Parser

Moraru Dora-Maria + Mihalca Stefania

GitHub repository: https://github.com/stefa13/FLCD

Grammar has a field for each set of the grammar, namely: terminals, nonterminals, productions and a starting symbol. We also have the class Production which contains the left hand-side of a production (startingNonTerminal) and a set of lists which contains the right hand-side of the production (rules). Since we are implementing the LL(1) algorithm, we also implemented the first and follow algorithms.

The first algorithm builds a set for each non-terminal that contains all terminals from which we can start a sequence, starting from that given non-terminal.

The follow algorithm builds a set for each non-terminal returns the first of what is after, namely all the non-terminals into which we can proceed from the given non-terminal.

Having these 2 sets built for each non-terminal (and for terminals also), we proceed to build the LL(1) parse table. We follow the rules given at the lecture: we build a table that has as rows non-terminals + terminals, and as rows, all terminals, plus the $ sign in both rows and columns. After that, we populate the table with values obtained by the rules given at the lecture: (e.g.: M (a, a) = pop, M ($, $) =acc).

Having the parse table built, the next step is parsing a given input sequence with the LL(1) parsing algorithm, following the push/pop rules. This will build an output which we subsequently recursively build a tree – we start from the root, and then we take care of its first child and then right sibling. The last step is to iterate through the tree and print the obtained data.

Input example 1:

(G1.txt + sequence + output)

G1.txt:

```
S,A,B,C
(,),+,*,int,ε
S
S -> A B
A -> ( S ) | int C
B -> + S | ε
C -> * A | ε
```

Seq.txt

```
( int ) + int
```

Out.txt

```
1 | S | null | null
2 | A | 1 | null
3 | B | 1 | 2
4 | ( | 2 | null
5 | S | 2 | 4
6 | ) | 2 | 5
7 | A | 5 | null
8 | B | 5 | 7
9 | int | 7 | null
10 | C | 7 | 9
11 | ε | 10 | null
12 | ε | 8 | null
13 | + | 3 | null
14 | S | 3 | 13
15 | A | 14 | null
16 | B | 14 | 15
17 | int | 15 | null
18 | C | 15 | 17
19 | ε | 18 | null
20 | ε | 16 | null
```

Input example 2:

G2.txt

```
program,simple_type,type,declaration,statement_list,statement,compound_statement,simple_s
tatement,assignment_statement,io_statement,stmtTemp,tempIf,relation,condition,if_statemen
t,for_statement,while_statement
start,end,int,str,char,bool,arr,[,],{,},ε,>=,==,<=,!=,&&,if,elif,for,while,+,-
,*,/,(,),<,=,!,>,;,scan,print,integer_constant,identifier,else,string_constant,boolean_co
nstant,character_constant,constant
program
program -> start compound_statement end
simple_type -> int | str | char | bool
type -> simple_type
declaration -> type identifier
statement_list -> statement stmtTemp
```

```
stmtTemp -> ε | statement_list
statement -> simple_statement | if_statement | for_statement | while_statement
compound_statement -> { statement_list }
simple_statement -> assignment_statement ; | io_statement ; | declaration ;
assignment_statement -> identifier = identifier
io_statement -> scan ( identifier ) | print ( identifier )
if_statement -> if condition compound_statement tempIf
tempIf -> ε | else compound_statement
condition -> ( identifier relation identifier )
relation -> < | <= | == | != | >= | >
for_statement -> for ( assignment_statement ; condition ; assignment_statement )
compound_statement
while_statement -> while condition compound_statement
```

Pif.out

```
start { while ( identifier == identifier ) { bool identifier ; scan ( identifier ) ; }
int identifier ; for ( identifier = identifier ; ( identifier < identifier ) ; identifier
= identifier ) { char identifier ; scan ( identifier ) ; } print ( identifier ) ; if (
identifier > identifier ) { identifier = identifier ; } print ( identifier ) ; if (
identifier > identifier ) { print ( identifier ) ; } else { scan ( identifier ) ; } } end
```

Parsing output:

```
1, 15, 8, 14, 33, 25, 28, 15, 8, 11, 18, 7, 6, 5, 10, 8, 11, 17, 20, 9, 10, 8, 11, 18, 7,
6, 2, 10, 8, 13, 32, 19, 25, 26, 19, 15, 8, 11, 18, 7, 6, 4, 10, 8, 11, 17, 20, 9, 10, 8,
11, 17, 21, 10, 8, 12, 22, 25, 31, 15, 8, 11, 16, 19, 9, 23, 10, 8, 11, 17, 21, 10, 8,
12, 22, 25, 31, 15, 8, 11, 17, 21, 9, 24, 15, 8, 11, 17, 20, 9, 9
```