD/tree/master/Lab7

Testing:

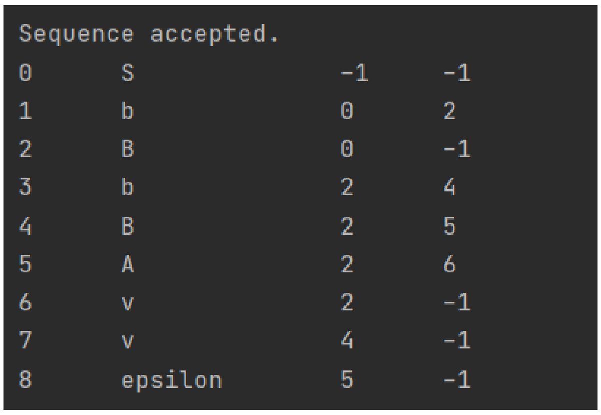
Grammar =



```
S A B C
a b c v x epsilon
S
S -> a A C | b B
A -> x A | epsilon
B -> b B A v | b B | v
C -> c
```

Input sequence = “b b v v”

Result =



```
Sequence accepted.
0      S              -1     -1
1      b              0      2
2      B              0      -1
3      b              2      4
4      B              2      5
5      A              2      6
6      v              2      -1
7      v              4      -1
8      epsilon        5      -1
```