Tarea Independiente 11/09/2025

David Núñez Franco

September 20, 2025

Inventario de Conceptos Claves

- Modelo de parsing lexer+parser
- Metáfora de Pacman de lexer
- Expresiones Regulares (RE) y Autómatas Finitos (FA)
- Gramáticas Libres de Contexto (CFG)
- Principios de ANTLR como ejemplo de un transpiler
- Reglas de lexer (para tokens)
- Reglas de Parser (para frases)
- Importancia de orden de la Reglas
- Importancia de si minúscula o mayúscula
- Metáfora de "Pacman" de grammar (en particular con recursividad)
- Pacman de lexer versus pacman de parser
- Proyecto node básico: package.json y npm (instalación del proyecto y build)

(

Ejercicio 1)

```
1. Generalice la gramática Expr.g4 para que también maneje números en punto flotante h
```

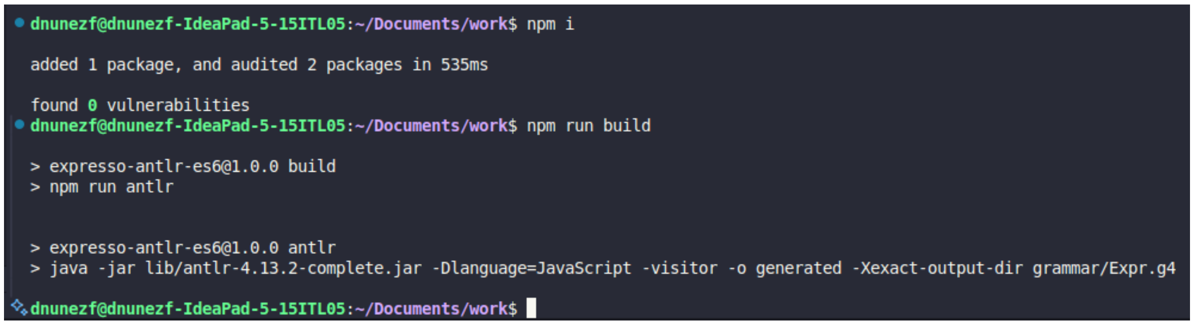
1.a

a) Escriba lo necesario en el LEXER para reconocer números en punto flotante no solo e

```
        // Agregamos el token FLOAT al LEXER en grammar/Expr.g4:

        // LEXER
        fragment DIGITS : [0-9]+ ;

        FLOAT
          : [+\-]? DIGITS '.' DIGITS ( [eE] [+\-]? DIGITS )?
          | [+\-]? DIGITS [eE] [+\-]? DIGITS
          ;

        INT : DIGITS ;
        NEWLINE: ('\r'? '\n') ;
        WS : [ \t]+ -> skip ;
```

1.b

b) Haga build usando npm, Verifique que no salgan errores del transpilador de ANTLR4.
OJO: Se debe ejecutar build cada vez que se cambie la gramática
El comando para hacer build es en la carpeta del proyecto en un cmd (terminal):
npm run build



```
dnunezf@dnunezf-IdeaPad-5-15ITL05:~/Documents/work$ npm i

added 1 package, and audited 2 packages in 535ms

found 0 vulnerabilities
dnunezf@dnunezf-IdeaPad-5-15ITL05:~/Documents/work$ npm run build

> expresso-antlr-es6@1.0.0 build
> npm run antlr


> expresso-antlr-es6@1.0.0 antlr
> java -jar lib/antlr-4.13.2-complete.jar -Dlanguage=JavaScript -visitor -o generated -Xexact-output-dir grammar/Expr.g4

dnunezf@dnunezf-IdeaPad-5-15ITL05:~/Documents/work$
```

Figure 1: Hallazgo

La carpeta generated/ se regenera sin errores.
En generated/ExprLexer.js y generated/Expr.tokens aparece el token FLOAT.

```
        // Generated from grammar/Expr.g4 by ANTLR 4.13.2
        // jshint ignore: start
        import antlr4 from 'antlr4';


        const serializedATN =
           [4,0,10,79,6,-1,2,0,7,0,2,1,7,1,2,2,7,2,2,3,7,3,2,
        4,7,4,2,5,7,5,2,6,7,6,2,7,7,7,2,8,7,8,2,9,7,9,2,10,7,10,1,0,1,0,1,1,1,1

        1,2,1,2,1,3,1,3,1,4,1,4,1,5,1,5,1,6,4,6,37,8,6,11,6,12,6,38,1,7,3,7,42,

        7,1,7,1,7,1,7,1,7,1,7,3,7,49,8,7,1,7,3,7,52,8,7,1,7,3,7,55,8,7,1,7,1,7,

        7,3,7,60,8,7,1,7,1,7,3,7,64,8,7,1,8,1,8,1,9,3,9,69,8,9,1,9,1,9,1,10,4,1

        74,8,10,11,10,12,10,75,1,10,1,10,0,0,11,1,1,3,2,5,3,7,4,9,5,11,6,13,0,1

        7,17,8,19,9,21,10,1,0,4,1,0,48,57,2,0,43,43,45,45,2,0,69,69,101,101,2,0

        9,9,32,32,86,0,1,1,0,0,0,0,3,1,0,0,0,0,5,1,0,0,0,0,7,1,0,0,0,0,9,1,0,0,

        0,11,1,0,0,0,0,15,1,0,0,0,0,17,1,0,0,0,0,19,1,0,0,0,0,21,1,0,0,0,1,23,1

        0,0,0,3,25,1,0,0,0,5,27,1,0,0,0,7,29,1,0,0,0,9,31,1,0,0,0,11,33,1,0,0,0

        13,36,1,0,0,0,15,63,1,0,0,0,17,65,1,0,0,0,19,68,1,0,0,0,21,73,1,0,0,0,2

        24,5,45,0,0,24,2,1,0,0,0,25,26,5,42,0,0,26,4,1,0,0,0,27,28,5,47,0,0,28,

        1,0,0,0,29,30,5,43,0,0,30,8,1,0,0,0,31,32,5,40,0,0,32,10,1,0,0,0,33,34,

        41,0,0,34,12,1,0,0,0,35,37,7,0,0,0,36,35,1,0,0,0,37,38,1,0,0,0,38,36,1,

        0,0,38,39,1,0,0,0,39,14,1,0,0,0,40,42,7,1,0,0,41,40,1,0,0,0,41,42,1,0,0

        0,42,43,1,0,0,0,43,44,3,13,6,0,44,45,5,46,0,0,45,51,3,13,6,0,46,48,7,2,

        0,47,49,7,1,0,0,48,47,1,0,0,0,48,49,1,0,0,0,49,50,1,0,0,0,50,52,3,13,6,

        51,46,1,0,0,0,51,52,1,0,0,0,52,64,1,0,0,0,53,55,7,1,0,0,54,53,1,0,0,0,5

        55,1,0,0,0,55,56,1,0,0,0,56,57,3,13,6,0,57,59,7,2,0,0,58,60,7,1,0,0,59,

        1,0,0,0,59,60,1,0,0,0,60,61,1,0,0,0,61,62,3,13,6,0,62,64,1,0,0,0,63,41,

        0,0,0,63,54,1,0,0,0,64,16,1,0,0,0,65,66,3,13,6,0,66,18,1,0,0,0,67,69,5,

        0,0,68,67,1,0,0,0,68,69,1,0,0,0,69,70,1,0,0,0,70,71,5,10,0,0,71,20,1,0,

        0,72,74,7,3,0,0,73,72,1,0,0,0,74,75,1,0,0,0,75,73,1,0,0,0,75,76,1,0,0,0
```

```
76,77,1,0,0,0,77,78,6,10,0,0,78,22,1,0,0,0,10,0,38,41,48,51,54,59,63,68

75,1,6,0,0];


const atn = new antlr4.atn.ATNDeserializer().deserialize(serializedATN);

const decisionsToDFA = atn.decisionToState.map( (ds, index) => new antlr4.dfa.DFA(ds, index) );

export default class ExprLexer extends antlr4.Lexer {

    static grammarFileName = "Expr.g4";
    static channelNames = [ "DEFAULT_TOKEN_CHANNEL", "HIDDEN" ];
    static modeNames = [ "DEFAULT_MODE" ];
    static literalNames = [ null, "'-'", "'*'", "'/'", "'+'", "'('", "')'" ];
    static symbolicNames = [ null, null, null, null, null, null, null, "FLOAT",
    "INT", "NEWLINE", "WS" ];
    static ruleNames = [ "T__0", "T__1", "T__2", "T__3", "T__4", "T__5", "DIGITS",
    "FLOAT", "INT", "NEWLINE", "WS" ];

    constructor(input) {
        super(input)
        this._interp = new antlr4.atn.LexerATNSimulator(this, atn, decisionsToDFA, new antlr4.atn.PredictionContextCache());
    }
}

ExprLexer.EOF = antlr4.Token.EOF;
ExprLexer.T__0 = 1;
ExprLexer.T__1 = 2;
ExprLexer.T__2 = 3;
ExprLexer.T__3 = 4;
ExprLexer.T__4 = 5;
ExprLexer.T__5 = 6;
ExprLexer.FLOAT = 7;
ExprLexer.INT = 8;
ExprLexer.NEWLINE = 9;
ExprLexer.WS = 10;
```

Listing 1: ExprLexer.js

```
T__0=1
T__1=2
T__2=3
T__3=4
T__4=5
T__5=6
FLOAT=7
INT=8
NEWLINE=9
WS=10
'-'=1
'*'=2
'/'=3
'+'=4
'('=5
')'=6
```

Listing 2: Expr.tokens

1.c

```
c) Abra src\AstBuilder.mjs: Este es un visitor. Este visitor construye Nodes del model
// int: INT
visitInt(ctx) {
 return new Num(Number(ctx.getText()))
}
Observe que Num es nuestro Num de src\ast.mjs.
Este método es el que maneja el caso donde el visitor está viendo un entero (un INT).
Modifique para que sea FLOAT
```

Primero, se corrigió el LEXER, quitando el signo opcional en FLOAT, para no chocar con unaryMinus

```
// LEXER
fragment DIGITS : [0-9]+ ;

FLOAT
  : DIGITS '.' DIGITS ( [eE] [+\-]? DIGITS )?
  | DIGITS [eE] [+\-]? DIGITS
  ;

INT : DIGITS ;
NEWLINE: ('\r'? '\n') ;
WS : [ \t]+ -> skip ;
```

Listing 3: Expr.g4

Luego, editó Expr.g4 para que acepte FLOAT en el parser:

```
        expr
          : '-' expr                        # unaryMinus
          | expr op=('*'|'/') expr          # MulDiv
          | expr op=('+'|'-') expr          # AddSub
          | INT                             # int
          | FLOAT                           # float
          | '(' expr ')'                    # parens
          ;
```

Listing 4: Expr.g4

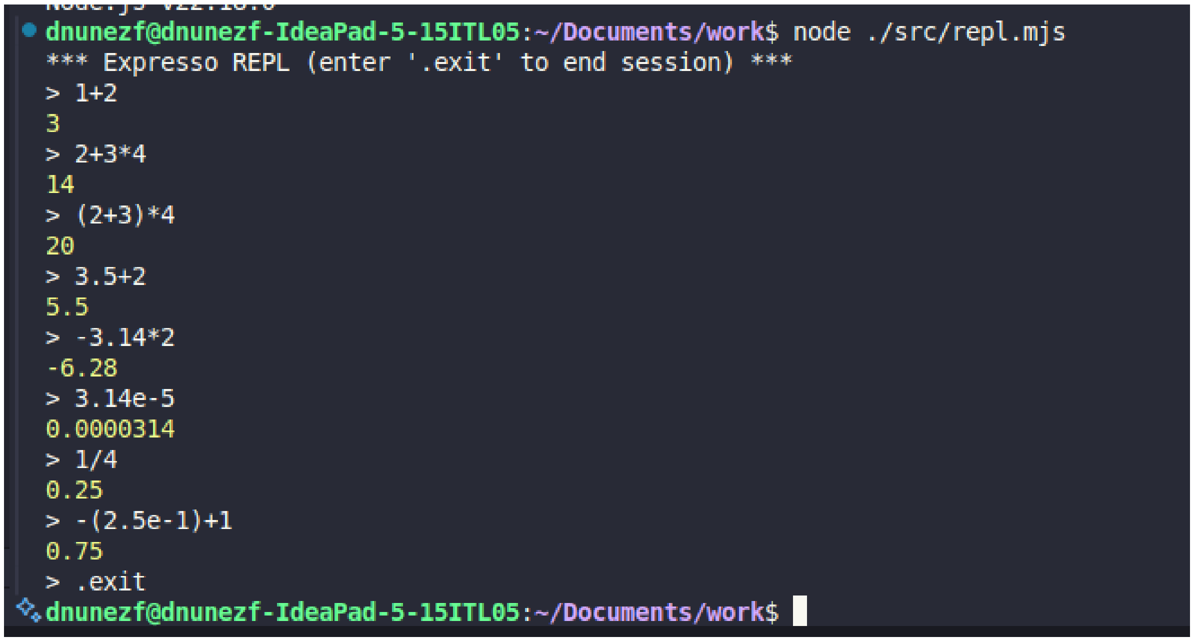
Finalmente, se añade el handler en AstBuilder.mjs:

```
        // float: FLOAT
        visitFloat(ctx) {
            return new Num(Number(ctx.getText()))
        }
```

Listing 5: AstBuilder.mjs

1.d

```
d) Pruebe con el repl.mjs:  Comando en CMD:
npm src\repl.mjs
y teclee con expresiones que tengan flotantes y/o enteros
```



```
dnunezf@dnunezf-IdeaPad-5-15ITL05:~/Documents/work$ node ./src/repl.mjs
*** Expresso REPL (enter '.exit' to end session) ***
> 1+2
3
> 2+3*4
14
> (2+3)*4
20
> 3.5+2
5.5
> -3.14*2
-6.28
> 3.14e-5
0.0000314
> 1/4
0.25
> -(2.5e-1)+1
0.75
> .exit
dnunezf@dnunezf-IdeaPad-5-15ITL05:~/Documents/work$
```

Figure 2: Prueba de ejecución