TD 2

AST, Attributions, Types

2.1 Derivation trees and attributions

EXERCISE #1 ► **Arithmetic expressions**
Let us consider the following grammar (the end of an expression is a semicolon):

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

- What are the derivation trees for $1+2+3;$, $1+2*3;$, $(1+2)*3;$?

EXERCISE #2 ► **Declarations of variables**
Write a grammar that accepts declarations of variables like:

```
int x=1;
float y,z;
int t;
float u,v=0;
```

and rejects:

```
int x, int y;
```

Then write an attribution that prints individual declarations (of the first case) like:

```
int x=1; float y; float z; int t; float u; float v=0;
```

EXERCISE #3 ► **Prefixed expressions**
Consider prefixed expressions like `* + * 3 4 5 7` (or `* + 1 2 * 3 4`) 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.
- Write grammar rules to compute the values of the 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.

2.2 The MiniC language

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

EXERCISE #4 ► **MiniC-grammar**

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

```
grammar MiniC;

prog: function* EOF #progRule;

function: INTTYPE ID OPAR CPAR OBRACE vardecl_l block RETURN INT SCOL CBRACE #funcDecl;

vardecl_l: vardecl* #varDeclList;

vardecl: typee id_l SCOL #varDecl;

id_l
    : ID                #idListBase
    | ID COM id_l       #idList
    ;

block: stat*   #statList;

stat
    : assignment SCOL
    | if_stat
    | while_stat
    | print_stat
    ;

assignment: ID ASSIGN expr #assignStat;

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

condition_block: OPAR expr CPAR stat_block  #condBlock;

stat_block
    : OBRACE block CBRACE
    | stat
    ;

while_stat: WHILE OPAR expr CPAR  stat_block #whileStat;


print_stat
    : PRINTINT OPAR expr CPAR SCOL         #printintStat
    | PRINTFLOAT OPAR expr CPAR SCOL       #printfloatStat
    | PRINTSTRING OPAR expr CPAR SCOL      #printstringStat
    ;

expr_l
    : /* Nothing */       #exprListEmpty
    | expr                #exprListBase
    | expr COM expr_l     #exprList
    ;


expr
    : MINUS expr                           #unaryMinusExpr
    | NOT expr                             #notExpr
    | expr myop=(MULT|DIV|MOD)  expr       #multiplicativeExpr
    | expr myop=(PLUS|MINUS) expr          #additiveExpr
    | expr myop=(GT|LT|GTEQ|LTEQ)  expr    #relationalExpr
    | expr myop=(EQ|NEQ) expr              #equalityExpr
    | expr AND expr                        #andExpr
    | expr OR expr                         #orExpr
    | atom                                 #atomExpr
    ;

atom
    : OPAR expr CPAR #parExpr
    | (INT | FLOAT)  #numberAtom
    | (TRUE | FALSE) #booleanAtom
    | ID             #idAtom
    | STRING         #stringAtom
    ;
```

Write a valid program for this grammar.

2.3 Typing

We recall the rules of the course for Typing:

$$\frac{c \in \mathbb{Z}}{\Gamma \vdash c : \texttt{int}} \qquad \frac{\Gamma(x) = t \quad t \in \{\texttt{int}, \texttt{bool}\}}{\Gamma \vdash x : t} \qquad \frac{\Gamma \vdash S_1 \;\; \Gamma \vdash S_2}{\Gamma \vdash S_1; S_2}$$

$$\frac{\Gamma \vdash e : t \quad \Gamma \vdash x : t \quad t \in \{\texttt{int}, \texttt{bool}\}}{\Gamma \vdash x = e : \texttt{void}} \qquad \frac{\Gamma \vdash b : \texttt{bool} \quad \Gamma \vdash S : \texttt{void}}{\Gamma \vdash \texttt{while(}\, b\, \texttt{)\{}\, S\, \texttt{\}} : \texttt{void}}$$

$$\frac{\Gamma \vdash b : \texttt{bool} \quad \Gamma \vdash S_1 : \texttt{void} \quad \Gamma \vdash S_2 : \texttt{void}}{\Gamma \vdash \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 : \texttt{void}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}} \qquad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 < e_2 : \texttt{bool}}$$

EXERCISE #5 ► **Typing under given environment**

Considering $\Gamma = \{x_1 \mapsto \texttt{int}\}$, prove that the given sequence of instructions is well typed:

```
x1 = 3 ;
while (x1 < 15) {
  x1 = x1 + 1 ;
}
```

EXERCISE #6 ► **Construction of Gamma with variable declarations**

Using the following rules:

$$\frac{}{t \;\; vlist; \rightarrow_d [v \mapsto t \text{ for v in } vlist]}$$

$$\frac{D_1 \rightarrow_d \Gamma_1 \quad D_2 \rightarrow_d \Gamma_2 \quad Dom(\Gamma_1) \cap Dom(\Gamma_2) = \emptyset}{D1; D_2 \rightarrow_d \Gamma_1 \cup \Gamma_2}$$

Compute Γ the typing environnement for the given list of declarations:

```
int x;
bool b1, b2;
```

EXERCISE #7 ► **Boolean expressions**

Write all rules to type boolean expressions of MiniC.