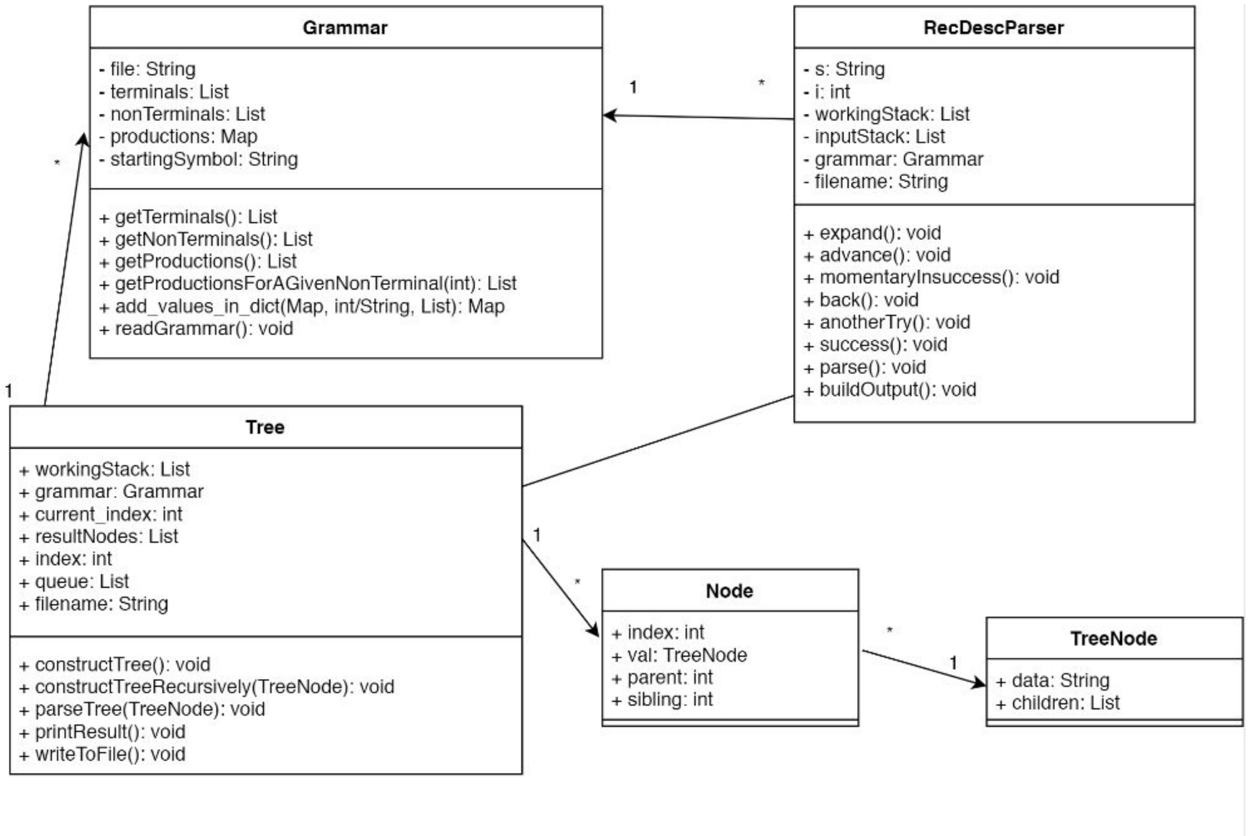
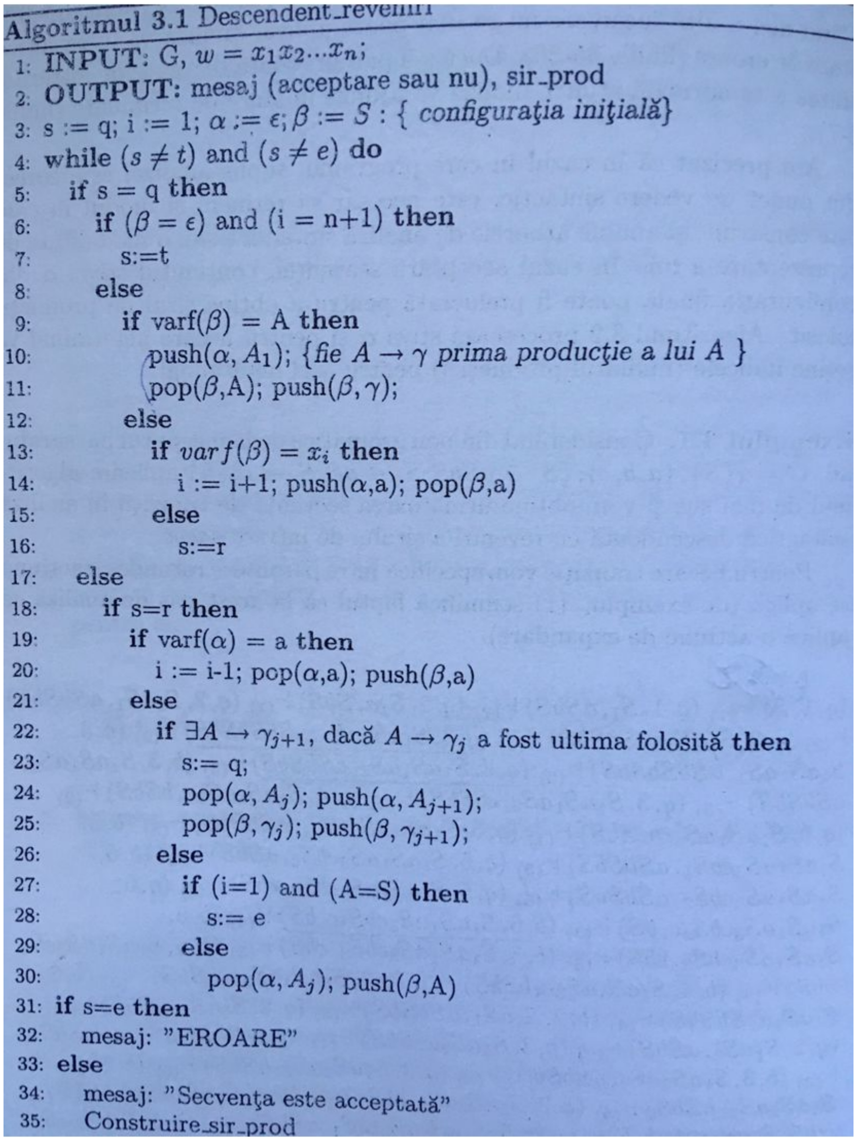
937/1 Tiutiu Natan, Ternovan Darius Lab 6
https://github.com/Natan-Gabriel/FLCD/tree/master/ParserAlgorithm

The purpose of this project was to create a parser algorithm using the recursive descendent method. The representation of the parsing tree (output) is in the form of a table using the father and sibling relation. Below you will find the class diagram:



The **Grammar** class reads and stores, as its name suggests, the grammar we want to use.

Our Recursive Descendent Parser, represented in the class **RecDescParser**, is a simple implementation of the algorithm depicted below:



Algoritmul 3.1 Descendent_revenire

```
INPUT: G, $w = x_1x_2...x_n$;
OUTPUT: mesaj (acceptare sau nu), sir_prod
s := q; i := 1; $\alpha := \epsilon; \beta := S$ : { configuraţia iniţială}
while $(s \neq t)$ and $(s \neq e)$ do
  if s = q then
    if $(\beta = \epsilon)$ and (i = n+1) then
      s:=t
    else
      if varf($\beta$) = A then
        push($\alpha, A_1$); {fie $A \rightarrow \gamma$ prima producţie a lui A }
        pop($\beta$,A); push($\beta, \gamma$);
      else
        if $varf(\beta) = x_i$ then
          i := i+1; push($\alpha$,a); pop($\beta$,a)
        else
          s:=r
  else
    if s=r then
      if varf($\alpha$) = a then
        i := i-1; pop($\alpha$,a); push($\beta$,a)
      else
        if $\exists A \rightarrow \gamma_{j+1}$, dacă $A \rightarrow \gamma_j$ a fost ultima folosită then
          s:= q;
          pop($\alpha, A_j$); push($\alpha, A_{j+1}$);
          pop($\beta, \gamma_j$); push($\beta, \gamma_{j+1}$);
        else
          if (i=1) and (A=S) then
            s:= e
          else
            pop($\alpha, A_j$); push($\beta$,A)
if s=e then
  mesaj: "EROARE"
else
  mesaj: "Secvenţa este acceptată"
  Construire_sir_prod
```

In our implementation, each case is split into its own method, hence the many void methods. The only method that is not also depicted in the above picture is the **buildOutput()** method, which simply initializes a new **Tree** object, calls tree's **constructTree()** method, then calls the methods that print the table to the console and to the file.

If the sequence given to the parser is accepted, the **Tree** class constructs the output table using a BFS algorithm that also retains the indices of a node's parent and sibling, if any exist. As requested in the requirements, this class also features methods that print the result to a given file, not only to the console.

Also, the program is able to read from the PIF.out file as requested in lab 7.