FLCD - LAB 7 - PARSER ALGORITHM
POP VLAD - PURCEL LOREDANA

https://github.com/Pop-Vlad/FLCD-Parser

Parsing Method: RECURSIVE DESCENDENT

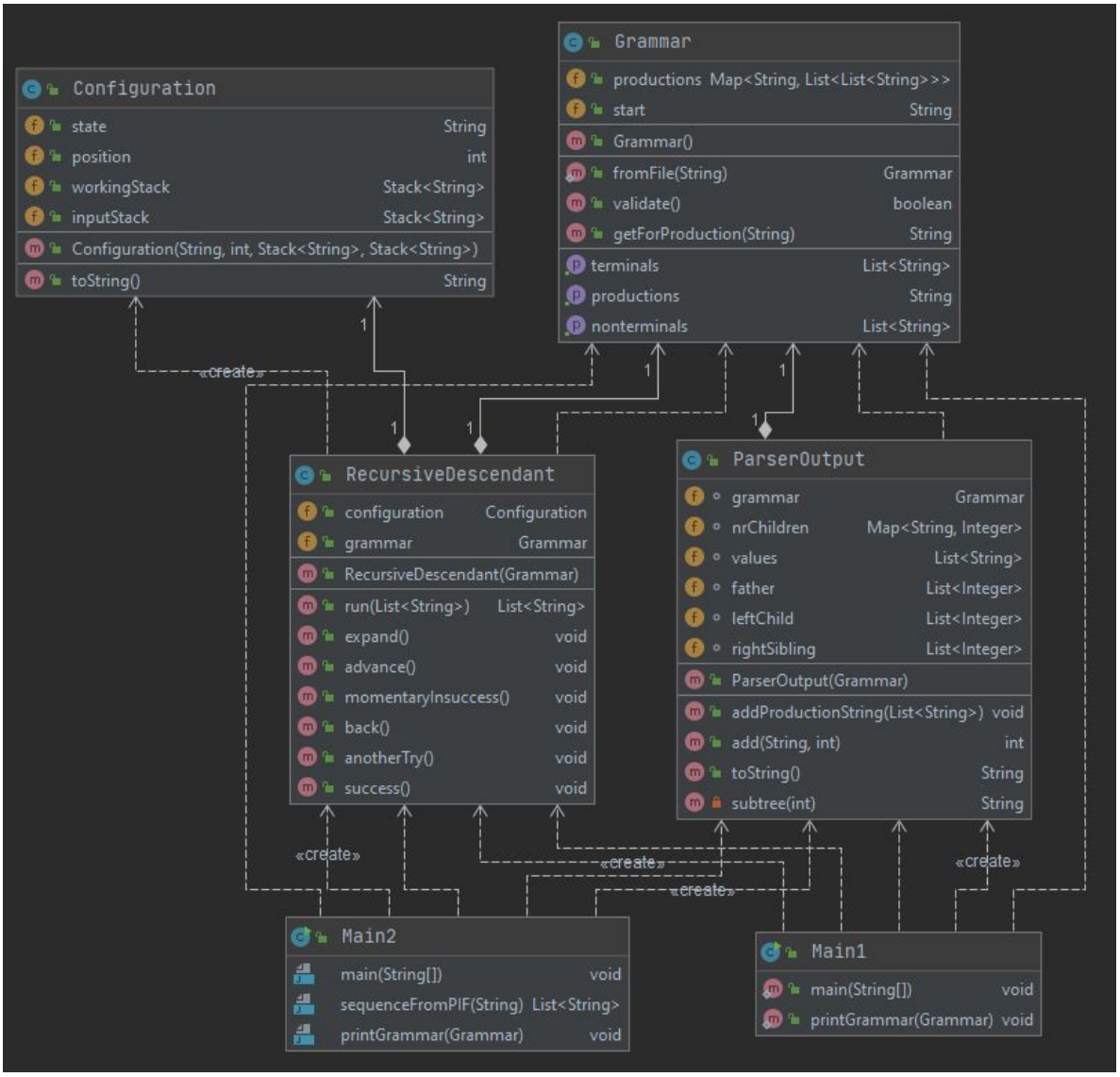
The representation of the parsing tree (output) will be: TABLE (using father and sibling relation)

1. Class grammar (required operations: read a grammar from file, print set of nonterminals, set of terminals, set of productions, production for a given nonterminal)
2. *Input*: g1.txt (grammar from seminar), seq.txt, g2.txt (grammar of the mini-language with syntax rules from Lab1), PIF.out (result of lab 3)
3. *Output*: out1.txt, out2.txt
4. Functions corresponding to parsing strategy
5. Functions corresponding to moves: **expand, advance, momentary insucces, back, another try, succes**
6. Algorithm corresponding to parsing tables (if needed) and parsing strategy
7. Class ParserOutput - DS and operations corresponding to table (lab 5) (required operations: transform parsing tree into representation; print DS to screen and to file)

Input File Format:

1. Nonterminals
2. Terminals
3. Starting symbol
4. 5. 6. ... Productions

Class Diagram:



Grammar class:

`public static Grammar fromFile(String fileName) :`

- Populates the “nonterminals” list, “terminals” list, productions and starting symbol from the file.

Configuration class:

Configuration: (s, i, α, β)
Where:

- s(state) can be:
  - q: normal state
  - b: back state
  - f: final state
  - e: error state
- i: position of current symbol in input sequence(w)

- **α**: working stack: stores the way the parser is built
- ***β***: input stack: part of the tree to be built

RecursiveDescendant class:

```
public List<String> run(List<String> w):
```

The function "run" stops when the state is either final or error so:
If the state is normal:

We EXPAND: when the head of the input stack is a nonterminal.

We ADVANCE: when the head of the input stack is a terminal AND this terminal is equal with the current symbol from input sequence.

Or we call MOMENTARY INSUCCESS: when the head of the input stack is a terminal AND this terminal is different from the current symbol from input sequence.

Otherwise if the state is back:

We call ANOTHER TRY: when the head of the input stack is a nonterminal.

Or we go BACK if the head of the input stack is a terminal.

Tests:

Input 1:

- Grammar (g1.txt):

  S
  a b c
  S
  S->a#S#b#S
  S->a#S|c
- Input sequence(seq.txt):

  "a", "a", "c", "b", "c"

Output1:

- Working stack(**α**): S#1, a, S#2, a, S#3, c, b, S#3, c
- Parser Output (out1.txt):

  Values: [epsilon, S#1, a, S#2, a, S#3, c, b, S#3, c]
  Father: [0, 1, 1, 3, 3, 5, 1, 1, 8]
  Left child: [1, 2, -1, 4, -1, 6, -1, -1, 9, -1]
  Right sibling: [-1, -1, 3, 7, 5, -1, -1, 8, -1, -1]

  epsilon
  \-- S#1
  |-- a
  |-- S#2
  | |-- a
  | \-- S#3
  | \-- c

```
|-- b
\-- S#3
  \-- c
```

Input 2:

- Grammar(g2.txt):

  program stmt_list stmt simple_stmt struct_stmt assign_stmt decl_stmt io_stmt identifier constant expression array_elem term factor arithmetic_operand decl_stmt type simple_type comp_type read_operand write_operand if_stmt while_stmt condition relation pif_symbol := - + = * / % ( ) [ ] [] read write if else while int char { } ; == < > <= >= != 0 1
  program
  program->stmt_list
  stmt_list->stmt#;|struct_stmt|stmt#;#stmt_list|struct_stmt#stmt_list
  stmt->simple_stmt|struct_stmt
  simple_stmt->assign_stmt|decl_stmt|io_stmt
  assign_stmt->identifier#:=#expression|array_elem#:=#expression
  expression->term#+#expression|term#-#expression|term
  term->factor#*#term|factor#/#term|factor#%#term|factor
  factor->arithmetic_operand|(#expression#)
  arithmetic_operand->identifier|constant|array_elem
  decl_stmt->type#identifier
  type->simple_type|comp_type
  simple_type->int|char
  comp_type->simple_type#[#constant#]
  io_stmt->read#(#read_operand#)|write#(#write_operand#)
  read_operand->identifier|array_elem
  write_operand->identifier|array_elem|constant
  struct_stmt->{#stmt_list#}|if_stmt|while_stmt
  if_stmt->if#(#condition#)#struct_stmt|if#(#condition#)#struct_stmt#else#stmt
  while_stmt->while#(#condition#)#struct_stmt
  condition->expression#relation#expression
  relation->==|<|>|<=|>=|!=
  identifier->pif_symbol
  array_elem->identifier#[#expression#]
  constant->pif_symbol
  pif_symbol->0|1

- Input sequence(PIF.out):

  (int,-1)
  (0,120)
  (;,-1)
  (int,-1)
  (0,121)
  (;,-1)
  (read,-1)

((,-1)
(0,120)
(),-1)
(;,-1)
(read,-1)
((,-1)
(0,121)
(),-1)
(;,-1)
(while,-1)
((,-1)
(0,120)
(!=,-1)
(0,121)
(),-1)
({,-1)
(if,-1)
((,-1)
(0,120)
(>,-1)
(0,121)
(),-1)
({,-1)
(0,120)
(:=,-1)
(0,120)
(-,-1)
(0,121)
(;,-1)
(},-1)
(else,-1)
({,-1)
(0,121)
(:=,-1)
(0,121)
(-,-1)
(0,120)
(;,-1)
(},-1)
(},-1)
(write,-1)
((,-1)
(0,120)
(),-1)
(;,-1)

Output2:

- Parser output (out2.txt):

Values: [epsilon, program#1, stmt_list#3, stmt#1, simple_stmt#2, decl_stmt#1, type#1, simple_type#1, int, identifier#1, 0, ;, stmt_list#3, stmt#1, simple_stmt#2, decl_stmt#1, type#1, simple_type#1, int, identifier#1, 0, ;, stmt_list#3, stmt#1, simple_stmt#3, io_stmt#1, read, (, read_operand#1, identifier#1, 0,), ;, stmt_list#3, stmt#1, simple_stmt#3, io_stmt#1, read, (, read_operand#1, identifier#1, 0,), ;, stmt_list#4, struct_stmt#3, while_stmt#1, while, (, condition#1, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, relation#6, !=, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0,), struct_stmt#1, {, stmt_list#2, struct_stmt#2, if_stmt#2, if, (, condition#1, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, relation#3, >, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0,), struct_stmt#1, {, stmt_list#1, stmt#1, simple_stmt#1, assign_stmt#1, identifier#1, 0, :=, expression#2, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, -, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, ;, }, else, stmt#2, struct_stmt#1, {, stmt_list#1, stmt#1, simple_stmt#1, assign_stmt#1, identifier#1, 0, :=, expression#2, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, -, expression#3, term#4, factor#1, arithmetic_operand#1, identifier#1, 0, ;, }, }, stmt_list#1, stmt#1, simple_stmt#3, io_stmt#2, write, (, write_operand#1, identifier#1, 0,), ;]

Father: [0, 1, 2, 3, 4, 5, 6, 7, 5, 9, 2, 2, 12, 13, 14, 15, 16, 17, 15, 19, 12, 12, 22, 23, 24, 25, 25, 25, 28, 29, 25, 22, 22, 33, 34, 35, 36, 36, 36, 39, 40, 36, 33, 33, 44, 45, 46, 46, 46, 49, 50, 51, 52, 53, 54, 49, 56, 49, 58, 59, 60, 61, 62, 46, 46, 65, 65, 67, 68, 69, 69, 69, 72, 73, 74, 75, 76, 77, 72, 79, 72, 81, 82, 83, 84, 85, 69, 69, 88, 88, 90, 91, 92, 93, 94, 93, 93, 97, 98, 99, 100, 101, 97, 97, 104, 105, 106, 107, 108, 90, 88, 69, 69, 113, 114, 114, 116, 117, 118, 119, 120, 119, 119, 123, 124, 125, 126, 127, 123, 123, 130, 131, 132, 133, 134, 116, 114, 65, 44, 139, 140, 141, 142, 142, 142, 145, 146, 142, 139]

Left child: [1, 2, 3, 4, 5, 6, 7, 8, -1, 10, -1, -1, 13, 14, 15, 16, 17, 18, -1, 20, -1, -1, 23, 24, 25, 26, -1, -1, 29, 30, -1, -1, -1, 34, 35, 36, 37, -1, -1, 40, 41, -1, -1, -1, 45, 46, 47, -1, -1, 50, 51, 52, 53, 54, 55, -1, 57, -1, 59, 60, 61, 62, 63, -1, -1, 66, -1, 68, 69, 70, -1, -1, 73, 74, 75, 76, 77, 78, -1, 80, -1, 82, 83, 84, 85, 86, -1, -1, 89, -1, 91, 92, 93, 94, 95, -1, -1, 98, 99, 100, 101, 102, -1, -1, 105, 106, 107, 108, 109, -1, -1, -1, -1, 114, 115, -1, 117, 118, 119, 120, 121, -1, -1, 124, 125, 126, 127, 128, -1, -1, 131, 132, 133, 134, 135, -1, -1, -1, -1, 140, 141, 142, 143, -1, -1, 146, 147, -1, -1, -1]

Right sibling: [-1, -1, -1, 11, -1, -1, 9, -1, -1, -1, -1, 12, -1, 21, -1, -1, 19, -1, -1, -1, -1, 22, -1, 32, -1, -1, 27, 28, 31, -1, -1, -1, 33, -1, 43, -1, -1, 38, 39, 42, -1, -1, -1, 44, -1, 139, -1, 48, 49, 64, 56, -1, -1, -1, -1, -1, 58, -1, -1, -1, -1, -1, -1, -1, 65, -1, 67, 138, -1, -1, 71, 72, 87, 79, -1, -1, -1, -1, -1, 81, -1, -1, -1, -1, -1, -1, -1, 88, 112, 90, 111, 110, -1, -1, 96, -1, 97, -1, 103, -1, -1, -1, -1, 104, -1, -1, -1, -1, -1, -1, -1, -1, 113, -1, -1, 116, 137, 136, -1, -1, 122, -1, 123, -1, 129, -1, -1, -1, -1, 130, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 149, -1, -1, 144, 145, 148, -1, -1, -1, -1]

... (tree representation is too long)