PROYECTO 2

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

MANUAL TECNICO

ANTENNAE

FRANCISCO LUIS SUAREZ LÓPEZ

201807190

Funcionamiento de la aplicación

CODIGO DE LA APLICACIÓN

- **Backend con nodeJS**

```
Backend > src > JS index.js > ...
You, 20 days ago | 1 author (You)
const express = require("express");
const app = express();
const morgan = require("morgan");
const cors = require("cors");

//Settings
app.set("port", process.env.PORT || 3000);
app.set("json spaces", 2);

//Middlewares
app.use(morgan("dev"));
app.use(cors());
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

//Start REST
//Routes
app.use(require("./Routes/routes"));

//Starting server
app.listen(3000, () => {
  console.log("Server on port", app.get("port"));
});
```

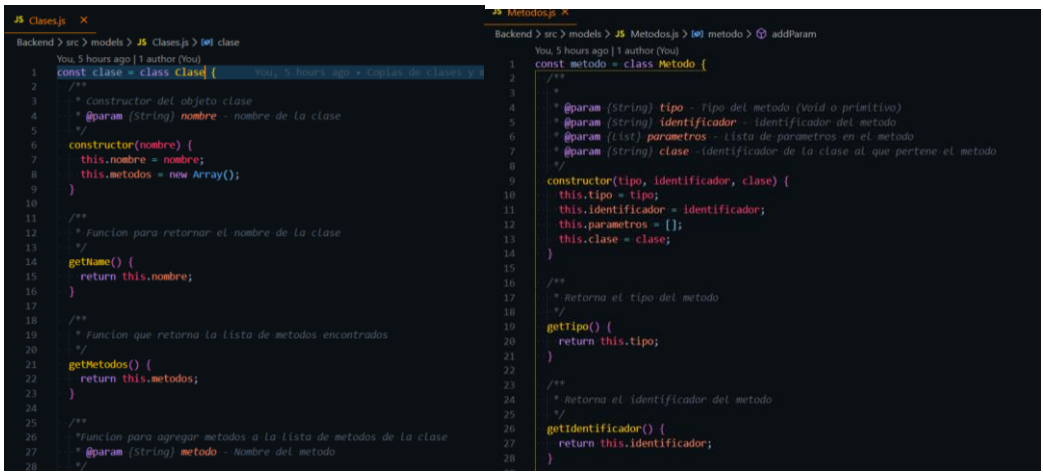
  - Rutas del backend

```
Backend > src > Routes > JS routes.js > ...
You, 5 hours ago | 1 author (You)
const { Router } = require("express");
const router = Router();
const createAST = require("../Parser/parser").createAST;
const Copias = require("../Copias/Copias");

router.get("/", (req, res) => {
  res.json({ Saludo: "Bienvenido a mi backend" });
});

router.post("/parser", async (req, res) => {
  const text = req.body.text;
  const analisis = await createAST.create(text);
  const ast = JSON.stringify(analisis.AST, null, 2);
  const errores = JSON.stringify(analisis.ERRORES, null, 2);
  res.send({ message: "Compilacion terminada", AST: ast, ERRORES: errores });
});

router.post("/copias", async (req, res) => {
  const analisisOriginal = await createAST.create(req.body.original);
  const analisisCopia = await createAST.create(req.body.copia);
  const original = await Copias.Copias.analisisArchivo(analisisOriginal.AST);
  const copia = await Copias.Copias.analisisArchivo(analisisCopia.AST);
  const resultadoCopia = await Copias.Copias.compareClase(original, copia);
  res.send({ message: "Analisis terminado", COPIAS: resultadoCopia });
});

module.exports = router;
```

  - Analisis de Copias

```
Backend > src > Copias > JS Copias.js > [e] Copias
You, an hour ago | 1 author (You)
const Clases = require("../models/Clases");
const Metodos = require("../models/Metodos");
const Parametros = require("../models/Parametros");

var clasesEncontradas = new Array();
var newClase = new Clases.clase(null);
var newMetodo_Funcion = new Metodos.metodo(null, null, null);
const Copias = {
  /**
   * Funcion para ver los los odos de la raiz del archivo (imports y clases)
   * @param {Objeto AST} ast
   */
  analisisArchivo: function (ast) {
    if (ast["RAIZ"] !== undefined) {
      clasesEncontradas = new Array();
      this.analisisClases(ast["RAIZ"]);
      return clasesEncontradas;
    }
  },

  /**
   * Funcion recursiva para entrar a los nodos que tengan que ver con clases
   * @param {Objeto AST} ast
   */
  analisisClases: function (ast) {
    if (ast["CLASS"] !== undefined) {
      this.analisisClases(ast["CLASS"]);
      clasesEncontradas.push(this.analisisClase(ast["CLASS"]));
```

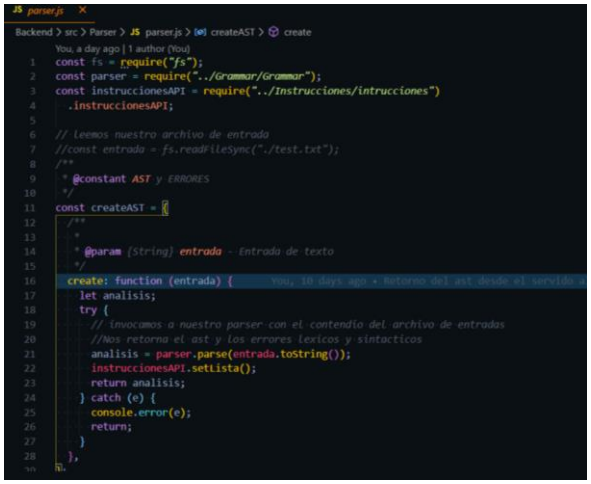
- Modelos de objetos utilizados



```
// Clases.js
const clase = class Clase {
  /**
   * Constructor del objeto clase
   * @param {String} nombre - nombre de la clase
   */
  constructor(nombre) {
    this.nombre = nombre;
    this.metodos = new Array();
  }

  /**
   * Funcion para retornar el nombre de la clase
   */
  getName() {
    return this.nombre;
  }

  /**
   * Funcion que retorna la lista de metodos encontrados
   */
  getMetodos() {
    return this.metodos;
  }

  /**
   *Funcion para agregar metodos a la lista de metodos de la clase
   * @param {String} metodo - Nombre del metodo
   */
```

```
// Metodos.js
const metodo = class Metodo {
  /**
   *
   * @param {String} tipo - Tipo del metodo (Void o primitivo)
   * @param {String} identificador - identificador del metodo
   * @param {List} parametros - Lista de parametros en el metodo
   * @param {String} clase -identificador de la clase al que pertene el metodo
   */
  constructor(tipo, identificador, clase) {
    this.tipo = tipo;
    this.identificador = identificador;
    this.parametros = [];
    this.clase = clase;
  }

  /**
   * Retorna el tipo del metodo
   */
  getTipo() {
    return this.tipo;
  }

  /**
   * Retorna el identificador del metodo
   */
  getIdentificador() {
    return this.identificador;
  }
```

- Parser para el análisis



```
// parser.js
const fs = require("fs");
const parser = require("../Grammar/Grammar");
const instruccionesAPI = require("../Instrucciones/intrucciones")
  .instruccionesAPI;

// Leemos nuestro archivo de entrada
//const entrada = fs.readFileSync("./test.txt");
/**
 * @constant AST y ERRORES
 */
const createAST = {
  /**
   *
   * @param {String} entrada - Entrada de texto
   */
  create: function (entrada) {
    let analisis;
    try {
      // invocamos a nuestro parser con el contendio del archivo de entradas
      //Nos retorna el ast y los errores lexicos y sintacticos
      analisis = parser.parse(entrada.toString());
      instruccionesAPI.setLista();
      return analisis;
    } catch (e) {
      console.error(e);
      return;
    }
  },
};
```

- Léxico y Gramática utilizada en el proyecto (Jison)

```
/**
 * Ejemplo Intérprete Sencillo con Jison utilizando Nodejs en Ubuntu
 */


/* Definición Léxica */
%lex


%options case-sensitive
```

```
%%

\s+                                                                    //
Espacios en blanco
"//".*                                                                 //
Comentario de linea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]                    // Comentario multilina

//Tipos de datos
"int"                  return 'RINT';
"double"                    return 'RDOUBLE';
"boolean"                   return 'RBOOLEAN';
"char"                 return 'RCHAR';
"String"                    return 'RSTRING';

//Palabras reservadas
"true"                                          return "RTRUE";
"false"                                         return "RFALSE";
"if"                        return 'RIF';
"else"                      return 'RELSE';
"switch"                    return 'RSWITCH';
"case"                      return 'RCASE';
"default"                   return 'RDEFAULT';
"break"                         return 'RBREAK';
"while"                         return 'RWHILE';
"do"                        return 'RDO';
"for"                       return 'RFOR';
"continue"                      return 'RCONTINUE';
"return"                        return 'RRETURN';
```

```
"System.out.println"return "RPRINT";
"System.out.print"          return "RPRINT";
"class"                     return 'RCLASS';
"import"                    return 'RIMPORT';
"main"              return 'RMAIN';
"void"              return 'RVOID';

//Operaciones aritmeticas
"++"                        return 'INCREMENTO';
"+"                             return 'SUMA';
"--"                        return 'DECREMENTO';
"-"                             return 'RESTA';
"*"                             return 'MULTIPLICACION';
"/"                             return 'DIVISION';
"^"                             return 'POTENCIA';
"%"                             return 'MODULO';

//Operciones relaciones
"=="                        return 'IGUALDAD';
"!="                        return 'DISTINTO';
">="                        return 'MAYORIGUALQUE';
">"                             return 'MAYORQUE';
"<="                        return 'MENORIGUALQUE';
"<"                             return 'MENORQUE';

//Operaciones logicas
"&&"                        return 'AND'
"||"                        return 'OR';
```

```
"!"                         return 'NOT';


//Simbolos del lenguaje
"{"                         return 'LLAVEIZQUIERDA';
"}"                         return 'LLAVEDERECHA';
"("                         return 'PARENTESISIZQUIERDO';
")"                         return 'PARENTESISDERECHO';
";"                         return 'PUNTOYCOMA';
","                         return 'COMA';
"="                         return 'IGUAL';
":"                         return 'DOSPUNTOS';


//TIPOS DE EXPRESIONES YA SEAN NUMERICAS, CADENAS DE TEXTO, CARACTERES O IDENTIFICADORES
\"([^\\\"]|\\.)*\"          { yytext = yytext.substr(1,yyleng-2); return 'CADENA'; }
\'([^\\\"]|\\.)\'           { yytext = yytext.substr(1,yyleng-2); return 'CARACTER'; }
[0-9]+("."[0-9]+)?\b  return 'NUMERO';
([a-zA-Z_])[a-zA-Z0-9_]*    return 'IDENTIFICADOR';



<<EOF>>                     return 'EOF';
.                           { console.error('Este es un error léxico: ' + yytext + ', en la linea: ' + yylloc.first_line + ', en la columna: ' + yylloc.first_column);
instruccionesAPI.pushLista(instruccionesAPI.errorLS("Lexico", undefined, yytext, yylloc.first_line, yylloc.first_column)); }


/lex
```

```
%{
        const instruccionesAPI      =
require("../Instrucciones/intrucciones").instruccionesAPI;
%}
/* Asociación de operadores y precedencia */

%left 'AND' 'OR'
%left 'IGUALDAD' 'DISTINTO'
%left 'MENORQUE' 'MENORIGUALQUE' 'MAYORQUE'
'MAYORIGUALQUE'
%left 'SUMA' 'RESTA'
%left 'MULTIPLICACION' 'DIVISION'
%left 'POTENCIA' 'MODULO'
%left UMENOS
%right 'NOT'
%right 'INCREMENTO' 'DECREMENTO'

%start INICIO

%% /* Definición de la gramática */

//Metodo de inicio de la gramatica
//Retornaremos el ast y los errores
// Cuado se haya reconocido la entrada completa retornamos el AST
INICIO
        : EOF {return {AST: instruccionesAPI.raiz(undefined, undefined),
ERRORES: instruccionesAPI.getLista()};}
        | IMPORTS EOF {return {AST: instruccionesAPI.raiz($1, undefined),
ERRORES: instruccionesAPI.getLista()};}
```

| IMPORTS CLASS EOF {return {AST: instruccionesAPI.raiz($1, $2), ERRORES: instruccionesAPI.getLista()};}

| CLASS EOF {return {AST: instruccionesAPI.raiz(undefined, $1), ERRORES: instruccionesAPI.getLista()};}

;

//Imports de clases

IMPORTS

: IMPORTS RIMPORT IMPORT { $$ = instruccionesAPI.inicio_imports($1, $3); }

| RIMPORT IMPORT { $$ = instruccionesAPI.inicio_imports(undefined, $2); }

;

IMPORT

:IDENTIFICADOR PUNTOYCOMA { $$ = instruccionesAPI.inst_import($1); }

|error PUNTOYCOMA { console.error('Este es un error sintáctico: ' + yy.parser.hash.token + ', en la linea: ' + @1.first_line + ', en la columna: ' + @1.first_column + " se esperaba: " + yy.parser.hash.expected); instruccionesAPI.pushLista(instruccionesAPI.errorLS("Sintactico", yy.parser.hash.expected, yy.parser.hash.token, @1.first_line, @1.first_column)); }

;

//Metodo para el analisis de clases

CLASS

: CLASS CLASSP { $$ = instruccionesAPI.inicio_clases($1, $2); }

| CLASSP { $$ = instruccionesAPI.inicio_clases(undefined, $1); }

;

CLASSP

```
    : RCLASS IDENTIFICADOR BLOQUE_CLASE { $$ = instruccionesAPI.inst_class($2, $3); }

    ;


//Bloque de una clase identificada { Instrucciones }

BLOQUE_CLASE

    : LLAVEIZQUIERDA BLOQUE_CLASEP LLAVEDERECHA { $$ = $2; }

    | LLAVEIZQUIERDA LLAVEDERECHA { $$ = undefined; }

    | LLAVEIZQUIERDA error LLAVEDERECHA { console.error('Este es un error sintáctico: ' + yy.parser.hash.token +  ', en la linea: ' + @1.first_line + ', en la columna: ' + @1.first_column + " se esperaba: " + yy.parser.hash.expected );
instruccionesAPI.pushLista(instruccionesAPI.errorLS("Sintactico", yy.parser.hash.expected, yy.parser.hash.token, @1.first_line, @1.first_column)); }

    ;


BLOQUE_CLASEP

    : BLOQUE_CLASEP BLOQUE_CLASEPP { $$ = instruccionesAPI.bloque_class($1, $2); }

    | BLOQUE_CLASEPP { $$ = instruccionesAPI.bloque_class(undefined, $1); }

    ;


BLOQUE_CLASEPP

    : DECLARACION { $$ = $1; }

    | METODOS { $$ = $1; }

    ;


//Metodos que pueden venir adentro del bloque de una clase

METODOS
```

```
    : TIPO IDENTIFICADOR ASIGNACIONPARAMETROS BLOQUE_METODO { $$ = instruccionesAPI.inst_funciones($1, $2, $3, $4); }

    | RVOID IDENTIFICADOR ASIGNACIONPARAMETROS BLOQUE_METODO { $$ = instruccionesAPI.inst_metodos($2, $3, $4); }

    | RVOID RMAIN PARENTESISIZQUIERDO PARENTESISDERECHO BLOQUE_METODO { $$ = instruccionesAPI.inst_metodos($2, undefined, $5); }

    ;


//Bloque de instrucciones en un metodo

BLOQUE_METODO

    : LLAVEIZQUIERDA INSTRUCCIONES LLAVEDERECHA { $$ = $2; }

    | LLAVEIZQUIERDA LLAVEDERECHA { $$ = undefined; }

    ;


//Asignacion de parametros que puede o no tener un metodo o funcion

ASIGNACIONPARAMETROS

    : PARENTESISIZQUIERDO LISTAPARAMETROS PARENTESISDERECHO { $$ = $2; }

    | PARENTESISIZQUIERDO PARENTESISDERECHO { $$ = undefined; }

    ;


LISTAPARAMETROS

    : LISTAPARAMETROS COMA PARAMETROS { $$ = instruccionesAPI.lista_parametros($1, $2, $3); }

    | PARAMETROS { $$ = instruccionesAPI.lista_parametros(undefined, undefined, $1); }

    ;
```

```
PARAMETROS
    : TIPO IDENTIFICADOR { $$ = instruccionesAPI.parametro($1, $2); }
    ;


// Metodo de instrucciones
INSTRUCCIONES
    : INSTRUCCIONES INSTRUCCION { $$ = instruccionesAPI.bloque_instrucciones($1, $2); }
    | INSTRUCCION { $$ = instruccionesAPI.bloque_instrucciones(undefined, $1); }
;


//Posibles instrucciones como if-else, switch, while, do-while, for, llamadas a funciones, print, etc.
INSTRUCCION
    : IF { $$ = instruccionesAPI.inicio_if($1); }
    | SWITCH { $$ = $1; }
    | WHILE { $$ = $1; }
    | DO_WHILE { $$ = $1; }
    | FOR { $$ = $1; }
    | LLAMADAFUNCIONES { $$ = $1; }
    | BREAK { $$ = $1; }
    | RETURN { $$ = $1; }
    | CONTINUE { $$ = $1; }
    | PRINT { $$ = $1; }
    | DECLARACION { $$ = $1; }
    | ASIGNACION { $$ = $1; }
    | error { console.error('Este es un error sintáctico: ' + yy.parser.hash.token +  ', en la linea: ' + @1.first_line + ', en la columna: ' + @1.first_column + " se esperaba: " + yy.parser.hash.expected );
```

instruccionesAPI.pushLista(instruccionesAPI.errorLS("Sintactico", yy.parser.hash.expected, yy.parser.hash.token, @1.first_line, @1.first_column)); }

;

//Instruccion print

PRINT

:RPRINT PARENTESISIZQUIERDO EXPRESION PARENTESISDERECHO PUNTOYCOMA { $$ = instruccionesAPI.inst_print($3); }

;

//Instruccion declaracion de variables

DECLARACION

: TIPO DECLARACIONP PUNTOYCOMA { $$ = instruccionesAPI.bloque_declaraciones($1, $2); }

| error PUNTOYCOMA { console.error('Este es un error sintáctico: ' + yy.parser.hash.token + ', en la linea: ' + @1.first_line + ', en la columna: ' + @1.first_column + " se esperaba: " + yy.parser.hash.expected); instruccionesAPI.pushLista(instruccionesAPI.errorLS("Sintactico", yy.parser.hash.expected, yy.parser.hash.token, @1.first_line, @1.first_column)); }

;

DECLARACIONP

: DECLARACIONP COMA DECLARACIONPP { $$ = instruccionesAPI.bloque_declaracionesP($1, $2, $3); }

| DECLARACIONPP { $$ = instruccionesAPI.bloque_declaracionesP(undefined, undefined, $1); }

;

DECLARACIONPP

```
        : IDENTIFICADOR IGUAL EXPRESION { $$ = instruccionesAPI.inst_declaracion($1, $2,  $3); }
        | IDENTIFICADOR { $$ = instruccionesAPI.inst_declaracion($1, undefined, undefined); }
        ;

//Instruccion asignacion de variables
ASIGNACION
        : IDENTIFICADOR IGUAL EXPRESION PUNTOYCOMA { $$ = instruccionesAPI.inst_asignacion($1, $2, $3); }
        | IDENTIFICADOR INCREMENTO PUNTOYCOMA { $$ = instruccionesAPI.inst_asignacion($1, $2, undefined); }
        | IDENTIFICADOR DECREMENTO PUNTOYCOMA { $$ = instruccionesAPI.inst_asignacion($1, $2, undefined); }
;

//Tipos de variables admitidos en el lenguaje
TIPO
        : RINT { $$ = $1; }
  | RDOUBLE { $$ = $1; }
  | RBOOLEAN { $$ = $1; }
        | RSTRING { $$ = $1; }
        | RCHAR { $$ = $1; }
;

//Expresiones
EXPRESION
        : RESTA EXPRESION %prec UMENOS { $$ = instruccionesAPI.operacionUnaria($2, "-"); }
        | NOT EXPRESION { $$ = instruccionesAPI.operacionUnaria($2, "!"); }
```

```
| EXPRESION SUMA EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "+"); }

| EXPRESION RESTA EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "-"); }

| EXPRESION MULTIPLICACION EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "*"); }

| EXPRESION DIVISION EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "/"); }

| EXPRESION MODULO EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "%"); }

| EXPRESION POTENCIA EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "^"); }

| EXPRESION AND EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "&&"); }

| EXPRESION OR EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "||"); }

| EXPRESION IGUALDAD EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "=="); }

| EXPRESION DISTINTO EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "!="); }

| EXPRESION MENORIGUALQUE EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "<="); }

| EXPRESION MENORQUE EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, "<"); }

| EXPRESION MAYORIGUALQUE EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, ">="); }

| EXPRESION MAYORQUE EXPRESION { $$ = instruccionesAPI.operacionBinaria($1, $3, ">"); }

| PARENTESISIZQUIERDO EXPRESION PARENTESISDERECHO { $$ = $2 }

| NUMERO { $$ = {NUMERO: Number($1)}; }

| RTRUE { $$ = {LOGICO: $1}; }

| RFALSE { $$ = {LOGICO: $1}; }

| CADENA { $$ = {CADENA: $1}; }
```

| CARACTER { $$ = {CARACTER: $1}; }

| IDENTIFICADOR PARENTESISIZQUIERDO LISTAEXPRESIONES PARENTESISDERECHO { $$ = instruccionesAPI.llamada_funciones($1, $3, undefined); }

| IDENTIFICADOR PARENTESISIZQUIERDO PARENTESISDERECHO { $$ = instruccionesAPI.llamada_funciones($1, undefined, undefined); }

| IDENTIFICADOR { $$ = {IDENTIFICADOR: $1}; }

;

//Metodo para llamadas a funcioens identificador(ListaExpresiones);

LLAMADAFUNCIONES

:IDENTIFICADOR PARENTESISIZQUIERDO LISTAEXPRESIONES PARENTESISDERECHO PUNTOYCOMA { $$ = instruccionesAPI.llamada_funciones($1, $3, $5); }

|IDENTIFICADOR PARENTESISIZQUIERDO PARENTESISDERECHO PUNTOYCOMA { $$ = instruccionesAPI.llamada_funciones($1, undefined, $4); }

;

LISTAEXPRESIONES

:LISTAEXPRESIONES COMA EXPRESION { $$ = instruccionesAPI.lista_expresiones($1, $2, $3); }

|EXPRESION { $$ = instruccionesAPI.lista_expresiones(undefined, undefined, $1); }

;

//Sentrencia if-else

IF

: RIF CONDICION BLOQUE_INSTRUCCIONES { $$ = instruccionesAPI.inst_if($2, $3, undefined, undefined); }

| RIF CONDICION BLOQUE_INSTRUCCIONES RELSE BLOQUE_INSTRUCCIONES { $$ = instruccionesAPI.inst_if($2, $3, undefined, instruccionesAPI.inst_else($5)); }

| RIF CONDICION BLOQUE_INSTRUCCIONES RELSE IF { $$ = instruccionesAPI.inst_if($2, $3, instruccionesAPI.inst_else_if($5), undefined); }

;

//Sentencia switch

SWITCH

:RSWITCH CONDICION LLAVEIZQUIERDA CASES LLAVEDERECHA { $$ = instruccionesAPI.inst_switch($2, $4); }

;

//Casos del switch

CASES

:CASES CASE_EVALUAR { $$ = instruccionesAPI.listaCasos($1, $2); }

|CASE_EVALUAR { $$ = instruccionesAPI.listaCasos(undefined, $1); }

;

CASE_EVALUAR

:RCASE EXPRESION DOSPUNTOS INSTRUCCIONES { $$ = instruccionesAPI.caso($1, $2, $4); }

|RCASE EXPRESION DOSPUNTOS { $$ = instruccionesAPI.caso($1, $2, undefined); }

|RDEFAULT DOSPUNTOS INSTRUCCIONES { $$ = instruccionesAPI.caso($1, undefined, $3); }

|RDEFAULT DOSPUNTOS { $$ = instruccionesAPI.caso($1, undefined, undefined); }

;

```
//Condiciones
CONDICION
        : PARENTESISIZQUIERDO EXPRESION PARENTESISDERECHO
        { $$ = instruccionesAPI.condicion($2); }
        ;

//Bloque de instrucciones para un algunas istrucciones
BLOQUE_INSTRUCCIONES
        : LLAVEIZQUIERDA INSTRUCCIONES LLAVEDERECHA  { $$ = $2;
}
        | LLAVEIZQUIERDA  LLAVEDERECHA { $$ = undefined; }
  ;

//Sentencia while
WHILE
        : RWHILE CONDICION BLOQUE_INSTRUCCIONES { $$ =
instruccionesAPI.inst_while($2, $3); }
        ;

//Sentencia Do-while
DO_WHILE
        : RDO BLOQUE_INSTRUCCIONES RWHILE CONDICION
PUNTOYCOMA { $$ = instruccionesAPI.inst_do_while($2, $4); }
        ;

//Sentencia for
FOR
        : RFOR PARENTESISIZQUIERDO DECLARACION EXPRESION
PUNTOYCOMA FORINC_DEC PARENTESISDERECHO
```

```
BLOQUE_INSTRUCCIONES { $$ = instruccionesAPI.inst_for($3, undefined, $4, $6, $8); }

    | RFOR PARENTESISIZQUIERDO DECLARACIONP PUNTOYCOMA EXPRESION PUNTOYCOMA FORINC_DEC PARENTESISDERECHO BLOQUE_INSTRUCCIONES { $$ = instruccionesAPI.inst_for(undefined, {ASIGNACION:$3, PUNTO_Y_COMA: ";"}, $5, $7, $9); }

    ;


FORINC_DEC

    : FORINC_DEC COMA INC_DEC { $$ = instruccionesAPI.modificador_For($1, $2, $3); }

    | INC_DEC { $$ = instruccionesAPI.modificador_For(undefined, undefined, $1); }

    ;


// Incremetento o decremento de variables a++ o a--

INC_DEC

    : IDENTIFICADOR INCREMENTO { $$ = instruccionesAPI.inst_modificacion($1, $2); }

    | IDENTIFICADOR DECREMENTO { $$ = instruccionesAPI.inst_modificacion($1, $2); }

;


//Sentencia return

RETURN

  : RRETURN EXPRESION PUNTOYCOMA { $$ = instruccionesAPI.returns($2); }

  | RRETURN PUNTOYCOMA { $$ = instruccionesAPI.returns(undefined); }

  ;


//Sentencia break
```

BREAK

: RBREAK PUNTOYCOMA { $$ = instruccionesAPI.breaks(); }

;

//Sentencia continue

CONTINUE

: RCONTINUE PUNTOYCOMA { $$ = instruccionesAPI.continues(); }

;

- **Frontend con Go y Javascript**
  - Funcionamiento de la aplicación JS

  - Servidor http con Go