ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DIPARTIMENTO DI
INFORMATICA - SCIENZA E INGEGNERIA

THE SIMPLAN LANGUAGE

COSIMO LANEVE
cosimo.laneve@unibo.it

CORSO 72671 - COMPLEMENTI DI LINGUAGGI DI PROGRAMMAZIONE

SIMPLAN

- is a basic **functional language** with types
  - it admits initialization
  - no assignment
  - two data-types (`int` and `bool`)
- it admits variable declarations
  - standard declaration `let int x = 4 ; in x+1`
  - multiple variable declarations `let int x = 4 ; int y = x+5 ; in x+y ;`
- it admits function definitions
  - standard definition `let int foo(int x) = x+1; in foo(34) ;`
  - nested function definitions
  - access to global variables

```
let int f(int x) =
    let int h(int y) = y+x ;
    in h(x+1) ;
in f(34) ;
```

SIMPLAN

- it does not admit assignments
- **it does not admits recursion**

ANTLR

`ANTLR` = ANother Tool for Language Recognition

- is a powerful parser generator for reading, processing, executing, or translating structured text or binary files
- it's widely used to build languages, tools, and frameworks
- from a grammar, ANTLR generates a parser that can build and walk parse trees

SIMPLAN

parser non-terminals are in lower-case characters

```
grammar SimpLan;
// PARSER RULES
prog     : exp ';'
         | let exp ';'
         ;

let      : 'let' (dec ';')+ 'in' ;
dec      : type ID '=' exp
         | type ID '(' ( param ( ',' param)* )? ')' (let)? exp
         ;

param    : type ID ;

type     : 'int' | 'bool'
         ;

exp      :  ('-')? left=term (('+' | '-') right=exp)?
         ;

term     : left=factor (('*' | '/') right=term)?
         ;

factor   : left=value ('==' right=value)?
         ;
```

SIMPLAN

```
grammar SimpLan;
// PARSER RULES
. . .

value   :  INTEGER
        | ('true' | 'false')
        | '(' exp ')'
        | 'if' exp 'then' '{' exp '}' 'else' '{' exp '}'
        | ID
        | ID '(' (exp (',' exp)* )? ')'
        ;

// LEXER RULES
fragment DIGIT : '0'..'9';
INTEGER        : DIGIT+;

fragment CHAR  : 'a'..'z' |'A'..'Z' ;
ID              : CHAR (CHAR | DIGIT)* ;

WS              : (' '|'\t'|'\n'|'\r')-> skip;
LINECOMENTS    : '//' (~('\n'|'\r'))* -> skip;
BLOCKCOMENTS    : '/*'( ~('/'|'*')|'/'~'*'|'*'~'/'|BLOCKCOMENTS)* '*/'
-> skip;
```

fragment = no node is generated in the syntax tree: digits are all collected in a node

lexer non-terminals are in upper-case characters

no node in the syntax tree is generated: the characters are skipped