

MADLINE ARIANA PÉREZ QUIÑÓNEZ  
201807117

# Manual Técnico

OLC 1 Sección “B”

08 / 05 / 2021

---

## ANALIZADOR.JSON

### ANALIZADOR LEXICO Y SINTACTICO

En esta parte se declaran los tokens a utilizar el lenguaje.

```
10 %lex
11
12 %options case-insensitive
13 %x string
14
15
16 %%
17
18 \s+ ..... /* skip whitespace */
19 "/*",* ..... // comentario simple línea
20 [/][*][^*]*+([/*][^*]*+)[/] ..... // comentario multiple líneas
21 // TIPO DE DATOS
22 "new" ..... return 'nnew'
23 "int" ..... return 'int'
24 "Double" ..... return 'Double'
25 "Boolean" ..... return 'Boolean'
26 "Char" ..... return 'Char'
27 "String" ..... return 'String'
28 "true" ..... return 'true'
29 "false" ..... return 'false'
30 // CICLOS
31 "list" ..... return 'list'
32 "add" ..... return 'add'
33 "if" ..... return 'if'
34 "else" ..... return 'else'
35 "switch" ..... return 'switch'
36 "case" ..... return 'case'
37 "default" ..... return 'default'
38 "break" ..... return 'break'
39 "while" ..... return 'while'
40 "for" ..... return 'for'
41 "do" ..... return 'do'
42 "continue" ..... return 'continue'
43 "return" ..... return 'return'
44 // METODOS Y FUNCIONES
45 "void" ..... return 'void'
46 "print" ..... return 'print'
```

En esta parte se declaran variables y se colocan las importaciones a utilizar para crear el árbol y lista de errores.

```
%{
    .....
    var cadena="";
    var listaErrores=[];
    const TIPO_ERROR .....= require('./controller/Enums/Tipo_Error');
    const ERRORES .....= require("./controller/Ambito/S_Error")
}%
```

```
%{
    .....
    const TIPO_OPERACION .....= require('./controller/Enums/TipoOperacion');
    const TIPO_VALOR .....= require('./controller/Enums/TipoValor');
    const TIPO_DATO .....= require('./controller/Enums/TipoDato');
    const INSTRUCCION .....= require('./controller/Instruccion/Instruccion');
}%
```

Se elabora el analizador sintáctico. Al ya tener todo eso, ya se empieza a estructurar la gramática.

(Explicación de la gramática en el archivo "Gramática\_201807117.pdf")

```

%start INICIO

%% /* GRAMATICA */
//CLASE
INICIO: OPINICIO EOF{var a={'errores':lista_Errores,'arbol':$1}; lista_Errores=[]; return a;}
;

//-----GLOBAL
> OPINICIO: OPINICIO CUERPO {$1.push($2); $$ = $1;} ...
;
> CUERPO: DEC_VAR ..... {$$ = $1;} ...
;

//-----METODOS Y FUNCIONES
> METFUNC: TIPO identificador parenA parenC llaveA OPCIONESCUERPO llaveC {$$ = INSTRUCCION.nuevaFUNCION($1,$2, null, $6, this._$.first_line, (this._$.first_line)); ...
;
> LLAMADA: identificador parenA LISTA_VALORES parenC {$$ = INSTRUCCION.Llamadas($1, $3, this._$.first_line, (this._$.first_line)); ...
;
> LISTA_VALORES: LISTA_VALORES coma VALORES {$1.push($3); $$ = $1;} ...
;
VALORES: EXPRESION {$$=$1}
;
> LISTAPARAMETROS: LISTAPARAMETROS coma PARAMETROS {$1.push($3); $$ = $1;} ...
;
PARAMETROS: TIPO identificador $$$ = INSTRUCCION.nuevaPARAMETRO($3,$4, this._$.first_line, (this._$.first_line)); ...
;

```

## INDEX.JS

En esta parte se declaran las rutas del api.

```

JS index.js  X
backend > JS index.js > ...
1  'use strict'
2  const express = require('express')
3  const bodyParser = require('body-parser')
4  let cors = require('cors')
5
6  const app = express()
7  const parser = require('./analizador')
8
9  app.use(bodyParser.json({limit: '50mb', extended: true}))
10 app.use(bodyParser.urlencoded({limit: '50mb', extended: true}))
11 app.use(cors())
12
13 app.get('/', (req, res) => {
14   res.send(respuesta)
15 })
16
17 const analizar = require('./endpoints/analizadores')(parser, app)
18 app.listen(3000, () => {
19   console.log("Servidor en puerto 3000")
20 })

```

## ANALIZADOR.JS

Aquí se manda a llamar es en donde:

- La entrada se envía al analizador json

- Revisa si no hay errores léxicos y sintácticos
- Envía el árbol creado por el json a el Ambito global para empezar el análisis semántico
- Manda a llamar una función para la creación del árbol AST
- Y por último envía la información al cliente

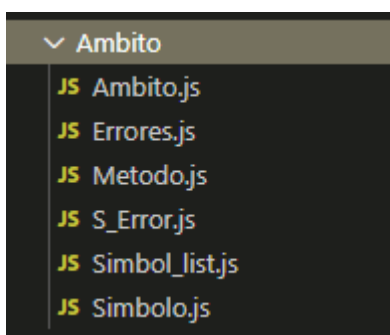
```

JS analizadores.js
backend > endpoints > JS analizadores.js > <unknown> > module.exports > app.post(/analizar/) callback
1 const Ambito = require("../controller/Ambito/Ambito")
2 const Errores = require("../controller/Ambito/Errores")
3 const Simbol_list = require("../controller/Ambito/Simbol_list")
4 const Bloque = require("../controller/Instruccion/Bloque")
5 const Global = require("../controller/Instruccion/Global")
6 const Grafico = require("../controller/Instruccion/Grafico")
7 const Generar_Graf = require("../controller/Instruccion/Generar_Graf")
8 module.exports=(parser, app)=>{
9   app.post(/analizar/, (req, res)=>{
10     var prueba = req.body.prueba
11     //try
12     var ast = parser.parse(prueba)
13     const Error = new Errores()
14     const Simbol= new Simbol_list()
15     var devuelve=""
16     ast.errores.forEach(error => {
17       Error.addErrores(error)
18       devuelve+= "Error "+error.tipo+": "+error.descripcion+"... linea: "+error.linea+" Columna: "+error.columna
19     });
20     const AmbitoGlobal = new Ambito(null)
21
22     devuelve+= Global(ast.arbol, AmbitoGlobal, Error, "Global", Simbol)
23     var Gdot="digraph mygraph { node [shape=box];\n"
24     ast.arbol.forEach(instruccion => {
25       Gdot += Grafico(instruccion, "Raiz", "Raiz")
26     });
27     Gdot+="\n"
28     var resultado = {
29       arbol: ast.arbol,
30       errores: Error,
31       Simbol_lit: Simbol,
32       consola: devuelve
33     }
34     res.send(resultado)
35     //console.log("-----act-----")
36     //console.log(Gdot)
37     //console.log("-----")
38     Generar_Graf(Gdot)
39     //console.log(Simbol)
40     // catch (error) { Find related code in Proyecto2_OLC1
41     // ... res.send(error)
42     //}
43   })
44 }

```

## AMBITO

En esta carpeta están los archivos que se utilizaran para guardar símbolos, errores y métodos.



## AMBITO.JS

Tiene los métodos para almacenar y acceder a los metodos y símbolos que se ingresen, según su entorno.

```

JS Ambitojs x
backend > controller > Ambito > JS Ambitojs > [0] <unknown>
1 class Ambito{
2   constructor(anterior){
3     this.anterior = _anterior
4   }
5   this.tablaSimbolos = new Map();
6   this.tablaMetodos = new Map();
7 }
8 // variables y de todos los simbolos
9 > addSimbolo(_s, _simbolo){ ...
10 }
11
12
13 > getSimbolo(_s){ ...
14 }
15
16
17 > existeSimbolo(_s){ ...
18 }
19
20 > existeSimbolodecla(_s){ ...
21 }
22
23 > actualizar(_s, _simbolo){ ...
24 }
25
26 // metodos
27 > addMetodo(_s, _metodo){ ...
28 }
29
30 > getMetodo(_s){ //(hola, clase simbolo)...
31 }
32
33 > existeMetodo(_s){ ...
34 }
35
36 > actualizarMetodo(_s, _metodo){ ...
37 }
38
39 }
40
41 module.exports = Ambito
Find related code in Proyecto2_OLC1

```

## ERRORES.JS

Tiene los métodos para almacenar los errores léxicos, semánticos y sintácticos.

```

class Errores{
  constructor(){
    this.tablaErrores=[];
  }
  // variables y de todos los simbolos
  addErrores(_simbolo){
    this.tablaErrores.push(_simbolo);
    //this.tablaErrores.set(_simbolo.tipo,_simbolo)
  }
  getErrores(_s){
  }
}
module.exports = Errores

```

## METODO.JS

Tiene la estructura con la que se almacenan los metodos.

```
JS Metodo.js x
backend > controller > Ambito > JS Metodo.js > Metodo > constructor
1 class Metodo{
2   constructor(_id, _lista_parametros, _entorno, _instrucciones, _tipo, _linea, _columna){
3     this.id = _id;
4     this.lista_parametros = _lista_parametros;
5     this.entorno = _entorno;
6     this.instrucciones = _instrucciones;
7     this.tipo = _tipo;
8     if(_tipo === "void"){
9       this.soy = "Metodo";
10    }else{
11      this.soy = "Funcion";
12    }
13    this.linea = _linea;
14    this.columna = _columna;
15  }
16 }
17
18
19 module.exports = Metodo
```

## S\_ERROR.JS

Tiene la estructura con que se guardaran los errores.

```
JS S_Error.js x
backend > controller > Ambito > JS S_Error.js > S_Error > constructor
1 class S_Error{
2   constructor(_tipo, _descripcion, _linea, _columna){
3     this.tipo = _tipo;
4     this.descripcion = _descripcion;
5     this.linea = _linea;
6     this.columna = _columna;
7   }
8 }
9
10 module.exports = S_Error
```

## SIMBOL\_LIST.JS

Tiene una lista en donde se guardarán todos los símbolos (métodos y variables). Esta servirá para la tabla de símbolos que se muestra en el cliente.

```
JS Simbol_list.js X
backend > controller > Ambito > JS Simbol_list.js > [?] <unknown>
1 class Simbol_list{
2     constructor(){
3         this.listado_Simbolos=[];
4     }
5     //variables y de todos los simbolos
6     add_s(_simbolo){
7         this.listado_Simbolos.push(_simbolo);
8         //this.tablaErrores.set(_simbolo.tipo,_simbolo)
9     }
10
11     getErrores(_s){
12
13     }
14
15 }
16
17 module.exports = Simbol_list Find related code in Proyecto2_OLC1
```

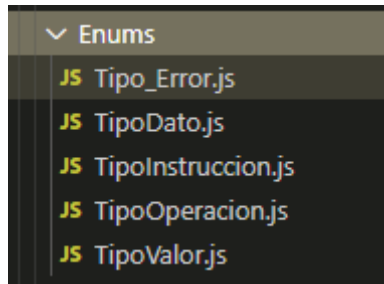
## SIMBOLO.JS

Tiene la estructura con que se guardaran las variables.

```
JS Simbolo.js X
backend > controller > Ambito > JS Simbolo.js > Simbolo > constructor
1 class Simbolo{
2     constructor(_id, _valor, _tipo, _entorno, _linea, _columna, _extra){
3         this.id = _id;
4         this.valor = _valor;
5         this.tipo = _tipo;
6         this.entorno = _entorno;
7         this.soy="Variable";
8         this.linea = _linea;
9         this.columna = _columna;
10        this.extra=_extra; Find related code in Proyecto2_OLC1
11    }
12 }
13
14 module.exports = Simbolo
```

## ENUMS

Aquí es en donde se declara los tipos de errores, tipos de datos, tipos de instrucciones, tipo de operaciones y tipo de valores.



## INSTRUCCIÓN

Esta carpeta almacena todas las instrucciones que se ejecutarán al recorrer el árbol que genera el analizador json



## ASIGNACION\_LISTA.JS Y ASIGNACION\_VEC.JS

Este archivo contiene el proceso de la asignación de valores para las posiciones de los vectores y de las listas.



```

JS Asignacion_LISTA.js X
backend > controller > Instruccion > JS Asignacion_LISTA.js > Asignacion_LISTA > tipos
1 const Operacion = require("../Operacion/Operacion");
2 const TIPO_DATO = require("../Enums/TipoDato");
3 const TIPO_ERROR = require("../Enums/Tipo_Error");
4 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
5 const ERRORES = require("../Ambito/S_Error");
6
7 function Asignacion_LISTA(_instruccion, _ambito, _Error, _entorno, Simbol){
8     const id = _instruccion.id;
9     const existe = _ambito.existeSimbolo(id)
10    if(existe){
11        var valor = Operacion(_instruccion.valor, _ambito, _Error, "", Simbol)
12        var simbolo = _ambito.getSimbolo(id)
13        var valores = simbolo.valor
14        //console.log(valores)
15        valores.push(valor)
16
17        var tipos = {
18            tipoSimbolo: simbolo.extra,
19            tipoNuevoValor: valor.tipo
20        }
21        if(tipos.tipoSimbolo !== tipos.tipoNuevoValor){
22            //console.log(valores)
23            simbolo.valor = valores
24            _ambito.actualizar(id, simbolo)
25            return null
26        }
27
28        var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No es posible asignar un valor de tipo ${tipos.tipoNuevoValor} a la lista '" +
29        _Error.addErrores(nuevo)
30        return valor.valores + " Error Semantico: No es posible asignar un valor de tipo '" + tipos.tipoNuevoValor + "' a la lista '"
31    }
32    var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "La lista '${String(id)}' no existe.", _instruccion.linea, _instruccion.columna)
33    _Error.addErrores(nuevo)
34    return "Error Semantico: La lista '${String(id)}' no existe... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}"
35
36 }
37
38
39 module.exports = Asignacion_LISTA

```

```

JS Asignacion_VEC.js X
backend > controller > Instruccion > JS Asignacion_VEC.js > Asignacion_VEC
1 const Operacion = require("../Operacion/Operacion");
2 const TIPO_DATO = require("../Enums/TipoDato");
3 const TIPO_ERROR = require("../Enums/Tipo_Error");
4 const ERRORES = require("../Ambito/S_Error");
5 function Asignacion_VEC(_instruccion, _ambito, _Error, _entorno, Simbol){
6     const id = _instruccion.id;
7     const existe = _ambito.existeSimbolo(id)
8     var posicion = Operacion(_instruccion.posicion, _ambito, _Error, _entorno, Simbol).valor
9     var posi = Operacion(_instruccion.posicion, _ambito, _Error, _entorno, Simbol)
10    if(existe){
11        var valor = Operacion(_instruccion.valor, _ambito, _Error, "", Simbol)
12        var valorviejo;
13        var simbolo = _ambito.getSimbolo(id)
14        var valores = simbolo.valor
15        var tam = valores.length;
16        if(posicion < tam && posicion >= 0){
17            if(posi.tipo === TIPO_DATO.ENTERO){
18                valorviejo = valores[posicion];
19                valorviejo = Operacion(valorviejo, _ambito, _Error, _entorno, Simbol)
20            } else {
21                var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "La posición ingresada: ${posicion} es de tipo ${_instruccion.posicion.tipo}, solo se permite asignar valores de tipo ENTERO... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}")
22                _Error.addErrores(nuevo)
23                return "Error Semantico: La posición ingresada: ${posicion} es de tipo ${_instruccion.posicion.tipo}, solo se permite asignar valores de tipo ENTERO... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}"
24            }
25        } else {
26            var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "La posición ingresada: ${posicion} no cumple con la longitud del vector... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}")
27            _Error.addErrores(nuevo)
28            return "Error Semantico: La posición ingresada: ${posicion} no cumple con la longitud del vector... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}"
29        }
30    }
31
32    var tipos = {
33        tipoSimbolo: valorviejo.tipo,
34        tipoNuevoValor: valor.tipo
35    }
36
37    if(tipos.tipoSimbolo !== tipos.tipoNuevoValor){
38        valores[posicion] = valor
39        simbolo.valor = valores
40        _ambito.actualizar(id, simbolo)
41        return null
42    }
43
44    var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No es posible asignar un valor de tipo ${tipos.tipoNuevoValor} al vector '" +
45    _Error.addErrores(nuevo)
46    return "Error Semantico: No es posible asignar un valor de tipo '" + tipos.tipoNuevoValor + "' al vector '"

```

## ASIGNACION.JS

Este archivo contiene el proceso de la asignación de valores a las variables declaradas en el ámbito.

```

JS Asignacion.js X
backend > controller > Instruccion > JS Asignacion.js > Asignacion
1 const Operacion = require("../Operacion/Operacion");
2 const TIPO_DATO = require("../Enums/TipoDato");
3 const TIPO_ERROR = require("../Enums/Tipo_Error");
4 const ERRORES = require("../Ambito/S_Error");
5 function Asignacion(_instruccion, _ambito, _Error, Simbol){
6     const id = _instruccion.id;
7     const existe = _ambito.existeSimbolo(id)
8     if(existe){
9         var valor = Operacion(_instruccion.expresion, _ambito, _Error, "", Simbol)
10        var simbolo = _ambito.getSimbolo(id)
11        var tipos = {
12            tipoSimbolo: simbolo.tipo,
13            tipoNuevoValor: valor.tipo
14        }
15        if(tipos.tipoSimbolo===tipos.tipoNuevoValor){
16            simbolo.valor = valor.valor
17            _ambito.actualizar(id,simbolo)
18            return null
19        }
20        if((tipos.tipoSimbolo===TIPO_DATO.DECIMAL && tipos.tipoNuevoValor===TIPO_DATO.ENTERO) ||
21            (tipos.tipoSimbolo===TIPO_DATO.ENTERO && tipos.tipoNuevoValor===TIPO_DATO.DECIMAL) ||
22            (tipos.tipoSimbolo===TIPO_DATO.CARACTER && tipos.tipoNuevoValor===TIPO_DATO.ENTERO) ||
23            (tipos.tipoSimbolo===TIPO_DATO.CADENA && tipos.tipoNuevoValor===TIPO_DATO.CARACTER)
24        ){
25            simbolo.valor = valor.valor
26            _ambito.actualizar(id,simbolo)
27            return null
28        }
29        var nuevo=new ERRORES(TIPO_ERROR.SEMANTICO,'No es posible asignar un valor de tipo ${tipos.tipoNuevoValor} a la var
30        _Error.addErrores(nuevo)
31        return valor.valor+" Error Semantico: No es posible asignar un valor de tipo "+tipos.tipoNuevoValor+" a la variable
32    }
33    var nuevo=new ERRORES(TIPO_ERROR.SEMANTICO,'la variable '${String(id)}' no existe.',_instruccion.linea,_instruccion.co
34    _Error.addErrores(nuevo)
35    return "Error Semantico: la variable '${String(id)}' no existe... Linea: ${_instruccion.linea} Columna: ${_instruccion.
36 }
37
38 module.exports = Asignacion

```

## BLOQUE.JS

Este archivo contiene un ciclo que recorre todas las instrucciones que trae el árbol.

```

JS Bloque.js M X
backend > controller > Instruccion > JS Bloque.js > Bloque
10 const procesarSwitch = require("../Switch");
11 const Ciclofor = require("../for");
12 const casteo = require("../casteo");
13 const Asignacion_vector = require("../Asignacion_VEC");
14 const Asignacion_LISTA = require("../Asignacion_LISTA");
15
16 function Bloque(_instrucciones, _ambito, _Error, entorno, Simbol){
17     var cadena = ""
18     var haybreak=false;
19     var hayreturn=false;
20     var valorR=null;
21     var haycontinue=false;
22
23     _instrucciones.forEach(instruccion => {
24         //console.log("valuando..." +instruccion.tipo)
25         if(haybreak){ ...
26         }
27         if(haycontinue){ ...
28         }
29         if(hayreturn){ ...
30         }
31         if(instruccion.tipo === TIPO_INSTRUCCION.DECLARACION){ ...
32         }
33         else if(instruccion.tipo === TIPO_INSTRUCCION.PRINT){ ...
34         }
35         else if(instruccion.tipo === TIPO_INSTRUCCION.CASTEO){ ...
36         }
37         else if(instruccion.tipo === TIPO_INSTRUCCION.ASIGNACION){ ...
38         }
39         else if(instruccion.tipo === TIPO_INSTRUCCION.MODIFICAR_V || instruccion.tipo === TIPO_INSTRUCCION.MODIFICAR_L){ ...
40         }
41         else if(instruccion.tipo === TIPO_INSTRUCCION.AGREGAR_VAL_LISTA){ ...
42         }
43         else if(instruccion.tipo === TIPO_INSTRUCCION.WHILE){ // break ...
44         }
45         else if(instruccion.tipo === TIPO_INSTRUCCION.DO_WHILE){ ...
46         }
47         else if(instruccion.tipo === TIPO_INSTRUCCION.IF){ // break ...
48         }
49         else if(instruccion.tipo === TIPO_INSTRUCCION.ELSEIF){ ...
50         }
51         else if(instruccion.tipo === TIPO_INSTRUCCION.FOR){ ...
52         }
53         else if(instruccion.tipo === TIPO_INSTRUCCION.SWITCH){ ...
54         }
55         else if(instruccion.tipo===TIPO_OPERACION.TERNARIO){ ...
56         }
57     })
58 }
59
60 module.exports = Bloque

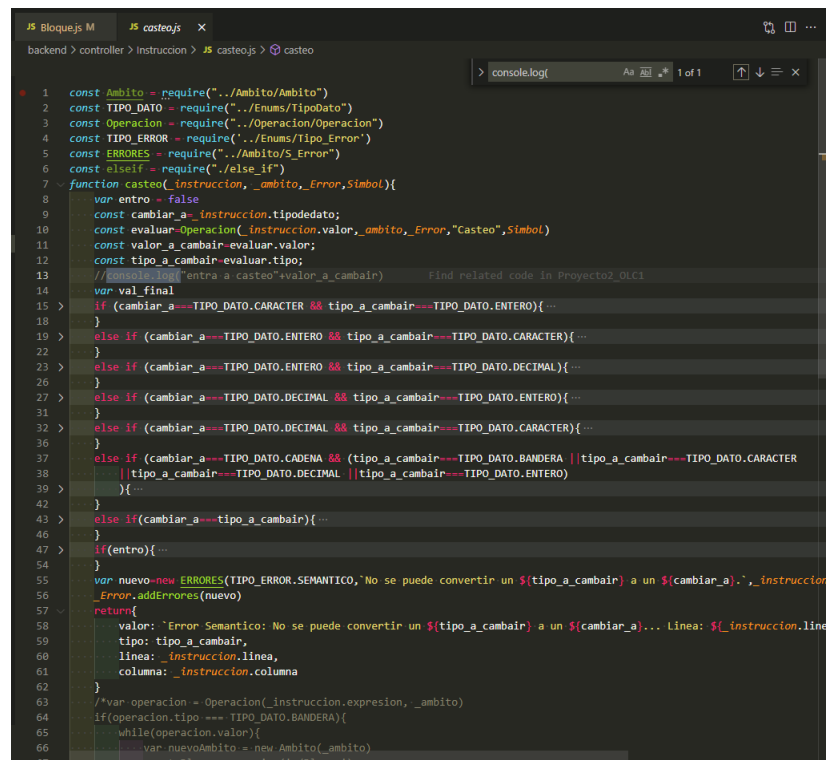
```

Y así al terminar el recorrido retornara una cadena y los valores del continue, return y break.

```
    }  
    return{  
        haybreak: haybreak,  
        cadena: cadena,  
        hayreturn: hayreturn,  
        retorno: valorR,  
        haycontinue: haycontinue  
    }  
}  
  
module.exports = Bloque
```

## CASTEO.JS

Este archivo contiene el proceso y restricciones del casteo.



```
1 const Ambito = require("../Ambito/Ambito")  
2 const TIPO_DATO = require("../Enums/TipoDato")  
3 const Operacion = require("../Operacion/Operacion")  
4 const TIPO_ERROR = require("../Enums/Tipo_Error")  
5 const ERRORES = require("../Ambito/S_Error")  
6 const elseif = require("../else_if")  
7 function casteo(_instruccion, _ambito, _error, _simbol){  
8     var entro = false  
9     const cambiar_a = _instruccion.tipodedato;  
10    const evaluar = Operacion(_instruccion.valor, _ambito, _error, "Casteo", _simbol)  
11    const valor_a_cambiar = evaluar.valor;  
12    const tipo_a_cambiar = evaluar.tipo;  
13    // console.log("entra a casteo="+valor_a_cambiar) Find related code in Proyecto2_OlC1  
14    var val final  
15    if (cambiar_a===TIPO_DATO.CARACTER && tipo_a_cambiar===TIPO_DATO.ENTERO){ ...  
16    }  
17    else if (cambiar_a===TIPO_DATO.ENTERO && tipo_a_cambiar===TIPO_DATO.CARACTER){ ...  
18    }  
19    else if (cambiar_a===TIPO_DATO.ENTERO && tipo_a_cambiar===TIPO_DATO.DECIMAL){ ...  
20    }  
21    else if (cambiar_a===TIPO_DATO.DECIMAL && tipo_a_cambiar===TIPO_DATO.ENTERO){ ...  
22    }  
23    else if (cambiar_a===TIPO_DATO.DECIMAL && tipo_a_cambiar===TIPO_DATO.CARACTER){ ...  
24    }  
25    else if (cambiar_a===TIPO_DATO.CADENA && (tipo_a_cambiar===TIPO_DATO.BANDERA || tipo_a_cambiar===TIPO_DATO.CARACTER  
26    || tipo_a_cambiar===TIPO_DATO.DECIMAL || tipo_a_cambiar===TIPO_DATO.ENTERO)  
27    ){ ...  
28    }  
29    else if (cambiar_a===tipo_a_cambiar){ ...  
30    }  
31    if(entro){ ...  
32    }  
33    var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No se puede convertir un $(tipo_a_cambiar) a un $(cambiar_a)... Linea: $_instruccion.linea  
34    _error.addErrores(nuevo)  
35    return{  
36        valor: "Error Semantico: No se puede convertir un $(tipo_a_cambiar) a un $(cambiar_a)... Linea: $_instruccion.linea  
37        tipo: tipo_a_cambiar,  
38        linea: _instruccion.linea,  
39        columna: _instruccion.columna  
40    }  
41    }  
42    }  
43    if(operacion.tipo === TIPO_DATO.BANDERA){  
44        while(operacion.valor){  
45            var nuevoAmbito = new Ambito(_ambito)  
46        }  
47    }  
48    }  
49    }  
50    }  
51    }  
52    }  
53    }  
54    }  
55    }  
56    }  
57    }  
58    }  
59    }  
60    }  
61    }  
62    }  
63    }  
64    }  
65    }  
66    }  
67    }  
68    }  
69    }  
70    }  
71    }  
72    }  
73    }  
74    }  
75    }  
76    }  
77    }  
78    }  
79    }  
80    }  
81    }  
82    }  
83    }  
84    }  
85    }  
86    }  
87    }  
88    }  
89    }  
90    }  
91    }  
92    }  
93    }  
94    }  
95    }  
96    }  
97    }  
98    }  
99    }  
100   }
```

## DECMETODO.JS

Este archivo contiene el proceso de la declaración de métodos y funciones.

```
JS Declaracion_Metodo.js X
backent > controller > Instruccion > JS Declaracion_Metodo.js > DecMetodo

> console.log( Aa Abi .* No results ↑ ↓ ≡ X

1  const Metodo = require("../Ambito/Metodo")
2  const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion")
3  const TIPO_ERROR = require("../Enums/Tipo_Error")
4  const ERRORES = require("../Ambito/S_Error")
5  function DecMetodo(_instruccion, _ambito, _Error, _entorno, Simbol){
6      var nuevoMetodo ;
7      if (_instruccion.tipo === TIPO_INSTRUCCION.DECLARACION_M){
8          nuevoMetodo = new Metodo(_instruccion.nombre, _instruccion.lista_parametros, _entorno, _instruccion.instrucciones, "vo
9      }else{
10         nuevoMetodo = new Metodo(_instruccion.nombre, _instruccion.lista_parametros, _entorno, _instruccion.instrucciones, _in
11     }
12     //verificamos si el nombre ya existe como simbolo
13     if(_ambito.existeSimbolo(nuevoMetodo.id)!=false){
14         var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, `No se puede declarar un metodo con el mismo nombre de una variable '${(
15         _Error.addErrores(nuevo)
16         return `Error Semantico: No se puede declarar un metodo con el mismo nombre de una variable '${nuevoMetodo.id}'...
17     }
18     //verificamos si el metodo ya existe
19     else if(_ambito.existeMetodo(nuevoMetodo.id)!=false){
20         var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, `El método '${nuevoMetodo.id}' ya existe...`, nuevoMetodo.linea, nuevoMet
21         _Error.addErrores(nuevo) Find related code in Proyecto2_OLC1
22         return `Error Semantico: El método '${nuevoMetodo.id}' ya existe... Linea: ${nuevoMetodo.linea} Columna: ${nuevoMet
23     }
24     //de lo contrario vamos a guardarlo
25     _ambito.addMetodo(nuevoMetodo.id, nuevoMetodo)
26     Simbol.add_s(nuevoMetodo)
27     return null
28 }
29 module.exports = DecMetodo
```

## DECLARACION.JS

Este archivo contiene el proceso de la declaración de variables, listas y vectores, con y sin valor.

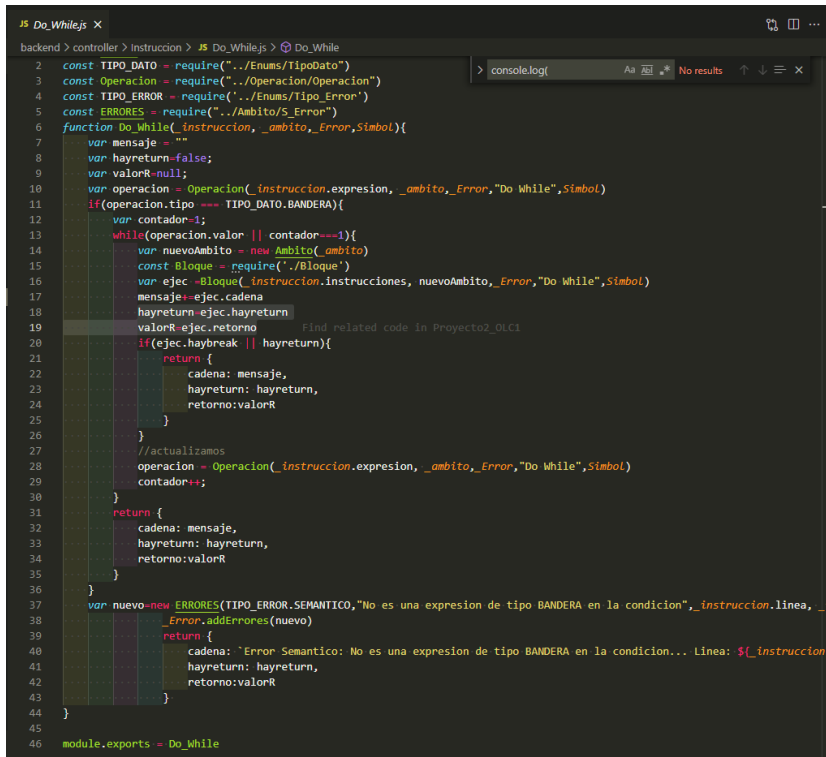
```
JS Declaracion.js X
backent > controller > Instruccion > JS Declaracion.js > [0] <unknown>

> console.log( Aa Abi .* ? of 20 ↑ ↓ ≡ X

1  const Simbolo = require("../Ambito/Simbolo");
2  const TIPO_DATO = require("../Enums/TipoDato");
3  const Operacion = require("../Operacion/Operacion");
4  const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion")
5  const casteo = require("../casteo");
6  const TIPO_ERROR = require("../Enums/Tipo_Error")
7  const ERRORES = require("../Ambito/S_Error")
8  const INSTRUCCION = require("../Instruccion");
9  const TIPO_OPERACION = require("../Enums/TipoOperacion")
10 function Declaracion(_instruccion, _ambito, _Error, _entorno, Simbol){
11     //console.log(_instruccion)
12 > if(_instruccion.tipo_dato === TIPO_DATO.VECTOR){ ...
13     }
14 > else if(_instruccion.tipo_dato === TIPO_DATO.LISTA){ ...
15     }
16 > else if(_instruccion.tipo_dato === TIPO_DATO.DECIMAL){ ...
17     }
18 > else if(_instruccion.tipo_dato === TIPO_DATO.CADENA){ ...
19     }
20 > else if(_instruccion.tipo_dato === TIPO_DATO.BANDERA){ ...
21     }
22 > else if(_instruccion.tipo_dato === TIPO_DATO.ENTERO){ ...
23     }
24 > else if(_instruccion.tipo_dato === TIPO_DATO.CARACTER){ ...
25     }
26 }
27 module.exports = Declaracion Find related code in Proyecto2_OLC1
```

## DO\_WHILE.JS

Este archivo contiene el proceso del ciclo do while, teniendo como restricción que mínimo una vez se debe ejecutar el cuerpo.



```
2 const TIPO_DATO = require("../Enums/TipoDato")
3 const Operacion = require("../Operacion/Operacion")
4 const TIPO_ERROR = require("../Enums/TipoError")
5 const ERRORES = require("../Ambito/S_Error")
6 function Do_While(instruccion, _ambito, _Error, Simbol){
7   var mensaje = ""
8   var hayreturn=false;
9   var valorR=null;
10  var operacion = Operacion(_instruccion.expresion, _ambito, _Error, "Do While", Simbol)
11  if(operacion.tipo === TIPO_DATO.BANDERA){
12    var contador=1;
13    while(operacion.valor || contador!==1){
14      var nuevoAmbito = new Ambito(_ambito)
15      const Bloque = require("../Bloque")
16      var ejec =Bloque(_instruccion.instrucciones, nuevoAmbito, _Error, "Do While", Simbol)
17      mensaje+=ejec.cadena
18      hayreturn =ejec.hayreturn
19      valorR =ejec.retorno
20      if(ejec.haybreak || hayreturn){
21        return {
22          cadena: mensaje,
23          hayreturn: hayreturn,
24          retorno:valorR
25        }
26      }
27      //actualizamos
28      operacion = Operacion(_instruccion.expresion, _ambito, _Error, "Do While", Simbol)
29      contador++;
30    }
31    return {
32      cadena: mensaje,
33      hayreturn: hayreturn,
34      retorno:valorR
35    }
36  }
37  var nuevo=new ERRORES(TIPO_ERROR.SEMANTICO, "No es una expresion de tipo BANDERA en la condicion", _instruccion.linea, _i
38  _Error.addErrores(nuevo)
39  return {
40    cadena: "Error Semantico: No es una expresion de tipo BANDERA en la condicion... Linea: ${_instruccion.
41    hayreturn: hayreturn,
42    retorno:valorR
43  }
44 }
45
46 module.exports = Do_While
```

## ELSE\_IF.JS

Este archivo contiene el proceso de la sentencia de control else if. Que contiene un if y una lista de else if y else... que tiene como condición que al entrar una sentencia ya no podrá entrar a otra.

```
JS else_if.js x
backend > controller > Instruccion > JS else_if.js > elseif

1 const Operacion = require("../Operacion/Operacion")
2 const Ambito = require("../Ambito/Ambito")
3 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
4 const TIPO_DATO = require("../Enums/TipoDato")
5 const TIPO_ERROR = require("../Enums/Tipo_Error")
6 const ERRORES = require("../Ambito/S_Error")
7 function elseif (instruccion, _ambito, _Error, Simbol) {
8     var mensaje=""
9     var haybreak=false;
10    var hayreturn=false;
11    var haycontinue=false;
12    var valorR=null;
13    var yaentro=false;
14    const primerif = Operacion(instruccion.expresion, _ambito, _Error, "Else If / Else", Simbol);
15    const nuevoAmbito = new Ambito(_ambito);
16    //console.log(primerif.tipo+"---"+TIPO_DATO.BANDERA)
17 > if(primerif.tipo=== TIPO_DATO.BANDERA){ Find related code in Proyecto2_OLC1
63 > }else{ ...
67 > }
68 > return {
69 >     haybreak: haybreak,
70 >     cadena: mensaje,
71 >     hayreturn: hayreturn,
72 >     retorno:valorR,
73 >     haycontinue:haycontinue
74 > }
75 > }
76 > }
77 module.exports = elseif
```

## EXEC\_LLAMADA.JS

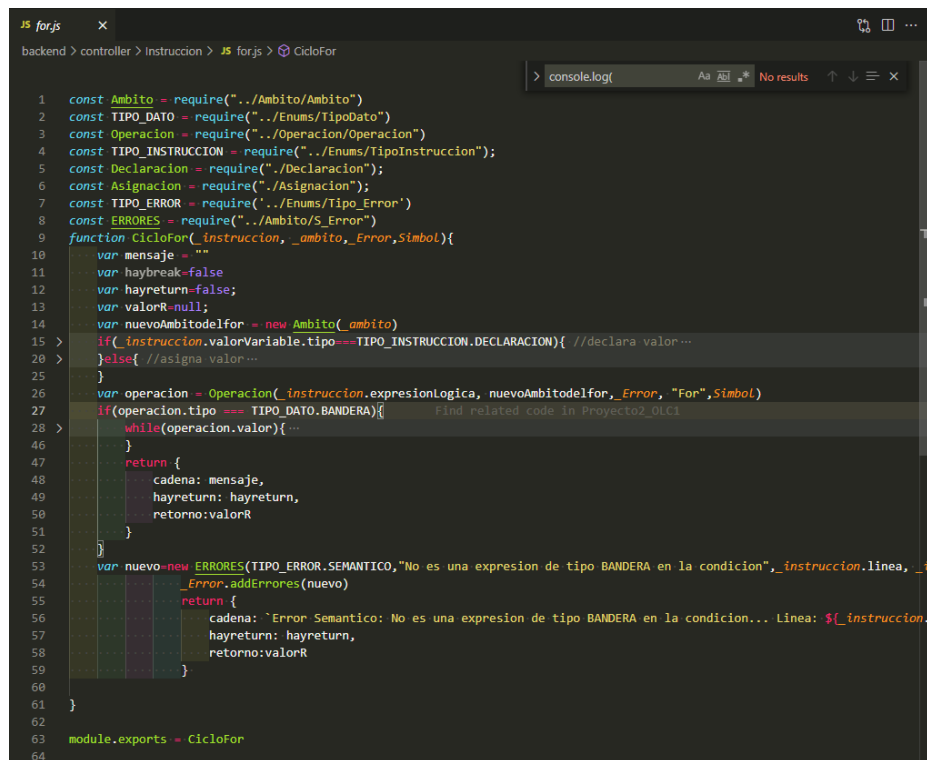
Este archivo contiene el proceso de la llamada de un método o función, en donde se declaran los parámetros y sus valores por medio de listas y verificaciones.

```
JS Exec_Llamada.js x
backend > controller > Instruccion > JS Exec_Llamada.js > Exec_Llamada

7 const ERRORES = require("../Ambito/S_Error")
8
9 function Exec_Llamada( instruccion, _ambito, _Error, _entorno, Simbol){
10     var hayreturn=false;
11     var valorR=null;
12     var metodoEjecutar = _ambito.getMetodo( instruccion.nombre)
13     if(metodoEjecutar!=null){ Find related code in Proyecto2_OLC1
14         var nuevoAmbito = new Ambito(_ambito)
15         let parametrosdemetodo_tipo=[]
16         let parametrosdemetodo_variable=[]
17         let valoresquemandan_tipo=[]
18         let valoresquemandan_valor=[]
19 > if( instruccion.lista_valores!=null){ ...
48 > }
49 > if( instruccion.lista_valores!=null){ ...
58 > }
59 >
60 > if(parametrosdemetodo_tipo.length===valoresquemandan_tipo.length){ ...
84 > }else{ ...
93 > }
94 > var ejec= Bloque(metodoEjecutar.instrucciones, nuevoAmbito, _Error, metodoEjecutar.id, Simbol)
95 > //console.log(ejec)
96 > var mensaje=ejec.cadena
97 > if(ejec.haybreak){ ...
101 > }
102 > if(ejec.haycontinue){ ...
106 > }
107 > hayreturn=ejec.hayreturn
108 > valorR=ejec.retorno
109 > var op = valorR;
110 > if(metodoEjecutar.soy==="Metodo"){ ...
120 > }else{ ...
141 > }
142 > return {
143 >     cadena: mensaje,
144 >     hayreturn: hayreturn,
145 >     retorno:valorR
146 > }
147 >
148 > var nuevo=new ERRORES(TIPO_ERROR.SEMANTICO, "El método "+instruccion.nombre+" no existe.", _instruccion.linea, _instruccion.columna);
149 > _Error.addErrores(nuevo)
150 > return {
151 >     cadena: "Error Semantico: El método ${_instruccion.nombre} no existe... Línea: ${_instruccion.linea} Columna: ${_instruccion.columna}",
152 >     hayreturn: hayreturn,
153 >     retorno:valorR
154 > }
155 > }
```

## FOR.JS

Este archivo contiene el proceso del ciclo for, se hace desde un while y se va actualizando la condición, y hasta que se deja de cumplir la condición sale y retorna los valores necesarios.



```
1 const Ambito = require("../Ambito/Ambito")
2 const TIPO_DATO = require("../Enums/TipoDato")
3 const Operacion = require("../Operacion/Operacion")
4 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
5 const Declaracion = require("../Declaracion");
6 const Asignacion = require("../Asignacion");
7 const TIPO_ERROR = require("../Enums/Tipo_Error")
8 const ERRORES = require("../Ambito/S_Error")
9 function CicloFor( instruccion, _ambito, _error, Simbol){
10     var mensaje = ""
11     var haybreak=false;
12     var hayreturn=false;
13     var valorR=null;
14     var nuevoAmbitodelfor = new Ambito(_ambito)
15     if(_instruccion.valorVariable.tipo==TIPO_INSTRUCCION.DECLARACION){ //declara valor...
16     }else{ //asigna valor...
17     }
18     var operacion = Operacion(_instruccion.expresionLogica, nuevoAmbitodelfor, _error, "For", Simbol)
19     if(operacion.tipo == TIPO_DATO.BANDERA){
20         while(operacion.valor){
21         }
22         return {
23             cadena: mensaje,
24             hayreturn: hayreturn,
25             retorno:valorR
26         }
27     }
28     var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No es una expresion de tipo BANDERA en la condicion", _instruccion.linea, _i
29     _error.addErrores(nuevo)
30     return {
31         cadena: `Error Semantico: No es una expresion de tipo BANDERA en la condicion... Linea: ${_instruccion.
32         hayreturn: hayreturn,
33         retorno:valorR
34     }
35 }
36 module.exports = CicloFor
```

## GENERAR\_GRAF.JS

Este archivo contiene el proceso de generar el dot, convertirlo a pdf y al final abrir el pdf en el navegador.

```
JS Generar_Grafjs X
backend > controller > Instruccion > JS Generar_Grafjs > Generar_Graf > fs.writeFile("ArbolAST.dot") callback
> console.log(

1
2
3 function Generar_Graf(contenido){
4   var fs = require('fs');
5
6   fs.writeFile("ArbolAST.dot", contenido, function(err) {
7     if (err) {
8       return console.log(err);
9     }
10    var exec = require('child_process').exec, child;
11
12    child = exec('dot -Tpdf ArbolAST.dot -o ArbolAST.pdf',
13      function (error, stdout, stderr) {
14        console.log('stdout: ' + stdout);
15        console.log('stderr: ' + stderr);
16        if (error !== null) {
17          console.log('exec error: ' + error);
18        }
19      });
20    const { spawn } = require('child_process');
21    const bat = spawn('cmd.exe', ['/c', 'ArbolAST.pdf']);
22
23    bat.stdout.on('data', (data) => {
24      console.log(data.toString());
25    });
26
27    bat.stderr.on('data', (data) => {
28      console.error(data.toString());
29    });
30
31    bat.on('exit', (code) => {
32      console.log("Child exited with code " + code);
33    });
34    return console.log("El archivo fue creado correctamente");
35  });
36 }
37
38 module.exports = Generar_Graf
```

## GLOBAL.JS

Este archivo contiene el proceso del verificar la cantidad de funciones exec, declarar variables globales y metodos y por ultimo mandar a llamar la clase de llamadas para ejecutar el método del exec.



```
JS Global.js x
backend > controller > Instruccion > JS Global.js > Global
10 const Asignacion_LISTA = require("../Asignacion_LISTA");
11
12 function Global(_instrucciones, _ambito, _error, _entorno, Simbol){
13     var cadena = ""
14     //1ERA: VERIFICAR DE QUE SOLO VENGA 1 EXEC
15     var contadorExec=0;
16     for(let i=0; i<_instrucciones.length; i++){...
20     }
21     if(contadorExec==0){...
26     }
27     else if(contadorExec>1){...
31     }
32     //2DA: DECLARAR VARIABLES, METODOS Y ASIGNAR VALORES
33     for(let i=0; i<_instrucciones.length; i++){
34
35         if(_instrucciones[i].tipo === TIPO_INSTRUCCION.DECLARACION){
36             var mensaje = Declaracion(_instrucciones[i], _ambito, _error, _entorno, Simbol)
37             if(mensaje!=null){...
41             }
42         }
43         else if(_instrucciones[i].tipo === TIPO_INSTRUCCION.ASIGNACION){
44             var mensaje = Asignacion(_instrucciones[i], _ambito, _error, Simbol)
45             if(mensaje!=null){...
49             }
50         }
51         else if(_instrucciones[i].tipo === TIPO_INSTRUCCION.MODIFICAR_V || _instrucciones[i].tipo === TIPO_INSTRUCCION.MODIFICAR_L){
58         }
59         else if(_instrucciones[i].tipo === TIPO_INSTRUCCION.AGREGAR_VAL_LISTA){...
66         }
67         else if(_instrucciones[i].tipo === TIPO_INSTRUCCION.DECLARACION_M || _instrucciones[i].tipo === TIPO_INSTRUCCION.DECLARACION_F){
74         }
75     }
76 }
77 //console.log(_ambito) Find related code in Proyecto2_OLC1
78 for(let i=0; i<_instrucciones.length; i++){
79     if(_instrucciones[i].tipo === TIPO_INSTRUCCION.EXEC){...
93     }
94 }
95 //3ERA. PASADA. VAMOS A BUSCAR EL EXEC QUE VAMOS EJECUTAR
96 return cadena
97 }
98
99 module.exports = Global
```

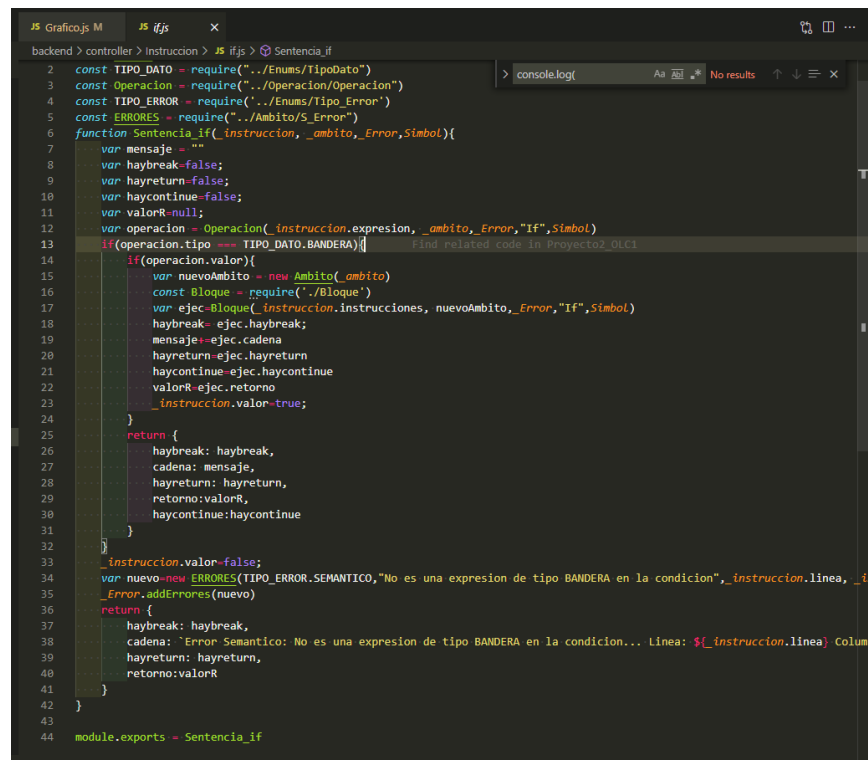
## GRAFICO.JS

Este archivo contiene el proceso de la creación del dot para el árbol AST. Lo hace en base al árbol que genera el analizador jison.

```
JS Grafico.js M x
backend > controller > Instruccion > JS Grafico.js > GDot > cadena
1 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
2 const TIPO_VALOR = require("../Enums/TipoValor");
3 const TIPO_OPERACION = require("../Enums/TipoOperacion");
4
5 function GDot(instruccion, padre, label_padre){
6     var cadena = ""
7     //Find related code in Proyecto2_OLC1
8     if(instruccion!=null){
9         if(instruccion.tipo === TIPO_INSTRUCCION.DECLARACION_M || instruccion.tipo === TIPO_INSTRUCCION.DECLARACION_F){...
32         }
33     }
34     else if(instruccion.tipo === TIPO_INSTRUCCION.EXEC || instruccion.tipo === TIPO_INSTRUCCION.LLAMADA){...
47     }
48 }
49 else if(instruccion.tipo === TIPO_INSTRUCCION.DECLARACION){...
73     }
74 }
75 else if(instruccion.tipo === TIPO_INSTRUCCION.PRINT){...
84     }
85 }
86 else if(instruccion.tipo === TIPO_INSTRUCCION.RETURN || instruccion.tipo === TIPO_INSTRUCCION.CONTINUE || instruccion.tipo === TIPO_INSTRUCCION.BREAK){...
97     }
98 }
99 else if(instruccion.tipo === TIPO_INSTRUCCION.ACCESO_V || instruccion.tipo === TIPO_INSTRUCCION.ACCESO_L){...
111     }
112 }
113 else if(instruccion.tipo === TIPO_INSTRUCCION.MODIFICAR_V || instruccion.tipo === TIPO_INSTRUCCION.MODIFICAR_L || instruccion.tipo === TIPO_INSTRUCCION.AGREGAR_VAL_LISTA){...
132     }
133 }
134 else if(instruccion.tipo === TIPO_INSTRUCCION.CASTEO){...
146     }
147 }
148 else if(instruccion.tipo === TIPO_INSTRUCCION.ASIGNACION){...
162     }
163 }
164 else if(instruccion.tipo === TIPO_INSTRUCCION.WHILE){...
178     }
179 }
180 else if(instruccion.tipo === TIPO_INSTRUCCION.DO_WHILE){...
194     }
195 }
196 else if(instruccion.tipo === TIPO_INSTRUCCION.IF){...
210     }
211 }
```

## IF.JS

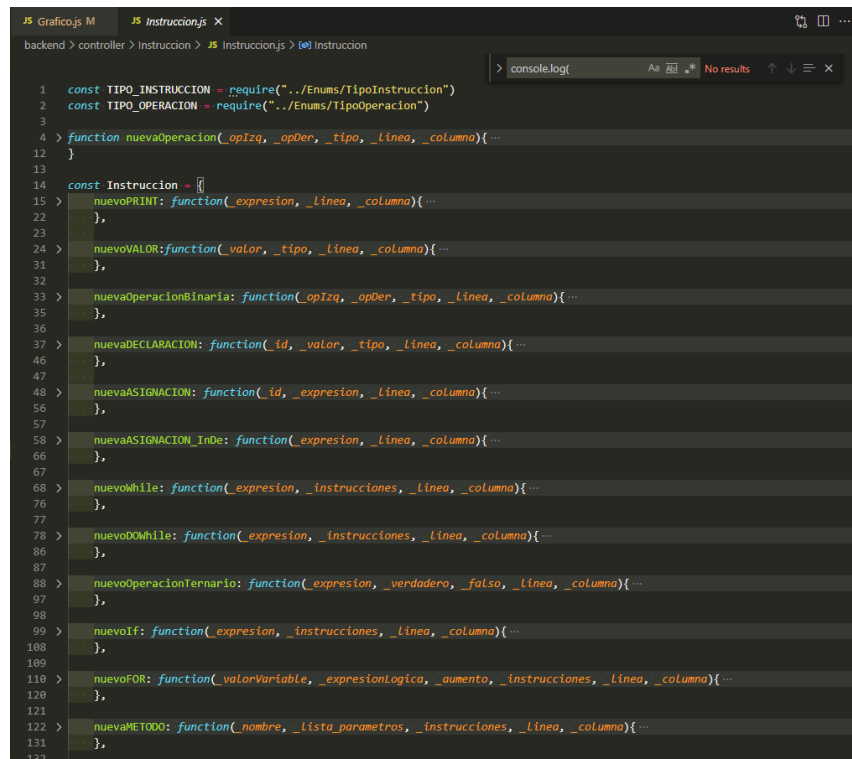
Este archivo contiene el proceso de la sentencia de control if que valida si la condición es verdadera, entrara a ejecutar las instrucciones contenidas, de lo contrario no retornara nada.



```
2 const TIPO_DATO = require("../Enums/TipoDato")
3 const Operacion = require("../Operacion/Operacion")
4 const TIPO_ERROR = require("../Enums/Tipo_Error")
5 const ERRORES = require("../Ambito/S_Error")
6 function Sentencia_if(_instruccion, _ambito, _Error, Simbol){
7   var mensaje = ""
8   var haybreak=false;
9   var hayreturn=false;
10  var haycontinue=false;
11  var valorR=null;
12  var operacion = Operacion(_instruccion.expresion, _ambito, _Error, "If", Simbol)
13  if(operacion.tipo === TIPO_DATO.BANDERA){
14    if(operacion.valor){
15      var nuevoAmbito = new Ambito(_ambito)
16      const Bloque = require("../Bloque")
17      var ejecBloque = _instruccion.Instrucciones, nuevoAmbito, _Error, "If", Simbol)
18      haybreak = ejecBloque.haybreak;
19      mensaje = ejecBloque.cadena;
20      hayreturn = ejecBloque.hayreturn;
21      haycontinue = ejecBloque.haycontinue;
22      valorR = ejecBloque.retorno;
23      _instruccion.valor=true;
24    }
25    return {
26      haybreak: haybreak,
27      cadena: mensaje,
28      hayreturn: hayreturn,
29      retorno:valorR,
30      haycontinue:haycontinue
31    }
32  }
33  _instruccion.valor=false;
34  var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No es una expresion de tipo BANDERA en la condicion", _instruccion.linea, _t
35  _Error.addErrores(nuevo)
36  return {
37    haybreak: haybreak,
38    cadena: "Error Semantico: No es una expresion de tipo BANDERA en la condicion... Linea: ${_instruccion.linea} Colum
39    hayreturn: hayreturn,
40    retorno:valorR
41  }
42 }
43
44 module.exports = Sentencia_if
```

## INSTRUCCIÓN.JS

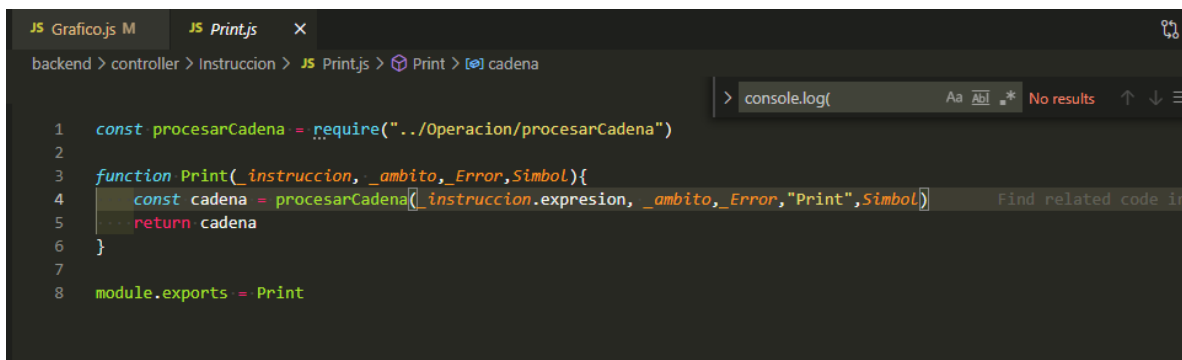
Este archivo contiene todas las estructuras que traerá cada producción del analizador jison.



```
1 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion")
2 const TIPO_OPERACION = require("../Enums/TipoOperacion")
3
4 > function nuevaOperacion(_opIzq, _opDer, _tipo, _linea, _columna){...
12 }
13
14 const Instruccion = []
15 > nuevoPRINT: function(_expresion, _linea, _columna){...
22 },
23
24 > nuevoVALOR: function(_valor, _tipo, _linea, _columna){...
31 },
32
33 > nuevaOperacionBinaria: function(_opIzq, _opDer, _tipo, _linea, _columna){...
35 },
36
37 > nuevaDECLARACION: function(_id, _valor, _tipo, _linea, _columna){...
46 },
47
48 > nuevaASIGNACION: function(_id, _expresion, _linea, _columna){...
56 },
57
58 > nuevaASIGNACION_InDe: function(_expresion, _linea, _columna){...
66 },
67
68 > nuevoWhile: function(_expresion, _instrucciones, _linea, _columna){...
76 },
77
78 > nuevoDOWhile: function(_expresion, _instrucciones, _linea, _columna){...
86 },
87
88 > nuevaOperacionTernario: function(_expresion, _verdadero, _falso, _linea, _columna){...
97 },
98
99 > nuevoIf: function(_expresion, _instrucciones, _linea, _columna){...
108 },
109
110 > nuevoFOR: function(_valorVariable, _expresionLogica, _aumento, _instrucciones, _linea, _columna){...
120 },
121
122 > nuevaMETODO: function(_nombre, _lista_parametros, _instrucciones, _linea, _columna){...
131 },
132
```

## PRINT.JS

Este archivo contiene el proceso del print en donde busca el valor y el tipo para poder imprimirlo en consola.



```
1 const procesarCadena = require("../Operacion/procesarCadena")
2
3 function Print( instruccion, _ambito, _Error, _Simbol){
4   const cadena = procesarCadena([instruccion.expresion, _ambito, _Error, "Print", _Simbol])
5   return cadena
6 }
7
8 module.exports = Print
```

## SWITCH.JS

Este archivo contiene el proceso de la sentencia del switch se evaluarán las condiciones y si llega a entrar alguno con un break adentro ya no ejecutará el default.

```

1  const Operacion = require("../Operacion/Operacion")
2  const Ambito = require("../Ambito/Ambito")
3  const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
4  const TIPO_ERROR = require("../Enums/Tipo_Error")
5  const ERRORES = require("../Ambito/S_Error")
6  function pSwitch(instruccion, _ambito, _error, Simbol) {
7      var mensaje = ""
8      var haybreak = false;
9      var hayreturn = false;
10     var valorR = null;
11     var haycontinue = false;
12     const valorExpCase = Operacion(instruccion.expresion, _ambito, _error, "Switch-Case", Simbol);
13     const nuevoAmbito = new Ambito(_ambito);
14     instruccion.casos.forEach(caso => {
15         const Bloque = require("../Bloque")
16         if (caso.tipo == TIPO_INSTRUCCION.SWITCH_CASO){
17             const valorExpCase = Operacion(caso.expresion, nuevoAmbito, _error, "Switch-Case", Simbol);
18             //console.log("CASO: ")
19             //console.log(valorExpCase.valor)
20             //console.log(valorExpresion.valor)
21             if (valorExpCase.valor == valorExpresion.valor && !haybreak){
22                 //console.log("CASO: "+mensaje);
23             }
24         }
25         else if (caso.tipo == TIPO_INSTRUCCION.SWITCH_DEFECTO){
26             if (!haybreak){
27                 var ejec = Bloque(caso.instrucciones, nuevoAmbito, _error, "Switch-Case", Simbol);
28                 mensaje += ejec.cadena
29                 haybreak = ejec.haybreak
30                 hayreturn = ejec.hayreturn
31                 valorR = ejec.retorno
32             }
33             //console.log("CASO: "+mensaje);
34         }
35     });
36     if (hayreturn){
37         return {
38             cadena: mensaje,
39             hayreturn: hayreturn,
40             retorno: valorR
41         }
42     }
43 }
44 module.exports = pSwitch

```

## WHILE.JS

Este archivo contiene el proceso del ciclo while, se hace desde un while y se va actualizando la condición, y hasta que se deja de cumplir la condición sale y retorna los valores necesarios.

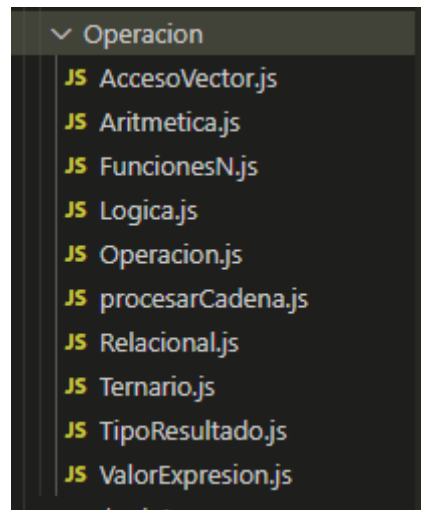
```

1  const Ambito = require("../Ambito/Ambito")
2  const TIPO_DATO = require("../Enums/TipoDato")
3  const Operacion = require("../Operacion/Operacion")
4  const TIPO_ERROR = require("../Enums/Tipo_Error")
5  const ERRORES = require("../Ambito/S_Error")
6  function CicloWhile(instruccion, _ambito, _error, Simbol){
7      var mensaje = ""
8      var hayreturn = false;
9      var valorR = null;
10     var operacion = Operacion(instruccion.expresion, _ambito, _error, "While", Simbol)
11     if(operacion.tipo == TIPO_DATO.BANDERA){
12         while(operacion.valor){
13             var nuevoAmbito = new Ambito(_ambito)
14             const Bloque = require("../Bloque")
15             var ejec = Bloque(instruccion.instrucciones, nuevoAmbito, _error, "While", Simbol)
16             mensaje += ejec.cadena
17             hayreturn = ejec.hayreturn
18             valorR = ejec.retorno
19             if(ejec.haybreak || hayreturn || ejec.haycontinue){
20                 return {
21                     cadena: mensaje,
22                     hayreturn: hayreturn,
23                     retorno: valorR
24                 }
25             }
26             //actualizamos
27             operacion = Operacion(instruccion.expresion, _ambito, _error, "While", Simbol)
28         }
29         return {
30             cadena: mensaje,
31             hayreturn: hayreturn,
32             retorno: valorR
33         }
34     }
35     var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "No es una expresion de tipo BANDERA en la condicion", instruccion.linea, instruccion.columna)
36     _error.addErrores(nuevo)
37     return {
38         cadena: "Error Semantico: No es una expresion de tipo BANDERA en la condicion... Linea: ${instruccion.linea}, Columna: ${instruccion.columna}"
39         hayreturn: hayreturn,
40         retorno: valorR
41     }
42 }
43 module.exports = CicloWhile

```

## OPERACIÓN

Esta carpeta contiene todas las operaciones y el acceso a todos los valores y tipos.



## ACCESOVECTOR.JS

Este archivo contiene el proceso del acceso de listas y vectores en una posición.

```
JS AccesoVector.js X
backend > controller > Operacion > JS AccesoVector.js > AccesoVector

1  const Simbolo = require("../Ambito/Simbolo");
2  const TIPO_DATO = require("../Enums/TipoDato");
3  const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
4  const TIPO_ERROR = require("../Enums/Tipo_Error");
5  const ERRORES = require("../Ambito/S_Error");
6  const Operacion = require("../Operacion/Operacion");
7  function AccesoVector(_instruccion, _ambito, _Error, _entorno, Simbol){
8      var valor = null;
9      var valores = [];
10     var posicion = Operacion(_instruccion.posicion, _ambito, _Error, _entorno, Simbol).valor;
11     var posi = Operacion(_instruccion.posicion, _ambito, _Error, _entorno, Simbol);
12     //console.log(posicion);
13     const id = _instruccion.id;
14     const existe = _ambito.existeSimbolo(id);
15     if(existe){
16         //var valor = Operacion(_instruccion.expresion, _ambito, _Error, "", Simbol);
17         var simbolo = _ambito.getSimbolo(id);
18         valores = simbolo.valor;
19         var tam = valores.length;
20         if(posicion < tam && posicion >= 0){
21             if(posi.tipo === TIPO_DATO.ENTERO){
22                 valor = valores[posicion];
23                 //console.log("valor: "+valor);
24                 //const Operacion = require("../Operacion");
25                 valor = Operacion(valor, _ambito, _Error, _entorno, Simbol);
26             } else { ... }
27         } else { ... }
28     } else { ... }
29     //console.log(valor.valor+"--->"+valor.tipo);
30     return {
31         valor: valor.valor,
32         tipo: valor.tipo,
33         linea: _instruccion.linea,
34         columna: _instruccion.columna
35     };
36 }
37
38 var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "el vector '${String(id)}' no existe.", _instruccion.linea, _instruccion.columna);
39 _Error.addErrores(nuevo);
40 return {
41     valor: "Error Semantico: el vector '${String(id)}' no existe... Linea: ${_instruccion.linea} Columna: ${_instruccion.columna}",
42     tipo: TIPO_DATO.CADENA,
43     linea: _instruccion.linea,
44     columna: _instruccion.columna
45 };
46 }
```

## ARITMETICA.JS

Este archivo contiene el proceso de las operaciones aritméticas, como lo son suma, resta, multiplicación, etc.

```
JS Arimetica.js M X
backend > controller > Operacion > JS Arimetica.js > Arimetica

1 const TIPO_DATO = require("../Enums/TipoDato")
2 const TIPO_OPERACION = require("../Enums/TipoOperacion")
3 const TIPO_VALOR = require("../Enums/TipoValor")
4 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
5 const TipoResultado = require("../TipoResultado")
6 const ValorExpresion = require("../ValorExpresion")
7 const TIPO_ERROR = require("../Enums/Tipo_Error")
8 const ERRORES = require("../Ambito/S_Error")
9
10
11 function Arimetica( expresion, _ambito, _Error, _entorno, Simbol){
12     if(_expresion.tipo === TIPO_VALOR.DECIMAL || _expresion.tipo === TIPO_VALOR.BANDERA ||
13        _expresion.tipo === TIPO_VALOR.CADENA || _expresion.tipo === TIPO_VALOR.IDENTIFICADOR
14        || _expresion.tipo === TIPO_VALOR.ENTERO || _expresion.tipo === TIPO_VALOR.CARACTER
15    ){
16        return ValorExpresion( expresion, _ambito, _Error)
17    }
18    else if(_expresion.tipo === TIPO_OPERACION.SUMA){
19        return suma( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
20    }
21    else if(_expresion.tipo === TIPO_OPERACION.RESTA){
22        return resta( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
23    }
24    else if(_expresion.tipo === TIPO_OPERACION.MULTIPLICACION){
25        return multiplicacion( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
26    }
27    else if(_expresion.tipo === TIPO_OPERACION.DIVISION){
28        return division( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
29    }
30    else if(_expresion.tipo === TIPO_OPERACION.POTENCIA){
31        return potencia( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
32    }
33    else if(_expresion.tipo === TIPO_OPERACION.MODULO){
34        return modulo( expresion.opIzq, _expresion.opDer, _ambito, _Error, _entorno, Simbol)
35    }
36    else if(_expresion.tipo === TIPO_OPERACION.NEGACION){
37        return negacion( expresion.opIzq, _ambito, _Error, _entorno, Simbol)
38    }
39    else{
40        const Operacion = require("../Operacion")
41        return Operacion( expresion, _ambito, _Error, _entorno, Simbol)
42    }
43 }
44
```

## FUNCIONESN.JS

Este archivo contiene el proceso de las funciones nativas y especiales.

```
JS Arimetica.js M JS FuncionesN.js X
backend > controller > Operacion > JS FuncionesN.js > <unknown>

1 const TIPO_DATO = require("../Enums/TipoDato")
2 const TIPO_OPERACION = require("../Enums/TipoOperacion")
3 const TIPO_VALOR = require("../Enums/TipoValor")
4 const ValorExpresion = require("../ValorExpresion")
5 const TIPO_ERROR = require("../Enums/Tipo_Error")
6 const ERRORES = require("../Ambito/S_Error")
7
8 > function FuncionesN( expresion, _ambito, _Error, _entorno, Simbol){ ...
41 }
42
43 > function f_TYPEOF( opIzq, _ambito, _Error, _entorno, Simbol){ ...
78 }
79
80 > function f_ROUND( opIzq, _ambito, _Error, _entorno, Simbol){ ...
100 }
101
102 > function f_TRUNCATE( opIzq, _ambito, _Error, _entorno, Simbol){ ...
122 }
123
124 > function f_LOWER( opIzq, _ambito, _Error, _entorno, Simbol){ ...
145 }
146
147 > function f_UPPER( opIzq, _ambito, _Error, _entorno, Simbol){ ...
168 }
169
170 > function f_length( opIzq, _ambito, _Error, _entorno, Simbol){ ...
191 }
192 > function f_TOCHARARRAY( opIzq, _ambito, _Error, _entorno, Simbol){ ...
222 }
223
224 module.exports = FuncionesN
```

## LOGICA.JS

Este archivo contiene el proceso de las operaciones lógicas.

```
JS Aritmetica.js M JS Logica.js X
backend > controller > Operacion > JS Logica.js > Logica
> console.log(
Aa Abi *

1 const TIPO_DATO = require("../Enums/TipoDato")
2 const TIPO_OPERACION = require("../Enums/TipoOperacion")
3 const TIPO_VALOR = require("../Enums/TipoValor")
4 const Relacional = require("../Relacional")
5 const ValorExpresion = require("../ValorExpresion")
6 const TIPO_ERROR = require("../Enums/Tipo_Error")
7 const ERRORES = require("../Ambito/S_Error")
8
9 > function Logica(_expresion, _ambito, _Error, _entorno, Simbol){
53 }
54
55 > function or(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
80 }
81
82 > function and(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
106 }
107 > function not(_opIzq, _ambito, _Error, _entorno, Simbol){ ...
136 }
137
138 module.exports = Logica
```

## OPERACIÓN.JS

Este archivo contiene es el que manda a llamar a cada archivo según el tipo de instrucción que sea.

```
JS Aritmetica.js M JS Operacion.js M X
backend > controller > Operacion > JS Operacion.js > Operacion
> console.log(
Aa Abi * No results
Find related code in Proyecto2_DECI

1 const TIPO_OPERACION = require("../Enums/TipoOperacion");
2 const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
3 const TIPO_VALOR = require("../Enums/TipoValor");
4 const Aritmetica = require("../Aritmetica");
5 const ValorExpresion = require("../ValorExpresion");
6 const Logica = require("../Logica");
7 const Relacional = require("../Relacional");
8 const ternario = require("../Ternario");
9 const funcionesM = require("../FuncionesM");
10
11 function Operacion(_expresion, _ambito, _Error, _entorno, Simbol){
12     if(_expresion.tipo === TIPO_VALOR.DECIMAL || _expresion.tipo === TIPO_VALOR.ENTERO ||
13         _expresion.tipo === TIPO_VALOR.BANDERA || _expresion.tipo === TIPO_VALOR.CARACTER ||
14         _expresion.tipo === TIPO_VALOR.CADENA || _expresion.tipo === TIPO_VALOR.IDENTIFICADOR){ ...
18     }
19
20     else if(_expresion.tipo === TIPO_OPERACION.SUMA || _expresion.tipo === TIPO_OPERACION.RESTA ||
21         _expresion.tipo === TIPO_OPERACION.MULTIPLICACION || _expresion.tipo === TIPO_OPERACION.DIVISION ||
22         _expresion.tipo === TIPO_OPERACION.MODULO || _expresion.tipo === TIPO_OPERACION.POTENCIA ||
23         _expresion.tipo === TIPO_OPERACION.NEGACION){ ...
24     }
25
26
27
28     else if(_expresion.tipo === TIPO_OPERACION.IGUALIGUAL || _expresion.tipo === TIPO_OPERACION.DIFERENTE ||
29         _expresion.tipo === TIPO_OPERACION.MENOR || _expresion.tipo === TIPO_OPERACION.MAYOR ||
30         _expresion.tipo === TIPO_OPERACION.MENORIGUAL || _expresion.tipo === TIPO_OPERACION.MAYORIGUAL){ ...
31     }
32
33
34     else if(_expresion.tipo === TIPO_OPERACION.OR || _expresion.tipo === TIPO_OPERACION.AND ||
35         _expresion.tipo === TIPO_OPERACION.NOT){ ...
36     }
37
38
39     else if(_expresion.tipo === TIPO_OPERACION.TERNARIO){ ...
40     }
41
42     else if(_expresion.tipo === TIPO_OPERACION.LENGTH || _expresion.tipo === TIPO_OPERACION.UPPER || _expresion.tipo === TIPO_OPE
43         _expresion.tipo === TIPO_OPERACION.TRUNCATE || _expresion.tipo === TIPO_OPERACION.ROUND || _expresion.tipo === TIPO_OPE
44         _expresion.tipo === TIPO_OPERACION.TOCHARARRAY){ ...
45     }
46
47
48     else if(_expresion.tipo === TIPO_INSTRUCCION.CASTEIO){ ...
49     }
50
51
52     else if(_expresion.tipo === TIPO_INSTRUCCION.ACCESO_V || _expresion.tipo === TIPO_INSTRUCCION.ACCESO_L){ ...
53     }
54
55
56     else if(_expresion.tipo === TIPO_INSTRUCCION.LLAMADA){ ...
57     }
58
59
60
61
62
63
64
65 }
```

## RELACIONAL.JS

Este archivo contiene el proceso de las operaciones relaciones.

```
JS Aritmetica.js M JS Operacion.js M JS Relacional.js X
backend > controller > Operacion > JS Relacional.js > [0] <unknown>
> console.log(
Aa Abi * N

1  const TIPO_DATO = require("../Enums/TipoDato");
2  const TIPO_OPERACION = require("../Enums/TipoOperacion");
3  const TIPO_VALOR = require("../Enums/TipoValor");
4  const TIPO_INSTRUCCION = require("../Enums/TipoInstruccion");
5  const Aritmetica = require("../Aritmetica");
6  const ValorExpresion = require("../ValorExpresion");
7  const TIPO_ERROR = require("../Enums/Tipo_Error");
8  const ERRORES = require("../Ambito/S_Error");
9
10
11
12 > function Relacional(_expresion, _ambito, _Error, _entorno, Simbol){ ...
46 }
47
48 > function igualigual(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
72 }
73 > function diferente(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
97 }
98 > function menor(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
122 }
123 > function mayor(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
147 }
148 > function menorigual(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
172 }
173 > function mayorigual(_opIzq, _opDer, _ambito, _Error, _entorno, Simbol){ ...
197 }
198 module.exports = Relacional; Find related code in Proyecto2 OLC1
```

## TERNARIO.JS

Este archivo contiene el proceso del operador ternario, que condición en donde si es verdadera ejecutará la primera expresión, de lo contrario ejecutará la segunda expresión.

```
JS Aritmetica.js M JS Operacion.js M JS Ternario.js X
backend > controller > Operacion > JS Ternario.js > Ternario
> console.log(
Aa Abi * ? of 4 ↑ ↓ ≡ ×

1  const TIPO_DATO = require("../Enums/TipoDato");
2  const TIPO_OPERACION = require("../Enums/TipoOperacion");
3  const TIPO_VALOR = require("../Enums/TipoValor");
4  const Relacional = require("../Relacional");
5  const ValorExpresion = require("../ValorExpresion");
6  const TIPO_ERROR = require("../Enums/Tipo_Error");
7  const ERRORES = require("../Ambito/S_Error");
8
9  function ternario(_instruccion, _ambito, _Error, _entorno, Simbol){
10 //console.log("-----entra al ternario de variables-----"> _entorno)
11 //console.log(_instruccion)
12 const c = _instruccion.expresion
13 const v = _instruccion.verdadero
14 const f = _instruccion.falso
15 const Operacion = require("../Operacion/Operacion")
16 const condicion = Operacion(c, _ambito, _Error, "ope. ternario", Simbol)
17 //console.log(condicion)
18 Find related code in Proyecto2 OLC1
19 if(condicion.tipo === TIPO_DATO.BANDERA){
20 var r_valor, r_tipo
21 > if(condicion.valor){ ...
25 > }else{ ...
29 }
30 //console.log("RESPUESTA-----"> r_valor)
31 return {
32 valor: r_valor,
33 tipo: r_tipo,
34 linea: _instruccion.linea,
35 columna: _instruccion.columna
36 }
37 }
38 //var respuesta = (opIzq.tipo === null ? opIzq.valor : "") + (opDer.tipo === null ? opDer.valor : "") //true+5+10+5
39 var nuevo = new ERRORES(TIPO_ERROR.SEMANTICO, "La condición no es de tipo BANDERA, es: ${condicion.tipo}", _instruccion.li
40 _Error.addErrores(nuevo)
41 return{
42 valor: "Error semántico: La condición no es de tipo BANDERA, es: ${condicion.tipo}... Línea: ${_instruccion.linea}
43 tipo: null,
44 linea: _instruccion.linea,
45 columna: _instruccion.columna
46 }
47 }
48
49 module.exports = ternario
```



## TIPORESULTADO.JS

Este archivo contiene las condiciones para saber el tipo de resultado para las operaciones aritméticas.

```
JS Arimeticajs M JS Operacionjs M JS TipoResultadojs X
backend > controller > Operacion > JS TipoResultadojs > TipoResultado
1 const TIPO_DATO = require("../Enums/TipoDato");
2
3 function TipoResultado(_tipo1, _tipo2, _operacion){
4     //SUMA
5     if(_operacion=="suma"){
6         if((_tipo1 === TIPO_DATO.ENTERO || _tipo2 === TIPO_DATO.ENTERO) && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA){
7             return TIPO_DATO.ENTERO
8         }
9         else if((_tipo1 === TIPO_DATO.CADENA || _tipo2 === TIPO_DATO.CADENA) && _tipo1===null && _tipo2===null){ return TIPO_DATO.CADENA }
10        else if((_tipo1 === TIPO_DATO.DECIMAL || _tipo2 === TIPO_DATO.DECIMAL) && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA){
11            return TIPO_DATO.DECIMAL
12        }
13        else if(_tipo1 === TIPO_DATO.CARACTER && _tipo2 === TIPO_DATO.CARACTER && _tipo1===null && _tipo2===null){ return TIPO_DATO.CARACTER }
14    }
15    else if(_operacion=="resta" && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA ){
16        if(_tipo1 === TIPO_DATO.ENTERO || _tipo2 === TIPO_DATO.ENTERO && _tipo1!==TIPO_DATO.DECIMAL && _tipo2!==TIPO_DATO.DECIMAL){
17            return TIPO_DATO.ENTERO
18        }
19        else if(_tipo1 === TIPO_DATO.DECIMAL || _tipo2 === TIPO_DATO.DECIMAL) { return TIPO_DATO.DECIMAL }
20    }
21    else if(_operacion=="multi" && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA && _tipo1!==TIPO_DATO.BANDERA && _tipo2!==TIPO_DATO.BANDERA){
22        if(_tipo1 === TIPO_DATO.ENTERO || _tipo2 === TIPO_DATO.ENTERO && _tipo1!==TIPO_DATO.DECIMAL && _tipo2!==TIPO_DATO.DECIMAL){
23            return TIPO_DATO.ENTERO
24        }
25        else if(_tipo1 === TIPO_DATO.DECIMAL || _tipo2 === TIPO_DATO.DECIMAL) { return TIPO_DATO.DECIMAL }
26    }
27    else if(_operacion=="division" && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA && _tipo1!==TIPO_DATO.BANDERA && _tipo2!==TIPO_DATO.BANDERA){
28        return TIPO_DATO.DECIMAL
29    }
30    else if(_operacion=="modulo" && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA && _tipo1!==TIPO_DATO.BANDERA && _tipo2!==TIPO_DATO.BANDERA){
31        return TIPO_DATO.DECIMAL
32    }
33    else if(_operacion=="potencia" && _tipo1!==TIPO_DATO.CADENA && _tipo2!==TIPO_DATO.CADENA && _tipo1!==TIPO_DATO.BANDERA && _tipo2!==TIPO_DATO.BANDERA){
34        if(_tipo1 === TIPO_DATO.ENTERO && _tipo2 === TIPO_DATO.ENTERO){ return TIPO_DATO.ENTERO }
35        else { return TIPO_DATO.DECIMAL }
36    }
37    else if(_operacion=="negacion" && (_tipo1===TIPO_DATO.ENTERO || _tipo1===TIPO_DATO.DECIMAL) ){
38        if(_tipo1 === TIPO_DATO.ENTERO){ return TIPO_DATO.ENTERO }
39        else if(_tipo1 === TIPO_DATO.DECIMAL){ return TIPO_DATO.DECIMAL }
40    }
41    return null
42 }
43
44 module.exports = TipoResultado
```

## VALOREXPRESION.JS

Este archivo devuelve el tipo, valor, id, línea y columna de una variable.

```
JS Arimeticajs M JS Operacionjs M JS ValorExpresionjs X
backend > controller > Operacion > JS ValorExpresionjs > <unknown>
1 const TIPO_DATO = require("../Enums/TipoDato");
2 const TIPO_VALOR = require("../Enums/TipoValor");
3 const TIPO_ERROR = require("../Enums/Tipo_Error");
4 const ERRORES = require("../Ambito/S_Error");
5
6 function ValorExpresion(_expresion, _ambito, _Error){
7     //console.log(_expresion.tipo+"<---->"+"_expresion.valor)
8     if(_expresion.tipo === TIPO_VALOR.DECIMAL){ ...
9     }
10    }
11    else if(_expresion.tipo === TIPO_VALOR.ENTERO){ ...
12    }
13    }
14    else if(_expresion.tipo === TIPO_VALOR.BANDERA){ ...
15    }
16    }
17    else if(_expresion.tipo === TIPO_VALOR.CADENA){ ...
18    }
19    }
20    else if(_expresion.tipo === TIPO_VALOR.CARACTER){ ...
21    }
22    }
23    else if(_expresion.tipo === TIPO_VALOR.IDENTIFICADOR){ ...
24    }
25    }
26    //IDENTIFICADOR
27    }
28
29 module.exports = ValorExpresion
```

## FRONTEND

En la carpeta frontedN >> src >> components contiene el html, css y ts a utilizar como cliente.

