Exercise session 2
Grammars and attributions

2.1 Grammars

All grammars will use the ANTLR4 syntax.

EXERCISE #1 ► **Well-founded parenthesis**
Write a grammar and files to accept any text with well-formed parenthesis ')' and '['.

EXERCISE #2 ► **CSV**
A csv (for comma-separated values[1]) file:

- is plain text using a character set such as ASCII, various Unicode character sets (e.g. UTF-8), EBCDIC, or Shift JIS,
- consists of records (typically one record per line),
- with the records divided into fields separated by delimiters (typically a single reserved character such as comma, semicolon, or tab; sometimes the delimiter may include optional spaces),
- ...

Write a grammar to recognize csv files.

For the sake of the exercise, you'll only consider one type of separator (comma ","). You are not obliged to be able to match strings like `"Super, ""luxurious"" truck"`.

2.2 Derivation trees and attributions

EXERCISE #3 ► **Arithmetic expressions**
Let us consider the following grammar (the end of an expression is a semicolon):

$$
\begin{aligned}
Z &\to E;\\
E &\to E+T\\
E &\to T\\
T &\to T*F\\
T &\to F\\
F &\to (E)
\end{aligned}
$$

- What are the derivation trees for $1+2+3$, $1+2*3$, $(1+2)*3$?
- Attribute the grammar to evaluate arithmetic expressions.

EXERCISE #4 ► **Prefixed expressions**
Consider prefixed expressions like `* + * 3 4 5 7` and assigments of such expressions to variables: `a=* + * 3 4 5  7`. Identifiers are allowed in expressions.

- Give a grammar that recognizes lists of such assigments.
- Write derivations trees and verify the priority of * over +.
- Write grammar rules to compute the values of E expressions.
- Write grammar rules to construct infix assigments during parsing: the former assigment will be transformed into the *string* `a=(3 * 4 + 5)*7`. Be careful to avoid useless parentheses.
- Modify the attribution to verify that the use of each identifier is done after his first definition.

[1] see `https://en.wikipedia.org/wiki/Comma-separated_values`

2.3 The Mu-language

The objective here is to be familiar with the grammar of the language we will compile.

EXERCISE #5 ▸ **Mu-grammar**

Here is the (simplified) grammar for the Mu language (expr are numerical or boolean expressions):

```
grammar Mu;

prog
 : block EOF
 ;

block
 : stat*   #statList
 ;

stat
 : assignment
 | if_stat
 | while_stat
 | log
 | OTHER {System.err.println("unknown␣char:␣" + $OTHER.text);}
 ;

assignment
 : ID ASSIGN expr SCOL #assignStat
 ;

if_stat
 : IF condition_block (ELSE IF condition_block)* (ELSE stat_block)? #ifStat
 ;

condition_block
 : expr stat_block  #condBlock
 ;

stat_block
 : OBRACE block CBRACE
 | stat
 ;

while_stat
 : WHILE expr stat_block #whileStat
 ;

log
 : LOG expr SCOL #logStat
 ;
```

Write two valid programs for this grammar.