Contenido

IDE utilizado:

Visual Studio Code.

Lenguaje Utilizado:

JavaScript

Sistema Operativo:

Windows 10 (64 bits)

Comandos Utilizados para backend:

```
npm init -y
npm i express
npm i morgan
npm i cors
npm i nodemon -D
```

Comandos Utilizados para client:

```
npm create vite@latest
npm i reac-router-dom
npm i axios
npm run dev
```

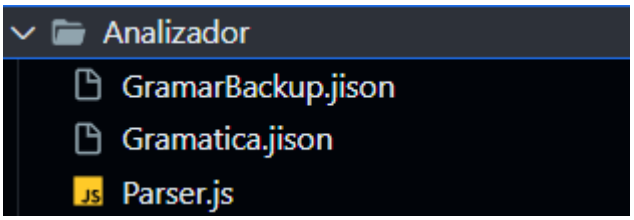
Librería utilizada para analizador léxico y sintáctico:

- Jison

Carpetas:

Analizador:

Carpeta donde se encuentran los archivos para los analizadores:



Gramatica.jison:

_

```
// ANALIZADOR LEXICO
%lex
// no importe si es mayusc o minusc
%options case-insensitive

//EXP
//comentario simple:
comment  "--".*;
integer  [0-9]+;
// double = numero punto numero
double    ([0-9]+("."[0-9]+)) ;
// id
id       [a-zA-Z][a-zA-Z0-9]*;
//Varchar:
varchar   \"([^\\\"]|\\.)*\"; // \" = comillas
//Fechas
dates ([0-9]{4} "-" [0-9]{2} "-"[0-9]{2}  );
//Variables:
variable  "@"[a-zA-Z][a-zA-Z0-9]*;
```

E

```
%%
//Reglas Lexicas
"--".*                  {console.log('coment
/*

https://github.com/jd-toralla/OLC1-2S2023/

*/
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/] {conso
//tipo de datos
"int"               {return 'R_INT';}
"double"            {return 'R_DOUBLE';}
"varchar"           {return 'R_VARCHAR';}
"date"              {return 'R_DATE';}
"boolean"           {return 'R_BOOLEAN';}

"table"             {return 'R_TABLE';}

//INSERT
"insert"            {return 'R_INSERT';}
"into"              {return 'R_INTO';}
"values"            {return 'R_VALUES';}
//DDL
"create"            {return 'R_CREATE';}
"alter"             {return 'R_ALTER';}
"drop"              {return 'R_DROP';}
"column"            {return 'R_COLUMN';}
"rename"            {return 'R_RENAME';}
"add"               {return 'R_ADD';}
"to"                {return 'R_TO';}
"delete"            {return 'R_DELETE';}
```

Espacio utilizado para Obtener las clases de Patron-Interprete a utilizar para poder interpretar :

```
/lex
//Sintactico

//llamados
%{
    const Primitivo = require('../Interprete/expresion/Primitivo.js');
    const expresion = require('../Interprete/expresion/Expresion.js');
    const expresionLo = require('../Interprete/expresion/ExpresionLo.js');
    const TipoDato = require('../Interprete/Enums/TipoDato.js');
    const TipoOp   = require('../Interprete/Enums/TipoOp.js');
    const Asig      = require('../Interprete/Clases/Asig.js');
    const Actualizar = require('../Interprete/Clases/Actualizar.js');
    const IfC = require('../Interprete/Clases/IF.js');
    const IfElse = require('../Interprete/Clases/IFELSE.js');
    //CallVar
    const CallVar = require('../Interprete/Clases/CallVar.js');
    //PRINT
    const Mostraar = require('../Interprete/Clases/PRINT.js');

    //Casteo:
    const Casteo = require('../Interprete/Clases/Casteo.js');
    const Columnas = require('../Interprete/Tables/Columnas.js');
    const TableeI = require('../Interprete/Tables/TableeI.js');
    const InsertIng = require('../Interprete/Tables/InsertIn.js');
    const AddColumn = require('../Interprete/Tables/AddColumn.js');
    const DropColumn = require('../Interprete/Tables/DropColumn.js');
    const AlterT = require('../Interprete/Tables/AlterT.js');
    const RenameTable = require('../Interprete/Tables/RenameTable.js');
    const RenameColumn = require('../Interprete/Tables/RenameColumn.js');
    const SelectTable = require('../Interprete/Tables/SelectTable.js');
    const ConditionT = require('../Interprete/Tables/CondicionT.js');
    const UpdateT = require('../Interprete/Tables/UpdateT.js');
    const DeleteT1= require('../Interprete/Tables/DeleteT.js');
    const Bloque = require('../Interprete/Clases/BloqueBegin.js');
```

Precedencia:

```
// Precedencia
%left 'OR'
%left 'AND'
%left 'NOT'
%left 'MAYORK' 'MENORK' 'EQUALS'
%left 'MAYORIK' 'MENORIK'
%left 'MAS' 'MENOS'
%left 'DIVI' 'POR' 'MODULO'
%left 'POTENCIA'
%left UMINUS
```

Precedencia utilizada: Izquierda.

Entre mas abajo este la precedencia mayor será su importancia.

Explicación breve de Jison:

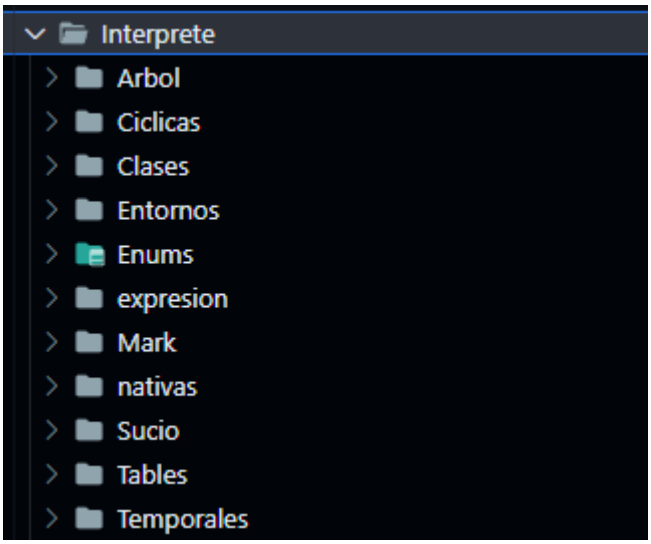
Meno expresión es para cuando vengas números, entre más abajo se encuentre la gramatica mayor precedencia tendrá.

```
expresion
    : MENOS expresion %prec UMINUS {
        console.log('-' +$2 );
        $$ = new expresion($2,null,TipoOp.RESTA, true,this._$.first_line, this._$.first_column);
        }
    | expresion AND expresion {
        console.log( 'and' );
        $$ = new expresionLo($1,$3,TipoOp.AND,this._$.first_line, this._$.first_column);
        }
    | expresion EQUALS expresion {
        console.log( 'IGUAL' );
        $$ = new expresionLo($1,$3,TipoOp.IGUAL,this._$.first_line, this._$.first_column);
        }
    | expresion OR expresion {
        console.log( 'OR' );
        $$ = new expresionLo($1,$3,TipoOp.OR,this._$.first_line, this._$.first_column);
        }

    | expresion MAYORK expresion {
        console.log( 'MAYORK' );
        $$ = new expresionLo($1,$3,TipoOp.MAYORK,this._$.first_line, this._$.first_column);
        }
    | expresion MENORK expresion {
        console.log( 'MENORK' );
        $$ = new expresionLo($1,$3,TipoOp.MENORK,this._$.first_line, this._$.first_column);
        }
    | expresion MENORIK expresion {
        console.log( 'MAYORIK' );
        $$ = new expresionLo($1,$3,TipoOp.MENORIK,this._$.first_line, this._$.first_column);
        }
    | expresion MAYORIK expresion {
        console.log( 'MAYORIK' );
        $$ = new expresionLo($1,$3,TipoOp.MAYORIK,this._$.first_line, this._$.first_column);
        }

    | expresion MAS expresion {
        console.log( 'SUMA ' +$1 + ' + ' +$3 );
        $$ = new expresion($1,$3,TipoOp.SUMA, false,this._$.first_line, this._$.first_column);
```

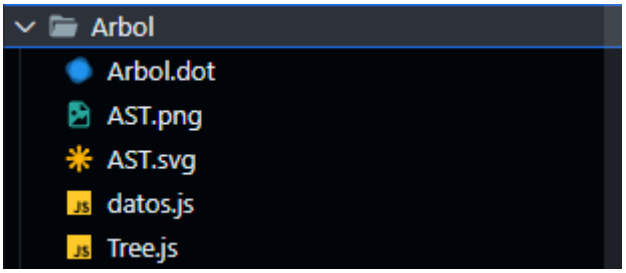
$$ es el valor que retorna dicho no terminal el cual en ciertas ocasiones se hace uso de las clases importadas anteriormente las cuales interpretaran el código.

```
print
    :R_PRINT  expresion  {

        console.log(`Print:`);
        $$ = new Mostraar($2,this._$.first_line, this._$.first_column);
    }
;
```

Interprete

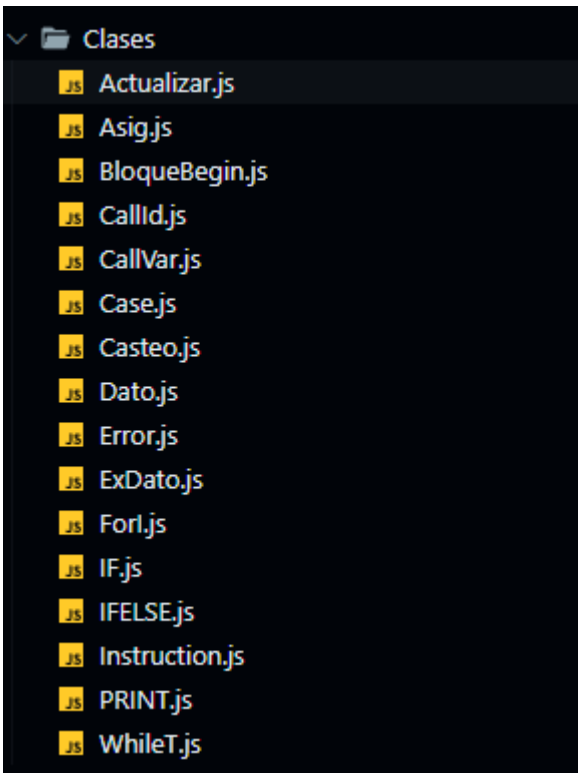


Arbol



Datos.js: Sera el archivo utilizado para asignar un valor a los nodos del ast el cual es un contador el cual va aumentando.

Clases:



Por ejemplo estas clases usan “extends Instruction” para poder hacer uso de métodos como “interpretar” el cual recibe entornos y lista de errores. Y generarAst();

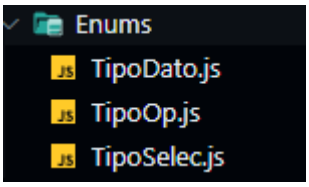
```
class PRINT extends Instruction{
    constructor(expresion, fila, columna){
        super();
        this.expresion = expresion;
        this.fila = fila;
        this.columna = columna;
    }

    CodiumAI: Test this method
    interpretar(entorno,lista_errores){
        try {
            var valor = this.expresion.interpretar(entorno,lista_errores);
            console.log('clg: ' + valor.valor)
            res.status(200).json({message:valor.valor });
        } catch (error) {
            console.log(`Error en interpretar PRINT ${this.fila}`)

        }
        agregarSalir(valor.valor);
        return valor.valor

    }

    CodiumAI: Test this method
    generarAst(){
        let nodo ={
            padre:-1,
            cadena:"",
        }
        let instPadre = obtenerContador();
        let exp = this.expresion.generarAst();
        let son = obtenerContador();
        let dad = obtenerContador();
        nodo.cadena = exp.cadena +
```

Entornos:

Usados para poder almacenar variables de cada entorno, ejemplo un If tiene su propio entorno en el cual se crean variables pero fuera de dicho if las variables declaradas no pueden ser llamadas.

```
/*
Auxiliar: ALex Guerra
https://github.com/AlexIngGuerra/OLC1-2S2023
*/
CodiumAI: Test this class
class Entorno{
    constructor(actual,anterior){
        this.actual = actual;
        this.anterior = anterior;
        this.variables = {};
        this.variablesId = new Map();
        this.funciones = {};
        this.tablas = {};
        //imprimir
        console.log(`Actual: ${actual}`)
    }

    CodiumAI: Test this method
    addVariable(variable){
        console.log(`Agregando variable: ${variable.id} en entorno: ${this.actual}`)
        this.variables[variable.id] = variable;
    }
```

ENUMS:



Utilizado para facilitar el uso de tipo de datos, operaciones y select.

```
const TipoDato = {
    'INT' : 'INT',
    'DOUBLE' : 'DOUBLE',
    'VARCHAR' :'VARCHAR',
    'DATE' : 'DATE',
    'BOOLEAN' : 'BOOLEAN',
    'NULL' : 'NULL',
    'TRUE' : 'TRUE',
    'FALSE' : 'FALSE',
    'ID':'ID'
}
```

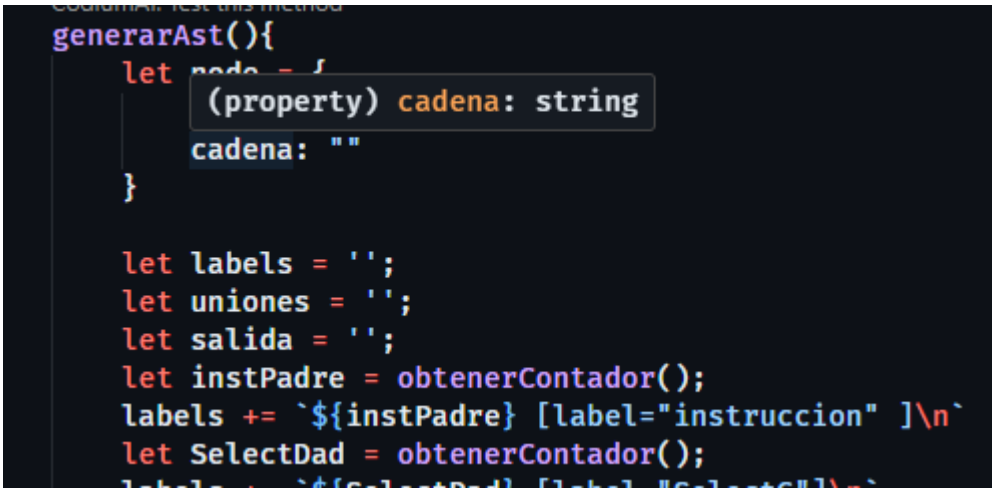
Temporales

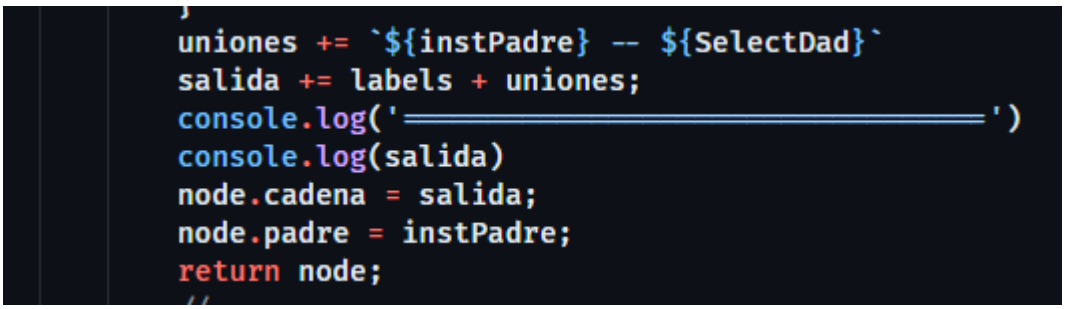
Variables creadas para almacenar la salida que será enviada a client.

```
let printsSalid = '';
let selectSalidas = '';

CodiumAI: Test this function
const agregarSalir = (texto) => {
    printsSalid += texto + '\n';
}

CodiumAI: Test this function
const agregarSelect = (texto) => {
    selectSalidas += texto + '\n';
}

CodiumAI: Test this function
const getSalida = () => {
    return printsSalid;
}
CodiumAI: Test this function
const getSeleccion = () => {
    return selectSalidas;
}

//clears
CodiumAI: Test this function
const clearSalida = () => {
    printsSalid = '';
}
CodiumAI: Test this function
const clearSelec = () => {
    selectSalidas = '';
}
module.exports = {agregarSalir, agregarSelect, getSalida, getSeleccion, clearSalida,clearSelec}
```

generarAst:

```
Auxiliar: ALex Guerra
https://github.com/AlexIngGuerra/OLC1-2S2023
```


generarAst(){
let
(property) cadena: string
cadena: ""
}
let labels = '';
let uniones = '';
let salida = '';
let instPadre = obtenerContador();
labels += `${instPadre} [label="instruccion" ]\n`
let SelectDad = obtenerContador();


uniones += `${instPadre} -- ${SelectDad}`
salida += labels + uniones;
console.log('═══════════════════════════════════════')
console.log(salida)
node.cadena = salida;
node.padre = instPadre;
return node;