<https://github.com/ComanacDragos/ToyLanguageCompiler>

**Statement**:**Implement a parser algorithm (cont.)**

**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](https://moodle.cs.ubbcluj.ro/mod/assign/view.php?id=2841)) (required operations: transform parsing tree into representation; print DS to screen and to file)

**Implementation**

The following classes are added:

Node  
 Long id = nextId++;  
 Symbol value;  
 List<Node> children;

public Node(Symbol value, List<Symbol> symbols) // creates recursively the tree

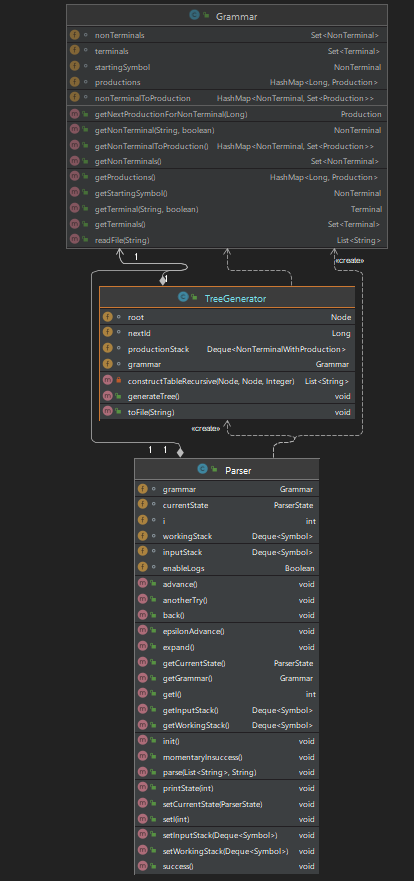
public class TreeGenerator   
 Node root; // root of the tree  
 Long nextId = 0L; // id generator for nodes  
 Deque<NonTerminalWithProduction> productionStack;  
 Grammar grammar;

public TreeGenerator(Deque<Symbol> workingStack, Grammar grammar) // instantiate the productionStack from a given workingStack so that it contains only the NonTerminalWithProduction classes

public void generateTree() // begins the generation of the tree

public void toFile(String outputDir) // writes to an output directory the table

private List<String> constructTableRecursive(Node currentNode, Node fatherNode, Integer positionRelativeToFather) // constructs the string representation of the table recursively



**Testing**

S A B  
a b  
S  
S ::= a B | b A  
A ::= a | a S | b A A  
B ::= epsilon | b | b S | a B B

Table

Description automatically generated with low confidence

program statement\_list statement simple\_statement compound\_statement simple\_type array\_type type expression binary\_operator unary\_operator declaration\_statement iostatement assignment\_statement if\_statement else\_branch while\_statement expression' expression\_simple  
id constant int char bool string float >> << while if else and or ! + - \* / % > < >= <= != == = ; [ ] { } ( ) , ^  
program  
program ::= statement\_list  
statement\_list ::= statement | statement statement\_list  
statement ::= simple\_statement | compound\_statement  
  
simple\_statement ::= assignment\_statement ; | iostatement ; | declaration\_statement ;  
  
compound\_statement ::= if\_statement | while\_statement  
  
simple\_type ::= bool | char | int | string | float  
  
array\_type ::= simple\_type [ constant ]  
  
type ::= simple\_type | array\_type  
  
  
expression\_simple ::= constant | id | id [ constant ] | id [ id ] | unary\_operator expression | ( expression )  
  
expression' ::= binary\_operator expression expression' | epsilon  
  
expression ::= expression\_simple expression'  
  
declaration\_statement ::= type id | type id = expression  
  
iostatement ::= << id | << id [ constant ] | << id [ id ] | >> expression  
  
assignment\_statement ::= id = expression  
  
if\_statement ::= if ( expression ) { statement\_list } else\_branch  
else\_branch ::= epsilon | else { statement\_list }  
  
while\_statement ::= while ( expression ) { statement\_list }  
  
unary\_operator ::= !  
binary\_operator ::= + | - | \* | / | ^ | % | and | or | > | < | >= | <=| != | ==

Graphical user interface

Description automatically generated with medium confidence