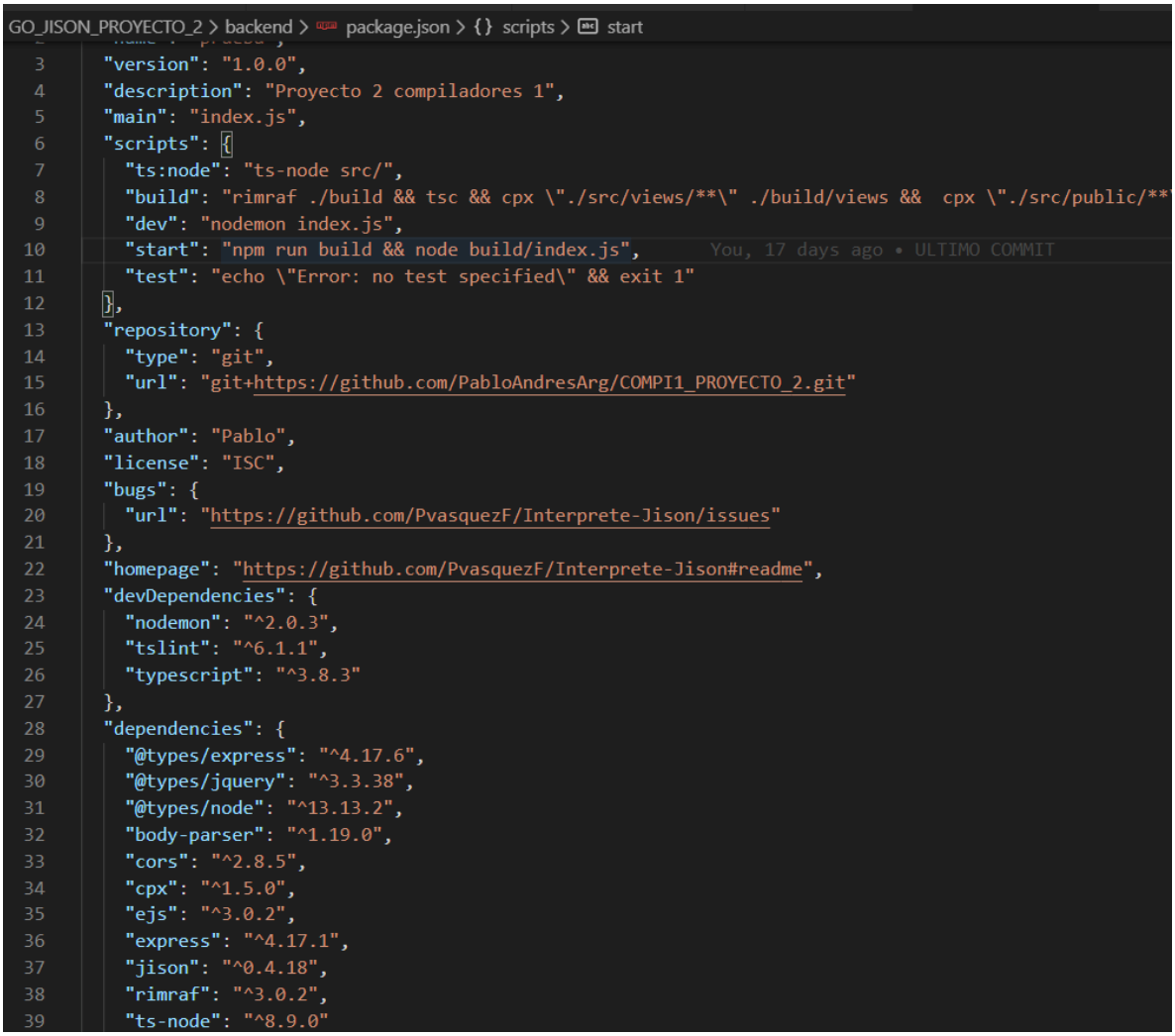
MANUAL TECNICO

Para la realización del proyecto se utilizo como editor de código visual Code por su versatilidad y ventajas que posee , mas que todo porque trae su propia consola.

SERVIDOR DE NODE:

el servidor realizado en node utilizo las siguientes dependencias las cuales aparecen en el package.json y son como la descripción de que lleva el proyecto en lo que es el vacíen y algunas herramientas como nodemon que permite estar al tanto de los cambios que se realizan en el backend y siempre y cuando este haya sido ejecutado.



Como se ve en la imagen mi server trabaja en el puerto 3000, y pues este recibe peticiones http en mi archivo index.ts que básicamente ahí es donde yo maneje todas las rutas del proyecto.

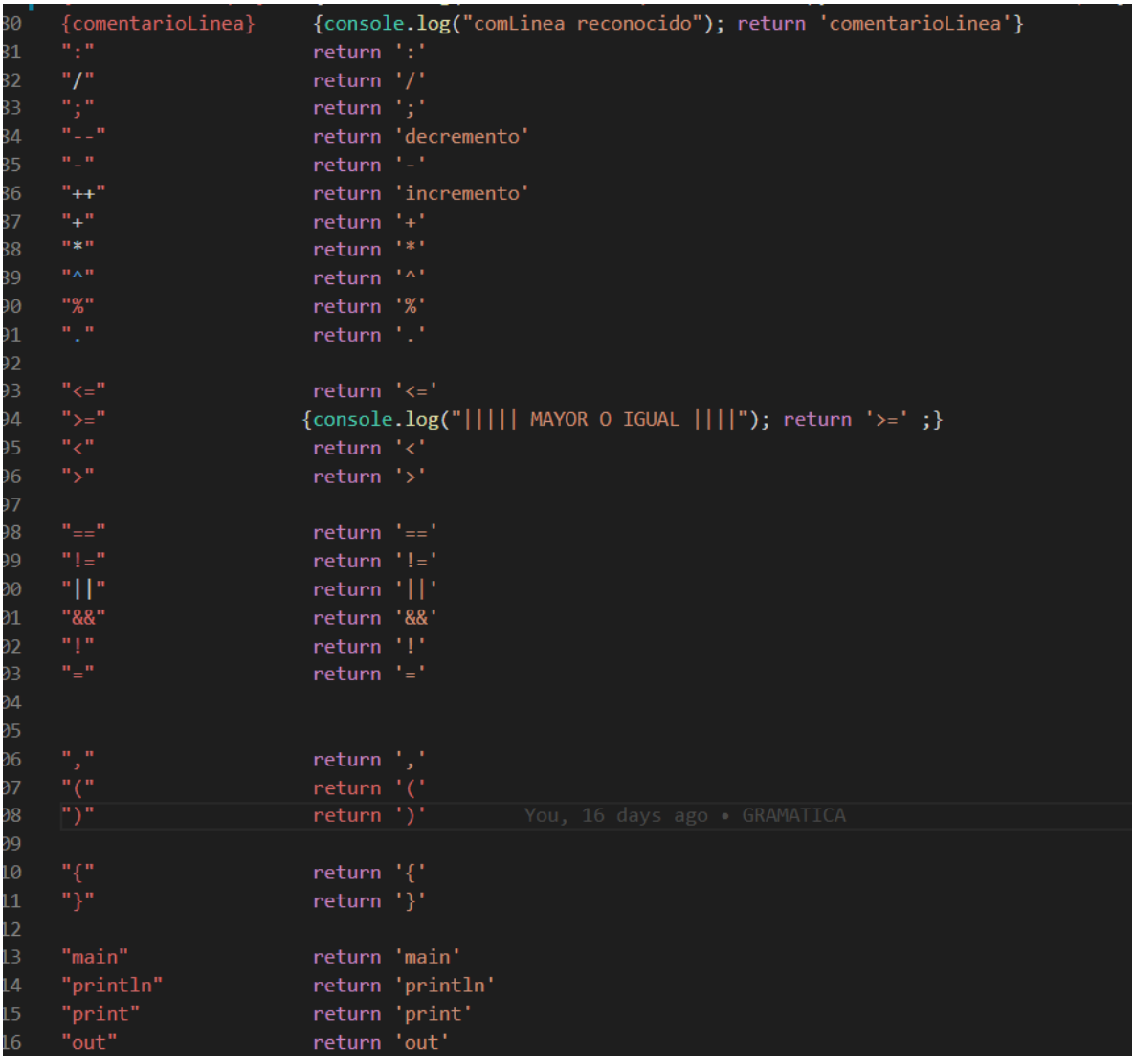
```
You, 5 hours ago | 1 author (You)
import express from 'express';
import { Table } from './Simbols/Table';
import { Break } from './Expresiones/Break';
import { Continue } from './Expresiones/Continue';
import { Exception } from './utils/Exception';
import { Errores } from "./ManejoErrores/Errores";
import { Importe } from './Otros/Importe';
import { Node } from './Abstract/Node';
import { fstat } from 'fs';
import { json } from 'body-parser';

import {Rep} from './REPORTES/Rep';

import { GraficaArbolAts } from './ManejoErrores/GraficaArbolAts';
import { Clase } from './REPORTES/Clase';


var bodyParser = require("body-parser");
const parser = require('./Grammar/Grammar.js');
const MyParser_300445 = require('./Grammar/graProyecto.js'); // ESTO ME SIRVE PARA LLAM
const cors = require('cors');
const app = express();
const port = 3000;

app.use(cors());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(__dirname + '/public'));
app.set('view engine', 'ejs');

app.set('views', __dirname);
app.use(express.urlencoded());
app.use(express.json());
app.use(bodyParser.json());

app.listen(port, err => {

```

SERVIDOR DE GO

El servidor de Go se utilizo para manejar el fronted como podemos ver en la función principal main de mi archivo .go se ven cargados varios archivos de javascript html y css , pestañas solo se utilizo una la cual esta en la función index por tanto el usuario al presentar la determinada ruta en el navegador pasara a visualizar todo el contenido de las carpetas que posee la función main.

GRAMATICA DE JISON

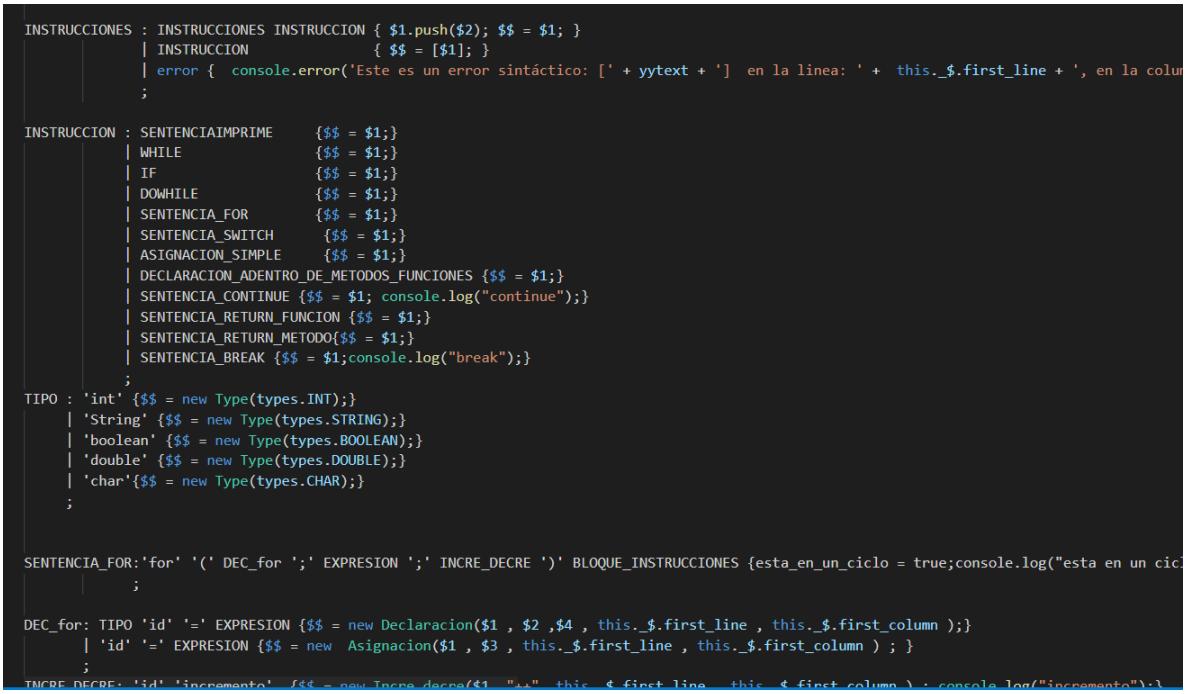
En este caso se muestra la sintaxis del análisis léxico con jison , esto no es parte de la gramática pero si es un paso necesario para realizarla puesto que son los tokens que se van a analizar sintácticamente.



```
{comentarioLinea}     {console.log("comLinea reconocido"); return 'comentarioLinea'}
":"                   return ':'
"/"                   return '/'
";"                   return ';'
"--"                  return 'decremento'
"-"                   return '-'
"++"                  return 'incremento'
"+"                   return '+'
"*"                   return '*'
"^"                   return '^'
"%"                   return '%'
"."                   return '.'

"<="                  return '<='
">="                 {console.log("||||| MAYOR O IGUAL ||||"); return '>=' ;}
"<"                   return '<'
">"                   return '>'

"=="                  return '=='
"!="                  return '!='
"||"                  return '||'
"&&"                  return '&&'
"!"                   return '!'
"="                   return '='


","                   return ','
"("                   return '('
")"                   return ')'        You, 16 days ago • GRAMATICA

"{"                   return '{'
"}"                   return '}'

"main"                return 'main'
"println"             return 'println'
"print"               return 'print'
"out"                 return 'out'
```

Jison utiliza gramática ascendente por tanto si hay recursividad a la derecha o hacia la izquierda le es indiferente.

La forma para extraer errores en jison es por medio de la práctica de **producción de error**, como es el caso en de la producción INSTRUCCIONES la cual posee un terminal llamado error el cual es un indicador que no venia lo que yo esperaba por tanto así se maneja en otras producciones no en todas porque causa conflictos solo tiene que ir en lugares específicos.

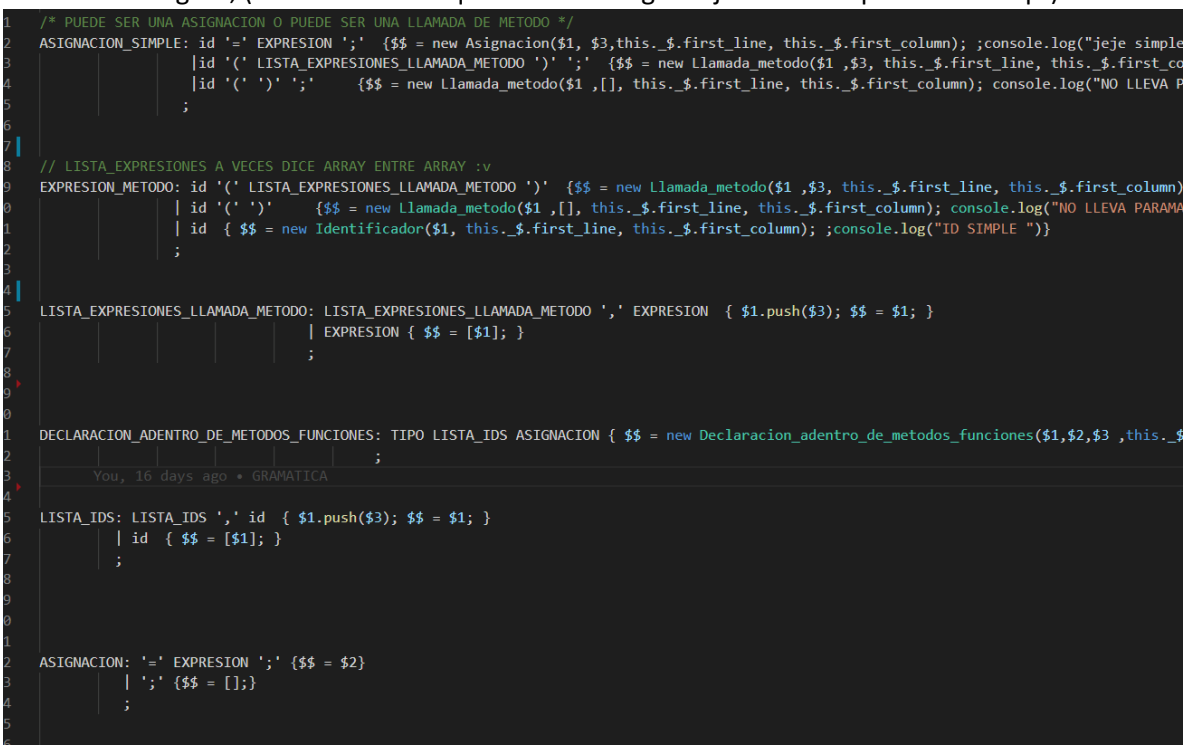


```
INSTRUCCIONES : INSTRUCCIONES INSTRUCCION { $1.push($2); $$ = $1; }
              | INSTRUCCION               { $$ = [$1]; }
              | error {  console.error('Este es un error sintáctico: [' + yytext + ']  en la linea: ' +  this._$.first_line + ', en la colu
              ;

INSTRUCCION : SENTENCIAIMPRIME     {$$ = $1;}
            | WHILE                {$$ = $1;}
            | IF                   {$$ = $1;}
            | DOWHILE              {$$ = $1;}
            | SENTENCIA_FOR        {$$ = $1;}
            | SENTENCIA_SWITCH      {$$ = $1;}
            | ASIGNACION_SIMPLE     {$$ = $1;}
            | DECLARACION_ADENTRO_DE_METODOS_FUNCIONES {$$ = $1;}
            | SENTENCIA_CONTINUE {$$ = $1; console.log("continue");}
            | SENTENCIA_RETURN_FUNCION {$$ = $1;}
            | SENTENCIA_RETURN_METODO{$$ = $1;}
            | SENTENCIA_BREAK {$$ = $1;console.log("break");}
            ;
TIPO : 'int' {$$ = new Type(types.INT);}
     | 'String' {$$ = new Type(types.STRING);}
     | 'boolean' {$$ = new Type(types.BOOLEAN);}
     | 'double' {$$ = new Type(types.DOUBLE);}
     | 'char'{$$ = new Type(types.CHAR);}
     ;


SENTENCIA_FOR:'for' '(' DEC_for ';' EXPRESION ';' INCRE_DECRE ')' BLOQUE_INSTRUCCIONES {esta_en_un_ciclo = true;console.log("esta en un cic
             ;

DEC_for: TIPO 'id' '=' EXPRESION {$$ = new Declaracion($1 , $2 ,$4 , this._$.first_line , this._$.first_column );}
       | 'id' '=' EXPRESION {$$ = new  Asignacion($1 , $3 , this._$.first_line , this._$.first_column ) ; }
       ;
INCRE_DECRE: 'id' 'incremento'  {$$ = new Incre_decre($1 , "++" , this._$.first_line , this._$.first_column ) ; console.log("incremento");}
```

LA sintaxis de JISON es la siguiente esos “$$” extraños que se ven ahí son variables dentro del lenguaje de jison y lo que dice es lo siguiente $$ hace referencia a la produccion , $n son los no terminales o terminales , donde n puede ser cualquier numero dependiendo de la cantidad de terminales y no terminales que se produce , como se puede ver la única vez que se tiene que retornar el $$ es cuando se va devolver el árbol por ejemplo en la producción de inicio , como es ascendente se forma de las hojas hasta la raíz cuando llega al EOF(símbolo para indicar que la entrada termino) ahí se retorna el árbol el cual es un nodo que tiene a todos los nodos.

```
INICIO : LISTA_IMPORTES_CLASES EOF {$$ = new Tree($1);console.log("CREACION DEL ARBOL"); return $$;}
       |  LISTA_IMPORTE EOF {$$ = new Tree($1);console.log("CREACION DEL ARBOL"); return $$;}
       |  LISTA_CLASES  EOF{$$ = new Tree($1);console.log("CREACION DEL ARBOL"); return $$;}
       | EOF{$$ = new Tree($1);console.log("CREACION DEL ARBOL"); return $$;}
       ;

LISTA_IMPORTES_CLASES:  LISTA_IMPORTE LISTA_CLASES {let init =  new Inicio($1, $2); $$ = init.Lista_importes_clases}
                     ;

LISTA_IMPORTE: LISTA_IMPORTE IMPORTE { $1.push($2); $$ = $1; }
       | IMPORTE    { $$ = [$1]; }
       ;

LISTA_CLASES: LISTA_CLASES SENTENCIA_CLASE{ $1.push($2); $$ = $1; }      You, 12 days ago • init ats
           | SENTENCIA_CLASE  { $$ = [$1]; }
           ;



IMPORTE: 'import' 'id' ';'   {$$ = new Importe($2, $2 ,  this._$.first_line, this._$.first_column);}
       ;


SENTENCIA_CLASE:'class' 'id' BLOQUE_DECLARACIONES_METFUNVAR {$$ = new ClaseInstruccion($2, $3 ,  this._$.first_line, this._$.first_column);}
              | error {  console.error('Este es un error sintáctico: [' + yytext + ']  en la linea: ' +  this._$.first_line + ', en la colum
              ;

// tengo que retornar algo en los errores para que mi arbol no truene

BLOQUE_DECLARACIONES_METFUNVAR : '{' LISTA_DECLARACIONES_METFUNVAR_P '}' {$$ = $2;}            /* este es para que acepte vacios*/
                               | '{' '}' {$$ = [];}
                               | error {  console.error('Este es un error sintáctico: [' + yytext + ']  en la linea: ' +  this._$.first_line
                               ;
```

La parte más compleja de jison al menos personalmente es la parte de las listas o producciones recursivas a la izquierda , pues la sintaxis de corchetes como generalmente suele ser es para arreglos , (tener en cuenta que todo el código de jison se compila a JavaScript)



```
/* PUEDE SER UNA ASIGNACION O PUEDE SER UNA LLAMADA DE METODO */
ASIGNACION_SIMPLE: id '=' EXPRESION ';'  {$$ = new Asignacion($1, $3,this._$.first_line, this._$.first_column); ;console.log("jeje simple
                 |id '(' LISTA_EXPRESIONES_LLAMADA_METODO ')' ';'  {$$ = new Llamada_metodo($1 ,$3, this._$.first_line, this._$.first_co
                 |id '(' ')' ';'     {$$ = new Llamada_metodo($1 ,[], this._$.first_line, this._$.first_column); console.log("NO LLEVA P
                ;


// LISTA_EXPRESIONES A VECES DICE ARRAY ENTRE ARRAY :v
EXPRESION_METODO: id '(' LISTA_EXPRESIONES_LLAMADA_METODO ')'  {$$ = new Llamada_metodo($1 ,$3, this._$.first_line, this._$.first_column)
                | id '(' ')'     {$$ = new Llamada_metodo($1 ,[], this._$.first_line, this._$.first_column); console.log("NO LLEVA PARAMA
                | id  { $$ = new Identificador($1, this._$.first_line, this._$.first_column); ;console.log("ID SIMPLE ")}
                ;


LISTA_EXPRESIONES_LLAMADA_METODO: LISTA_EXPRESIONES_LLAMADA_METODO ',' EXPRESION  { $1.push($3); $$ = $1; }
                                | EXPRESION { $$ = [$1]; }
                                ;



DECLARACION_ADENTRO_DE_METODOS_FUNCIONES: TIPO LISTA_IDS ASIGNACION { $$ = new Declaracion_adentro_de_metodos_funciones($1,$2,$3 ,this._$
                                        ;
      You, 16 days ago • GRAMATICA

LISTA_IDS: LISTA_IDS ',' id  { $1.push($3); $$ = $1; }
         | id  { $$ = [$1]; }
         ;




ASIGNACION: '=' EXPRESION ';' {$$ = $2}
          | ';' {$$ = [];}
          ;
```

Jison también permite colocar javascript puro entre llaves por si en alguna aplicación se necesita hacer uso de programación mas enfocada a nuestro problema , en el caso de la realizacion del proyecto se uso para ir generando los nodos e importes en la sección de hasta arriba de Jison.

```
INSTRUCCIONES : INSTRUCCIONES INSTRUCCION { $1.push($2); $$ = $1; }
              | INSTRUCCION              { $$ = [$1]; }
              | error {  console.error('Este es un error sintáctico: [' + yytext + ']  en la linea: ' +  this._$.first_line + ', en la colu
              ;

INSTRUCCION : SENTENCIAIMPRIME     {$$ = $1;}
            | WHILE                {$$ = $1;}
            | IF                   {$$ = $1;}
            | DOWHILE              {$$ = $1;}
            | SENTENCIA_FOR        {$$ = $1;}
            | SENTENCIA_SWITCH      {$$ = $1;}
            | ASIGNACION_SIMPLE     {$$ = $1;}
            | DECLARACION_ADENTRO_DE_METODOS_FUNCIONES {$$ = $1;}
            | SENTENCIA_CONTINUE {$$ = $1; console.log("continue");}
            | SENTENCIA_RETURN_FUNCION {$$ = $1;}
            | SENTENCIA_RETURN_METODO{$$ = $1;}
            | SENTENCIA_BREAK {$$ = $1;console.log("break");}
            ;
TIPO : 'int' {$$ = new Type(types.INT);}
     | 'String' {$$ = new Type(types.STRING);}
     | 'boolean' {$$ = new Type(types.BOOLEAN);}
     | 'double' {$$ = new Type(types.DOUBLE);}
     | 'char'{$$ = new Type(types.CHAR);}
     ;


SENTENCIA_FOR:'for' '(' DEC_for ';' EXPRESION ';' INCRE_DECRE ')' BLOQUE_INSTRUCCIONES {esta_en_un_ciclo = true;console.log("esta en un cic
             ;

DEC_for: TIPO 'id' '=' EXPRESION {$$ = new Declaracion($1 , $2 ,$4 , this._$.first_line , this._$.first_column );}
       | 'id' '=' EXPRESION {$$ = new  Asignacion($1 , $3 , this._$.first_line , this._$.first_column ) ; }
       ;
INCRE_DECRE: 'id' 'incremento'  {$$ = new Incre_decre($1 , "++" , this._$.first_line , this._$.first_column ) ; console.log("incremento");}
```

```
CONDICION : '(' EXPRESION ')' {$$ = $2;}
          ;




BLOQUE_INSTRUCCIONES : '{' INSTRUCCIONES '}' {$$ = $2;}              /* este es para que acepte vacios*/
                     | '{' '}'    {$$ = [];}
                     ;

EXPRESION : '-' EXPRESION %prec UMENOS       { $$ = new Arithmetic($2, null, '-', this._$.first_line, this._$.first_column); }
          | '!' EXPRESION                    { $$ = new Arithmetic($1, null, '!', this._$.first_line, this._$.first_column); }
          | EXPRESION '+' EXPRESION          { $$ = new Arithmetic($1, $3, '+', this._$.first_line, this._$.first_column); }
          | EXPRESION '-' EXPRESION          { $$ = new Arithmetic($1, $3, '-', this._$.first_line, this._$.first_column); }
          | EXPRESION '*' EXPRESION          { $$ = new Arithmetic($1, $3, '*', this._$.first_line, this._$.first_column); }
          | EXPRESION '/' EXPRESION          { $$ = new Arithmetic($1, $3, '/', this._$.first_line, this._$.first_column); }
          | EXPRESION '%' EXPRESION          { $$ = new Arithmetic($1, $3, '%', this._$.first_line, this._$.first_column); }
          | EXPRESION '^' EXPRESION          { $$ = new Arithmetic($1, $3, '^', this._$.first_line, this._$.first_column); }
          | EXPRESION '<' EXPRESION          { $$ = new Relational($1, $3, '<', this._$.first_line, this._$.first_column); }
          | EXPRESION '>' EXPRESION          { $$ = new Relational($1, $3, '>', this._$.first_line, this._$.first_column); }
          | EXPRESION '>=' EXPRESION         { $$ = new Relational($1, $3, '>=', this._$.first_line, this._$.first_column); console.log(">=
          | EXPRESION '<=' EXPRESION         { $$ = new Relational($1, $3, '<=', this._$.first_line, this._$.first_column); }
          | EXPRESION '==' EXPRESION         { $$ = new Relational($1, $3, '==', this._$.first_line, this._$.first_column); }
          | EXPRESION '!=' EXPRESION         { $$ = new Relational($1, $3, '!=', this._$.first_line, this._$.first_column); }
          | EXPRESION '||' EXPRESION         { $$ = new Logic($1, $3, '&&', this._$.first_line, this._$.first_column); }
          | EXPRESION '&&' EXPRESION         { $$ = new Logic($1, $3, '||', this._$.first_line, this._$.first_column); }
          | 'decimal'                        { $$ = new Primitive(new Type(types.DOUBLE), Number($1), this._$.first_line, this._$.first_col
          | 'true'                           { $$ = new Primitive(new Type(types.BOOLEAN), true, this._$.first_line, this._$.first_column);
          | 'false'                          { $$ = new Primitive(new Type(types.BOOLEAN), false, this._$.first_line, this._$.first_column)
          | STRING_LITERAL                   { $$ = new Primitive(new Type(types.STRING), $1.replace(/\"/g,""), this._$.first_line, this._$
          | EXPRESION_METODO                 { $$ = $1}
          | caracter                         { $$ = new Primitive(new Type(types.CHAR), $1.replace(/\'/g,""), this._$.first_line, this._$.f
          | entero                           { $$ = new Primitive(new Type(types.INT), Number($1) , this._$.first_line, this._$.first_colum
          | '(' EXPRESION ')'                { $$ = $2; }
          ;
```