Tarea Independiente 18/09/2025

David Núñez Franco

September 24, 2025

Inventario de Conceptos Claves

- Problema de ambigüedad de una gramática
- Solución en ANTLR por orden de reglas
- Objeto contexto (CTX) en ANTLR
- Relación entre regla y propiedades o métodos del contexto en ANTLR
- El visitor generado por ANTLR
- Interceptando métodos por medio de herencia en visitadores específicos

Ejercicio 0

Modifique la tokenización de número de Expr.g4 para que correctamente permita casos como .25, y 1. como números.

Solución

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

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

// LEXER
INT : DIGITS ;
```

Listing 1: Expr.g4

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

Listing 2: EvalVisitor.mjs

```
        *** Expresso REPL (enter '.exit' to end session) ***
        > .25
        0.25
        > 1.
        1
        > 1.+.25*2
        1.5
```

Listing 3: Casos de Prueba

Ejercicio 1

Añada un operador ternario tipo Java(C) (?:) al demo y logre que se evalúa en el repl. Que tenga más prioridad que los aritméticos.

Solución

```
        // Expresiones con precedencia y unario '-' (parser)
        expr
          : '-' expr                                  # unaryMinus
          | <assoc=right> expr '?' expr ':' expr      # ternary
          | expr op=('*'|'/') expr                    # MulDiv
          | expr op=('+'|'-') expr                    # AddSub
          | INT                                       # int
          | FLOAT                                     # float
          | '(' expr ')'                              # parens
          ;
```

Listing 4: Expr.g4

```
        export class TernaryOp extends Operation {
            constructor(cond, thenExpr, elseExpr) {
                super(new Oper('?:'), cond, thenExpr, elseExpr)
            }

            get cond() { return this.children[0] }
            get thenExpr() { return this.children[1] }
            get elseExpr() { return this.children[2] }
```

```
        toString() { return `${this.cond} ? ${this.thenExpr} : ${this.elseExpr}` }
    }
```

Listing 5: ast.mjs

```
    // ternary: expr '?' expr ':' expr
    visitTernary(ctx) {
        const c = this.visit(ctx.expr(0));
        const t = this.visit(ctx.expr(1));
        const e = this.visit(ctx.expr(2));
        return new TernaryOp(c, t, e);
    }
```

Listing 6: AstBuilder.mjs

```
    if (node instanceof TernaryOp) {
        const c = this.eval(node.cond);
        return (Number(c) !== 0)
        ? this.eval(node.thenExpr)
        : this.eval(node.elseExpr);
    }
```

Listing 7: EvalAst.mjs

```
    // ternary: expr '?' expr ':' expr
    visitTernary(ctx) {
        const c = Number(this.visit(ctx.expr(0)));
        return c !== 0 ? this.visit(ctx.expr(1)) : this.visit(ctx.expr(2));
    }
```

Listing 8: EvalVisitor.mjs

```
    *** Expresso REPL (enter '.exit' to end session) ***
    > 1?2:3
    2
    > 0?2:3
    3
    > 1+2?3:4
    4
    > 1?2+3:4*5
    25
    > 1?2:0?3:4
    2
    >
```

Listing 9: Casos de Prueba

Ejercicio 2

Añada una expresión let de la forma de expresión let x = expr in expr. Por ahora no se puede evaluar porque no hemos visto ambientes (clausuras), pero haga que el evaluador diga un error que la está viendo al menos. Note que ocupa una regla id y una de tokenización respectiva. Ya el modelo ast tiene nodo para Id.

Solución

```
        // Expresiones con precedencia y unario '-' (parser)
        expr
         : '-' expr                                  # unaryMinus
         | <assoc=right> expr '?' expr ':' expr      # ternary
         | expr op=('*'|'/') expr                    # MulDiv
         | expr op=('+'|'-') expr                    # AddSub
         | LET ID ASSIGN expr IN expr                # letExpr
         | ID                                        # idExpr
         | INT                                       # int
         | FLOAT                                     # float
         | '(' expr ')'                              # parens
         ;


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

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

        // LEXER
        INT     : DIGITS ;
        LET     : 'let' ;
        IN      : 'in' ;
        ASSIGN  : '=' ;
        ID      : [a-zA-Z_][a-zA-Z_0-9]* ; //acepta nombres tipo x,
           abc, total1, _tmp, my_var2.
        NEWLINE : ('\r'? '\n') ;
        WS      : [ \t]+ -> skip ;
```

Listing 10: Expr.g4

```
        export class LetExpr extends Node {
            constructor(id, valueExpr, bodyExpr) {
```

```
        super(id, valueExpr, bodyExpr)
    }

    get id(){ return this.head } // ID
    get valueExpr(){ return this.children[0] }
    get bodyExpr(){ return this.children[1] }
    toString(){ return `let ${this.id} = ${this.valueExpr}
       in ${this.bodyExpr}` }
}
```

Listing 11: ast.mjs

```
// letExpr: LET ID ASSIGN expr IN expr
visitLetExpr(ctx) {
    const name = ctx.ID().getText();
    const id = new Id(name);
    const val = this.visit(ctx.expr(0));
    const body = this.visit(ctx.expr(1));
    return new LetExpr(id, val, body);
}

// idExpr: ID
visitIdExpr(ctx) {
    return new Id(ctx.ID().getText());
}
```

Listing 12: AstBuilder.mjs

```
if (node instanceof LetExpr) {
    throw new Error(`LetExpr not supported yet: ${node.
       toString()}`);
}

if (node instanceof Id) {
    throw new Error(`Variable usage not supported yet: ${
       node.name}`);
}
```

Listing 13: EvalAst.mjs

```
// letExpr: LET ID ASSIGN expr IN expr
visitLetExpr(ctx) {
    const id = ctx.ID().getText();
    const val = this.visit(ctx.expr(0));
    const bodyText = ctx.expr(1).getText();
    throw new Error(`LetExpr not supported yet: let ${id} =
       ${val} in ${bodyText}`);
}
```

Listing 14: EvalVisitor.mjs

```
*** Expresso REPL (enter '.exit' to end session) ***
> let x = 2 in x+2
Error: LetExpr not supported yet: let x = 2 in x + 2
>
```

Listing 15: Pruebas Funcionales