Nueva Guatemala de la Asunción

Manual de Técnico

Proyecto 2: OLC-1

Elaborado por: Estuardo Gabriel Son Mux
Carné: 202003894
Fecha: 23/04/2022

Índice

Alcances y Objetivos del Programa

El objetivo principal del programa es realizar las acciones de un intérprete el cual realiza el análisis léxico, sintáctico y semántico de un archivo de texto de extensión cst el cual debe cumplir con condiciones gramaticales para ser reconocido adecuadamente. Así mismo debe presentar reportes de los errores gramaticales detectados durante la lectura del programa, el árbol de análisis sintáctico resultado del análisis y la tabla de símbolos resultado de la ejecución de las instrucciones cuya sintaxis es correcta y no posea errores semánticos.

Especificaciones Técnicas

Requisitos de Hardware

- Computadora con todos sus componentes para su correcto funcionamiento
- Laptop

Requisitos de Software

Sistema Operativo

El programa fue desarrollado en una laptop con sistema operativo Windows 10, Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz   1.80 GHz.

Especificaciones del dispositivo

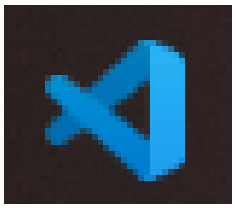
Nombre del dispositivo	DESKTOP-B6R45EM
Procesador	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
RAM instalada	8.00 GB
Id. del dispositivo	077D5E5A-52E8-4649-AED3-5A498B8678A7
Id. del producto	00325-80915-48720-AAOEM
Tipo de sistema	Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil	Compatibilidad con entrada táctil con 10 puntos táctiles

Lenguaje de Programación

- JavaScript
- TypeScript

IDE o Editor de Código

Visual Studio Code



Lógica del Programa

Estructura de Archivos CST

Los archivos CST poseen la siguiente estructura.

```
int var1 = 1;
int punteo = 0;

run InicioArchivo1();

InicioArchivo1():void {
    println("-----------------CALIFICACION ARCHIVO 1-----------------");
    println("Valor: 15 pts");
    println("------------------------------------------------------------");
    int var1 = 0;
    //Verificar ambitos, se toma con prioridad la variable local ante la global.
    if (var1 != 0) {
        println("No se toma con prioridad la variable local ante la global");
        println("Perdiste 8 puntos :c");
    }
    else {
        punteo = punteo + 8;
        println("Muy bien, prioridad de variable local correcta");
        println("Haz sumado 8 puntos");
        println("Punteo = " + punteo);
    }
```

Para ellos se hizo un reconocimiento de tokens, se debe tener en cuenta que la gramática es case-insensitive.

Palabras Reservadas

```
"void" Devuelve el token 'VOID'
"int" Devuelve el token 'INT';
"double" Devuelve el token 'DOUBLE';
"boolean" Devuelve el token 'BOOLEAN';
"switch" Devuelve el token 'SWITCH';
"case" Devuelve el token 'CASE';
"if" Devuelve el token 'IF';
"else" Devuelve el token 'ELSE';
"char" Devuelve el token 'CHAR';
"string" Devuelve el token 'STRING';
"true" Devuelve el token 'TRUE';
"false" Devuelve el token 'FALSE';
"break" Devuelve el token 'BREAK';
"continue" Devuelve el token 'CONTINUE';
"default" Devuelve el token 'DEFAULT';
"return" Devuelve el token 'RETURN';
"do" Devuelve el token 'DO';
"while" Devuelve el token 'WHILE';
"for" Devuelve el token 'FOR';
"println" Devuelve el token 'PRINTLN';
"print" Devuelve el token 'PRINT';
"tolower" Devuelve el token 'TOLOWER';
"toupper" Devuelve el token 'TOUPPER';
"round" Devuelve el token 'ROUND';
"length" Devuelve el token 'LENGTH';
"typeof" Devuelve el token 'TYPEOF';
"tostring" Devuelve el token 'TOSTRING';
"tochararray" Devuelve el token 'TOCHARARRAY';
"run" Devuelve el token 'RUN';
"new" Devuelve el token 'NEW';
```

Expresiones Regulares

```
//Expresion para reconocer numeros Identificadores o ID
(_)*[a-zA-ZnÑ]+[_a-zA-Z0-9ñÑ]*\b Devuelve el token 'IDENTIFICADOR';

//Expresion para reconocer Cadenas de Texto
\"(\\n|\\\"|\\\'|\\t|\\\\|\\r|[^\"\\\n])*\"  Devuelve el token 'CADENA'

//Expresion para reconocer numeros Decimales
[0-9]+(\.[0-9]+)\b Devuelve el token 'DECIMAL'

//Expresion para reconocer numeros Enteros
[0-9]+\b Devuelve el token 'ENTERO'

//Expresion para reconocer Caracteres Char
\'(\\n|\\\"|\\\'|\\t|\\\\|\\r|[^\'\\\n])\' Devuelve el token 'CARACTER'
```

Símbolos

```
"++" Devuelve el token 'INCREMENTO';
"--" Devuelve el token 'DECREMENTO';
"+" Devuelve el token 'SUMA';
"-" Devuelve el token 'RESTA';
"/" Devuelve el token 'DIVISION';
"^" Devuelve el token 'POTENCIA';
"*" Devuelve el token 'MULTIPLICACION';
"%" Devuelve el token 'MODULO';
"==" Devuelve el token 'IGUAL';
"=" Devuelve el token 'ASIGNACION';
"!=" Devuelve el token 'DIFERENTE';
"!" Devuelve el token 'NEGACION';
"<=" Devuelve el token 'MENORIGUAL';
"<" Devuelve el token 'MENOR'
">=" Devuelve el token 'MAYORIGUAL';
">" Devuelve el token 'MAYOR';
"(" Devuelve el token 'PARABRE';
")" Devuelve el token 'PARCIERRE';
"[" Devuelve el token 'CORCHETEABRE';
"]" Devuelve el token 'CORCHETECIERRE';
"{" Devuelve el token 'LLAVEABRE';
"}" Devuelve el token 'LLAVECIERRE';
";" Devuelve el token 'PTCOMA';
":" Devuelve el token 'DOSPT';
"," Devuelve el token 'COMA';
"&&" Devuelve el token 'AND';
"||" Devuelve el token 'OR';
"?" Devuelve el token 'TERNARIO';
```

Para el análisis sintáctico se utilizó la siguiente gramática:

```
ini //Produccion Inicial
    :Instrucciones EOF
    ;

Instrucciones //Produccion para generar una Lista de Instrucciones
    : TipoInstruccion
    | Instrucciones TipoInstruccion
;
```

```
TipoInstruccion //Prorduccion para Reconocer el tipo de Instruccion
    : Declaraciones
    | Inicializacion PTCOMA
    | Print
    | If
    | Ciclo
    | Break
    | Switch
    | LlamadoFuncion
    | Run
    | FuncMetod
    | error PTCOMA //Recuperacion con punto y coma
;

TipoVar // Produccion para reconocer el tipo de dato
    : INT
    | DOUBLE
    | BOOLEAN
    | CHAR
    | STRING
;

Declaraciones //Produccion para realizar los diversos tipos
    : TipoVar Variables
    | TipoVar IDENTIFICADOR CORCHETEABRE CORCHETECIERRE CORCHETEABRE CORCHETECIERRE ASIGNACION CORCHETEABRE ListaVectores CORCHETECIERRE PTCOMA
    | TipoVar IDENTIFICADOR CORCHETEABRE CORCHETECIERRE CORCHETEABRE CORCHETECIERRE ASIGNACION NEW TipoVar CORCHETEABRE Valor CORCHETECIERRE CORCHETEABRE Valor CORCHETECIERRE PTCOMA
    | TipoVar IDENTIFICADOR CORCHETEABRE CORCHETECIERRE ASIGNACION CORCHETEABRE ListaValores CORCHETECIERRE PTCOMA
    | TipoVar IDENTIFICADOR CORCHETEABRE CORCHETECIERRE ASIGNACION NEW TipoVar CORCHETEABRE Valor CORCHETECIERRE PTCOMA
    | TipoVar IDENTIFICADOR CORCHETEABRE CORCHETECIERRE ASIGNACION TOCHARARRAY PARABRE Valor PARCIERRE PTCOMA
;

ListaValores //Produccion para generar una lista de valores
    : ListaValores COMA Valor
    | Valor
;

ListaVectores //Produccion para generar una matriz
    : ListaVectores COMA CORCHETEABRE ListaValores CORCHETECIERRE
    | CORCHETEABRE ListaValores CORCHETECIERRE
;

Inicializacion //Produccion para generar las inicializaciones de variables
    : IDENTIFICADOR INCREMENTO
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE INCREMENTO
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE CORCHETEABRE Valor CORCHETECIERRE INCREMENTO
    | IDENTIFICADOR DECREMENTO
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE DECREMENTO
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE CORCHETEABRE Valor CORCHETECIERRE DECREMENTO
    | IDENTIFICADOR ASIGNACION Valor
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE ASIGNACION Valor
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE CORCHETEABRE Valor CORCHETECIERRE ASIGNACION Valor
;


Variables // Produccion para declarar variables primitivas
    : Variables2 ASIGNACION Valor PTCOMA
    | Variables2 PTCOMA
;

Variables2 // Produccion para generar una lista de identificadores
    : Variables2 COMA IDENTIFICADOR
    | IDENTIFICADOR
;
```

```
Valor //Produccion para generar todas las operaciones algebraicas o relacionales
    : RESTA Valor
    | Valor POTENCIA Valor
    | Valor MULTIPLICACION Valor
    | Valor DIVISION Valor
    | Valor SUMA Valor
    | Valor RESTA Valor
    | Valor MODULO Valor
    | Valor IGUAL Valor
    | Valor DIFERENTE Valor
    | Valor MENOR Valor
    | Valor MENORIGUAL Valor
    | Valor MAYOR Valor
    | Valor MAYORIGUAL Valor
    | Valor OR Valor
    | Valor AND Valor
    | Valor INCREMENTO
    | Valor DECREMENTO
    | Valor TERNARIO Valor DOSPT Valor
    | NEGACION Valor
    | PARABRE Valor PARCIERRE
    | ENTERO
    | DECIMAL
    | CADENA
    | CARACTER
    | TRUE
    | FALSE
    | IDENTIFICADOR
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE CORCHETEABRE Valor CORCHETECIERRE
    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE

    | IDENTIFICADOR CORCHETEABRE Valor CORCHETECIERRE |
    | PARABRE TipoVar PARCIERRE Valor %prec CAST1
    | TOSTRING PARABRE Valor PARCIERRE
    | LENGTH PARABRE Valor PARCIERRE
    | TOLOWER PARABRE Valor PARCIERRE
    | TOUPPER PARABRE Valor PARCIERRE
    | TYPEOF PARABRE Valor PARCIERRE
    | ROUND PARABRE Valor PARCIERRE
    | IDENTIFICADOR PARABRE PARCIERRE
    | IDENTIFICADOR PARABRE ListaValores PARCIERRE
;

Print //Produccion para el reconocimiento de la funcion Print
    : PRINT PARABRE Valor PARCIERRE PTCOMA
    | PRINT PARABRE PARCIERRE PTCOMA
    | PRINTLN PARABRE Valor PARCIERRE PTCOMA
    | PRINTLN PARABRE PARCIERRE PTCOMA
;

If //Produccion para el reconocimiento de la estructura de control IF
    : IF PARABRE Valor PARCIERRE Entorno Else
;

Else //Produccion para el reconocimiento de la estructura de control ELSE/ElSE IF
    : ELSE Entorno
    | ELSE If
    | Epsilon
;

Ciclo //Produccion para el reconocimiento de la estructura FOR/WHILE/DO WHILE
    : FOR PARABRE Param1 Valor PTCOMA Inicializacion PARCIERRE Entorno
    | WHILE PARABRE Valor PARCIERRE Entorno
    | DO Entorno WHILE PARABRE Valor PARCIERRE PTCOMA
;

Param1 //Produccion para reconocer el primer parametro del ciclo for
    : Declaraciones
    | Inicializacion PTCOMA
;

Switch //Produccion para reconocer la estructura de control SWITCH
    : SWITCH PARABRE Valor PARCIERRE LLAVEABRE EntornoS LLAVECIERRE
;

EntornoS //Produccion para reconocer el entorno de trabajo de la estructura SWITCH
    : Casos DEFAULT DOSPT Instrucciones
    | Casos
    | DEFAULT DOSPT Instrucciones
    | DEFAULT DOSPT
;

Casos //Produccion para reconocer los Cases de un Switch
    : CASE Valor DOSPT Instrucciones
    | Casos CASE Valor DOSPT Instrucciones
;
```

```
Entorno //Produccion para reconocer datos entre llaves
    : LLAVEABRE Instrucciones LLAVECIERRE
    | LLAVEABRE LLAVECIERRE
;

Break //Produccion para reconocer las instrucciones BREAK/ CONTINUE/ RETURN
    : BREAK PTCOMA
    | CONTINUE PTCOMA
    | RETURN PTCOMA
    | RETURN Valor PTCOMA
;

FuncMetod //Prouccion para reconocer las funciones y metodos
    : IDENTIFICADOR PARABRE ListaParametros PARCIERRE TipoFunc LLAVEABRE Instrucciones LLAVECIERRE
    | IDENTIFICADOR PARABRE PARCIERRE TipoFunc LLAVEABRE Instrucciones LLAVECIERRE
;

ListaParametros //Produccion para reconocer la lista de parametros
    : ListaParametros COMA TipoVar IDENTIFICADOR
    | TipoVar IDENTIFICADOR
;

Run //Produccion para reconocer las operaciones Run
    : RUN LlamadoFuncion
;

LlamadoFuncion //Produccion para reconocer
    : IDENTIFICADOR PARABRE ListaValores PARCIERRE PTCOMA
    | IDENTIFICADOR PARABRE PARCIERRE PTCOMA
;
```

Jison

Para la creación ejecución de la gramática se utilizó la librería jison. Es así como se creo el archivo Grammar.jison con el cual se ejecuta el análisis léxico y sintáctico del código que sea enviado desde el frontend. Así mismo es aquí donde se generan los objetos que servirán para la creación de la tabla de Errores, el árbol de análisis sintáctico y la tabla de símbolos.

```
const {Aritmetica} = require('../Expresion/aritmetica')
const {Relacional} = require('../Expresion/Relacional')
const {Negacion} = require('../Expresion/Negacion')
const {Ternario} = require('../Expresion/Ternario')
const {TOString, TOLower, TOUpper, LENGHT, LENGHT2, TypeOF, Redondear} = require('../Expresion/TO')
const {VectorDec1, VectorDec2, VectorDec3, MatrizDec1, MatrizDec2, InicializacionV, InicializacionM}
const {Casteo} = require('../Expresion/Casteo')
const {Literal} = require('../Expresion/Literal')
const {ErrorE} = require('../Error/Error')
const {If} = require('../Instruccion/If')
const {Ciclo} = require('../Instruccion/Ciclo')
const {BREAK, RETURN} = require('../Instruccion/BreakContinue')
const {Funcion, LlamadoFuncion, Run} = require('../Instruccion/Funcion')
const {Entorno,EntornoI, EntornoC, EntornoD, EntornoCase, EntornoW, EntornoF} = require('../Instrucci
const {Declaracion, Inicializacion} = require('../Instruccion/Declaracion')
const {Switch} = require('../Instruccion/Switch')
const {Llamado, LlamadoM, LlamadoV, LlamadoFuncionE} = require('../Expresion/Llamado')
const {Print, Println} = require('../Instruccion/Print')
const {Arbol} = require('../Extra/Arbol')
Errores = []
exports.Errores = Errores
Impresion = ""
exports.Impresion = Impresion
arbol = new Arbol();
exports.arbol = arbol

TablaSimbolos =[]
exports.TablaSimbolos = TablaSimbolos
```

Importaciones y Objetos utilizados durante el análisis del archivo

Backend

Para la creación del backend se utilizaron archivos TypeScript que posteriormente se compilaron para obtener sus equivalentes en JavaScirpt.

Controller.js

Este archivo es utilizado para procesar todas las peticiones realizadas por el frontend retornando archivos json con la información de repuesta según sea el caso.

```
const { Ambito } = require('../src/Interprete/Extra/Ambito');
const { Funcion, LlamadoFuncion } = require('../src/Interprete/Instruccion/Funcion');
var parser = require('../src/Interprete/Grammar/grammar');
var tabla = "<table class='table table-hover'><thead class='thead-dark'><tr><th>Linea</th><th>Columna</th><th>
var tablaS = "<table class='table table-hover'><thead class='thead-dark'><tr><th>Entorno</th><th>Nombre</th><t
var exec = require('child_process');
var fs = require('fs');
const { ErrorE } = require('../src/Interprete/Error/Error');
const { Declaracion, Inicializacion } = require('../src/Interprete/Instruccion/Declaracion');
const { MatrizDec1, VectorDec1, InicializacionM, InicializacionV, MatrizDec2, VectorDec2, VectorDec3 } = requi
var ContenidoEditor = { Codigo: "", Error: "" }
var Dot = "digraph G{}";
```

Importaciones y variables globales de controller.js

```
exports.index = async (req, res) => {
    res.send({ "Controlador": "Estuardo" })
}

exports.ingresarCodigo = async (req, res) => {···
}

exports.CodigoIngresado = async (req, res) => {···
}

exports.ReporteErrores = async (req, res) => {···
}

exports.ReporteAST = async (req, res) => {···
}

exports.ReporteSimbolos = async (req, res) => {···
```

Funciones de procesamiento de solicitudes controllr.js

- **Función index:** Función que retorna el mensaje que contiene el nombre del elaborador.
- **Función ingresarCodigo:** Función que realiza procesamiento del código, así como el análisis léxico, sintáctico y semánticos; incluyendo la ejecución por corridas de las instrucciones del programa. Al finalizar retorna un mensaje de aceptación u error dependiendo si la información procesada correctamente o no.
- **Función CodigoIngresado:** Función que retorna, en formato json, el resultado de la ejecución enviada a ingresarCodigo.
- **Función ReporteErrores:** Función que retorna, en formato json, una cadena string que contiene la información para crear una tabla html con todos los errores encontrados durante la ejecución del programa.
- **Función ReporteAST:** Función que retorna, en formato json, una respuesta de ok o error durante la creación de un archivo dot el cual se convierte en svg para mostrarse en el frontend.
- **Función ReporteSimbolos:** Función que retorna, en formato json, una cadena string que contiene la información para crear una tabla html con todos las variables o funciones declaradas durante la ejecución del programa.

Enrutador.js

Archivo que hace las relaciones y redirecciones para enrutar el frontend con el backend, dependiendo de la solicitud realizada por el frontend.

```
const express = require('express')
const router = express.Router()
const control = require("../Controllers/controller")

router.get("/cliente", control.index)

router.post("/Codigo", control.ingresarCodigo)

router.get("/Codigo", control.CodigoIngresado)

router.get("/Error", control.ReporteErrores)

router.get("/AST", control.ReporteAST)

router.get("/Simbolos", control.ReporteSimbolos)

module.exports = router
```

Código de Enrutador.js

Clases Error

Clase para la creación de objetos de tipo ErrorE.

```
export class ErrorE{
    constructor(public linea: number, public columna: number, public tipo: string, public mensaje: string) {

    }
}
```

Código de la clase ErrorE

Clase Expresion

Clase abstracta que se utiliza para la creación y como base para todas las clases que se derivan de esta para la realización de operaciones algebraicas o relacionales.

```
import {Retorno, tipoSUMA, tipoRESTA, tipoDIV, tipoMULTI, tipoMODULO, tipoPOTENCIA, Type, tipoRELACIONAL, tipo
import {Operador} from "./Aritmetica";
import { Ambito } from "../Extra/Ambito";

export abstract class Expresion {
    public linea: number;
    public columna: number;

    constructor(linea: number, columna: number){
        this.linea = linea;
        this.columna = columna;
    }
```

Importaciones y constructor de la clase Expresion

```
public abstract ejecutar(ambito: Ambito): Retorno;

public tipoDominante(Tipo: Operador, fila: Type, columna: Type){···
}
```

Funciones de la clase Expresion

- **Función ejecutar:** Función abstracta que debe ser implementada en todas las clases que hereden de esta clase.

- **Función tipoDominante:** Función que retorna un valor de la clase Type, después de haber hecho una comparación en una matriz designada según sus parámetros.

Clase Type

Clase enumerada que indica los tipos de datos primitivos aceptados por el lenguaje.

```
export enum Type {
    INT = 0,
    DOUBLE = 1,
    BOOLEAN = 2,
    CHAR = 3,
    STRING = 4
}
```

Clase enumerada Type

Objeto Retorno

Es un diccionario que almacena el tipo y valor que almacene un objeto que herede de la clase Expresion.

```
export type Retorno = {
    value: any,
    type: Type
}
```

Diccionario Retorno

Matrices de comprobación tipo**

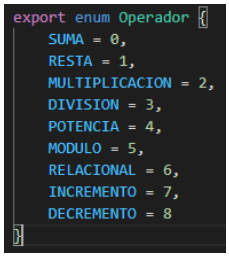
Son matrices que se utilizan para saber cual es el tipo dominante resultado de una operación aritmética o lógica.

```
export const tipoSUMA = [···
]

export const tipoRESTA = [···
]

export const tipoMULTI = [···
]

export const tipoDIV = [···
]

export const tipoPOTENCIA = [···
]

export const tipoMODULO = [···
]

export const tipoRELACIONAL = [···
]

export const tipoINCDEC = [···
]
```

Matrices para verificación de tipo dominante

Clase Operador

Clase enumerada utilizada para listar los tipos de operadores aritméticos y relacionales que existen.



Clase enumerada Operador

Clase Aritmetica

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se realice una operación aritmética, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.

Clase Aritmetica

Clase Cateo

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se haga el casteo de una variable, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.

Clase Casteo

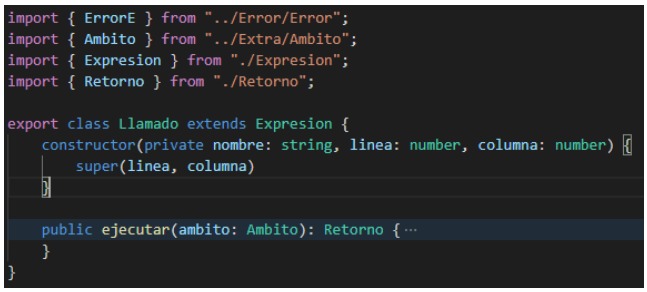
Clase Literal

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se obtenga un valor de String, entero, decimal, carácter o boleano; en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.

Clase Literal

Clase Llamado

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se obtenga un identificador de una variable, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.



Clase Llamado

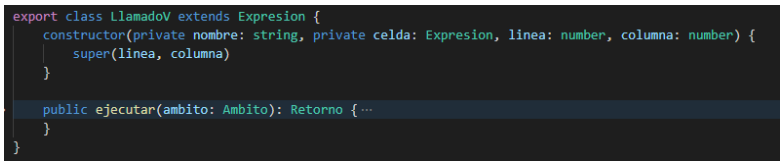
Clase LlamadoM

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se obtenga un identificador de una matriz, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.

Clase LlamadoM

Clase LlamadoV

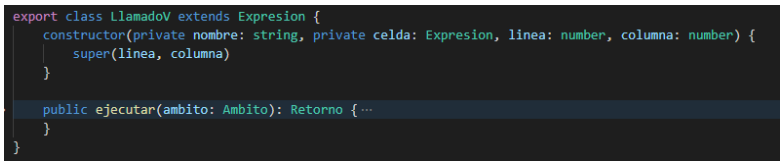
Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se obtenga un identificador de una vector, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.



Clase LlamadoV

Clase LlamadoFuncionE

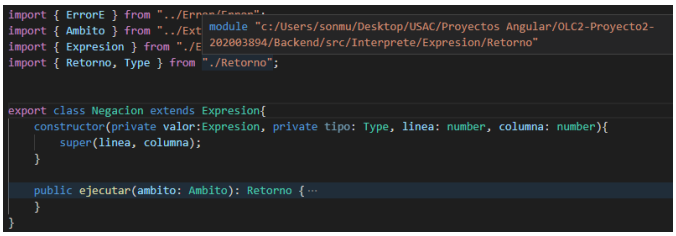
Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se obtenga un identificador de una función o método, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.



Clase LlamadoFuncionE

Clase Negacion

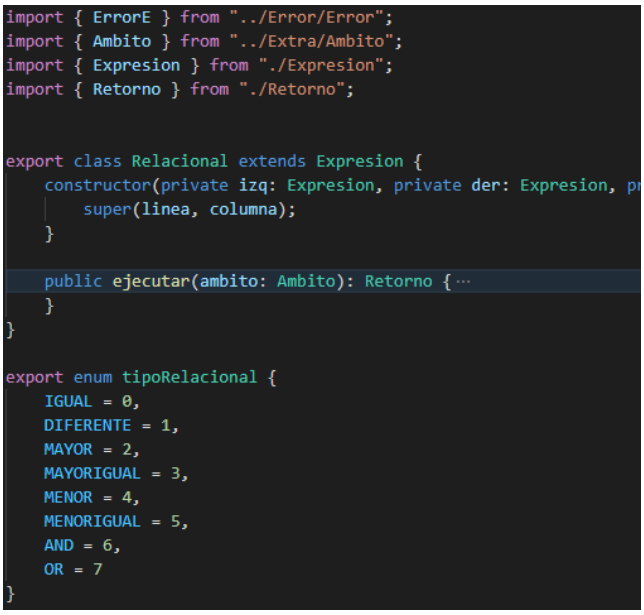
Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se desee realizar la negación de un valor booleano, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.



Clase Negacion

Clase Relacional

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se desee realizar una operación realacional, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.



Clase Relacional

Clase Ternario

Clase que hereda de la clase Expresion, se utiliza para procesar las producciones en las que se desee realizar una operación ternaria, en caso que exista un error en su ejecución retorna un objeto ErrorE para su almacenamiento, en caso contrario retorna un objeto retorno con el resultado de la operación.

Clase Ternario

TO.ts

En este archivo se encuentran las clases para la elaboración de todas las funciones internas del programa como lo es lenght o toUpper.

Archivo TO.ts

Clase Instruccion

Clase abstracta que se utiliza para la creación y como base para todas las clases que se derivan de esta para la realización de las instrucciones que se encuentran la entrada analizada.

```
import {Ambito} from "../Extra/Ambito";

export abstract class Instruccion{
    constructor(public linea:number, public columna:number){
    }
```

Importaciones y constructor de la clase Instruccion

```
public abstract ejecutar(ambito: Ambito);
```

Funciones de la clase Expresion

- **Función ejecutar:** Función abstracta que debe ser implementada en todas las clases que hereden de esta clase.

ARRAYyMATRIZ.ts

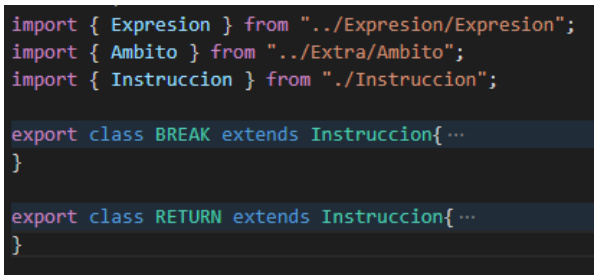
En este archivo se encuentran las clases para la elaboración de todos los tipos de declaración de arreglos o matrices.

```
import { ErrorE } from "../Error/Error";
import { Ambito } from "../Extra/Ambito";
import { Expresion } from "../Expresion/Expresion";
import { Type } from "../Expresion/Retorno";
import { Instruccion } from "./Instruccion";


export class VectorDec1 extends Instruccion {···
}
export class VectorDec3 extends Instruccion {···
}

export class VectorDec2 extends Instruccion {···
}

export class MatrizDec1 extends Instruccion {···
}

export class MatrizDec2 extends Instruccion {···
}

export class InicializacionV extends Instruccion {···
}

export class InicializacionM extends Instruccion {···
}
```

Archivo ARRAYyMATRIZ.ts

BreakContinue.ts

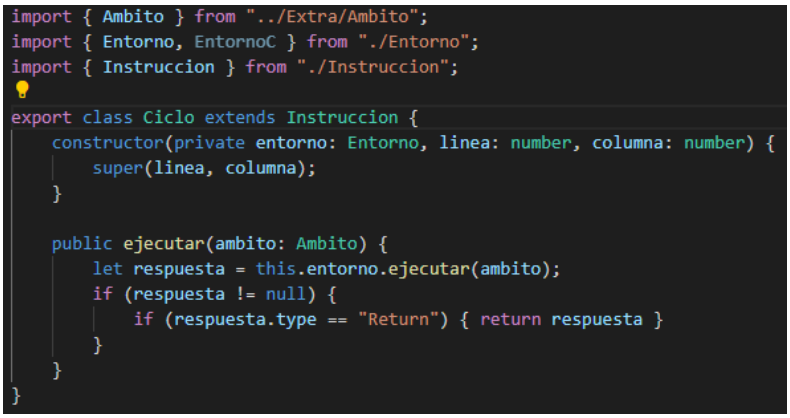
En este archivo se encuentran las clases para la ejecución de las instrucciones break, continue o return.



BreakContinue.ts

Ciclo.ts

En este archivo se encuentran las clases para la ejecución de ciclos.



Archivo Ciclo.ts

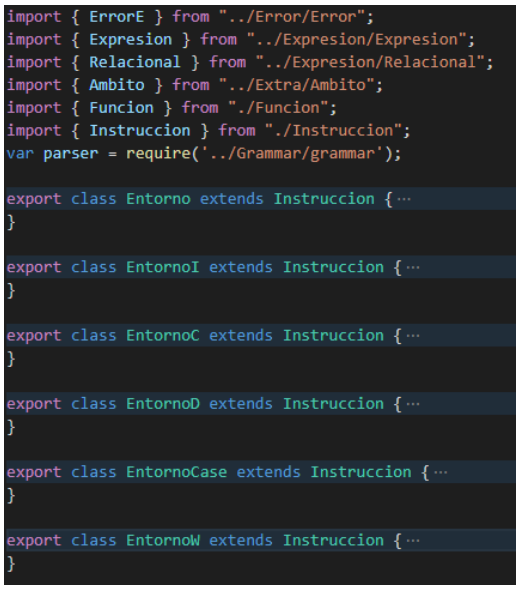
Declaracion.ts

En este archivo se encuentran las clases para la ejecución de las declaraciones e inicializaciones de variables cuyo dato sea de tipo primitivo.

Archivo Declaracion.ts

Declaracion.ts

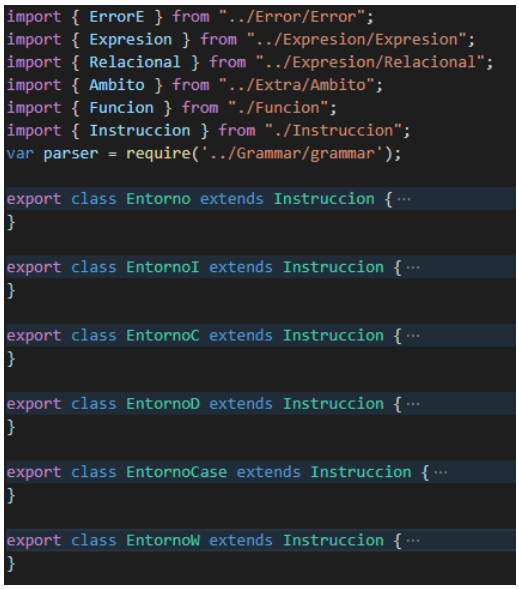
En este archivo se encuentran las clases para la ejecución de los entornos de cada instrucción que lo requiera como ciclos, instrucciones if o switch; o funciones.



Archivo Entorno.ts

Ambito.ts

En este archivo se encuentran las clases para la ejecución de los entornos de cada instrucción que lo requiera como ciclos, instrucciones if o switch; o funciones.



Archivo Funcion.ts

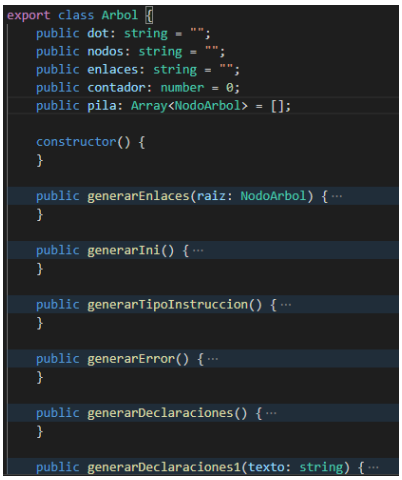
Funcion.ts

En este archivo se encuentran las clases para la creación de los diferentes ámbitos en los que se desenvuelven cada una de las instrucciones del programa, estos ámbitos cuentan con un mapa de variables y funciones.

Archivo Ambito.ts

Arbol.ts

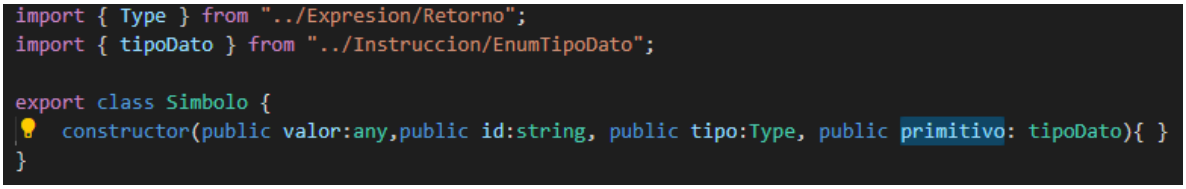
En este archivo se encuentran las clases para la ejecución del árbol de análisis sintáctico y cada uno de sus nodos.



Archivo Arbol.ts

Simbolo.ts

En este archivo se encuentran las clase para la creación de objetos Simbolos que conformaran la tabla de simbolos.



Archivo Simbolo.ts

Frontend

Para la creación del frontend de la aplicación se utilizo Angular. La misma esta formada de dos componentes los cuales son utilizados para visualizar los apartados de Editor y Reportes.

Editor

Para el HTML de editor se utilizó el siguiente diseño:

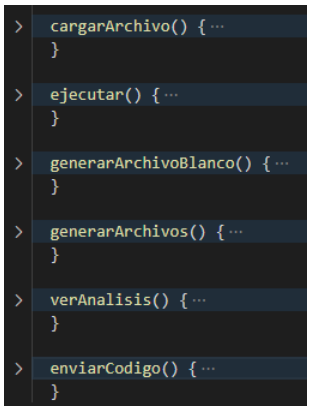
```
<div class="d-grid gap-5" style="grid-template-columns: 1fr 1fr 1fr 1fr 1fr; margin-top: 20px; margin-bottom:
  <input type="file" #file (change)="cargarArchivo()" class="btn btn-primary" accept=".cst">
  <input type="text" #nombreA name="nombreA" id="nombreA" placeholder="Nombre Archivo">
  <button class="btn btn-secondary" (click)="generarArchivoBlanco()" >Crear Archivo</button>
  <button class="btn btn-secondary" (click)="generarArchivos()">Guardar Archivo</button>
  <button class="btn btn-danger" (click)="ejecutar()">Ejecutar</button>
</div>

<div class="d-grid gap-3" style="grid-template-columns: 1fr 1fr; margin-top: 20px; margin-bottom: 20px;">
    <div class="bg-light border rounded-3">
      <div class="form-group" style="background-color: rgb(41, 6, 9); color: white;" #entrada>
          <label for="entrada" style="font-size: 20px;">Editor</label>
          <textarea #codigo id="entrada" class="form-control" name="" rows="10"></textarea>
      </div>
    </div>
    <div class="bg-light border rounded-3">
      <div class="form-group" style="background-color: rgb(41, 6, 9);">
          <label for="salida" style="font-size: 20px; color: white;">Consola</label>
          <textarea id="salida" class="form-control" name="" rows="10" disabled>{{salida}}</textarea>
      </div>
    </div>
</div>
```

HTLM componente Editor

Para su funcionalidad se utilizó el archivo ts que se crea por defecto.

```
import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';
import { BackendService } from '../services/backend.service';

@Component({
  selector: 'app-editor',
  templateUrl: './editor.component.html',
  styleUrls: ['./editor.component.css']
})
export class EditorComponent implements OnInit {
  public salida: string;
  public contenido: string;
  public guardar: string;

  constructor(private backend: BackendService) {
    this.salida = "";
    this.guardar = "";
  }

  ngOnInit(): void {
  }

  @ViewChild('file') archivo: ElementRef;
  @ViewChild('codigo') codigo: ElementRef;
  @ViewChild('nombreA') nombreA: ElementRef;
```

Parte del código TS del componente Editor



Funciones creadas para el componente Editor

- **Función cargarArchivo:** Esta función se acciona al cargarse un archivo en cst en para su lectura en el programa y mostrarlo en un TextArea.
- **Función ejecutar:** Esta función se acciona al presionar el botón ejecutar. Esta función pone en ejecución la función enviarCodigo.
- **Función generarArchivoBlanco:** Esta función se acciona al presionar el botón Archivo Blanco, lo que ocasionara que se descargue un archivo cst que posee de nombre el texto que se ingrese en el input Text.
- **Función verAnalisis:** Esta función se acciona al finalizar la función enviarCodigo. Muestra en un textArea el resultado del análisis y ejecución del condigo enviado al servidor del Backend.
- **Función enviarCodigo:** Esta función envía el texto contenido en el textArea del Editor hacía el servidor del backend donde será analizado y ejecutado.

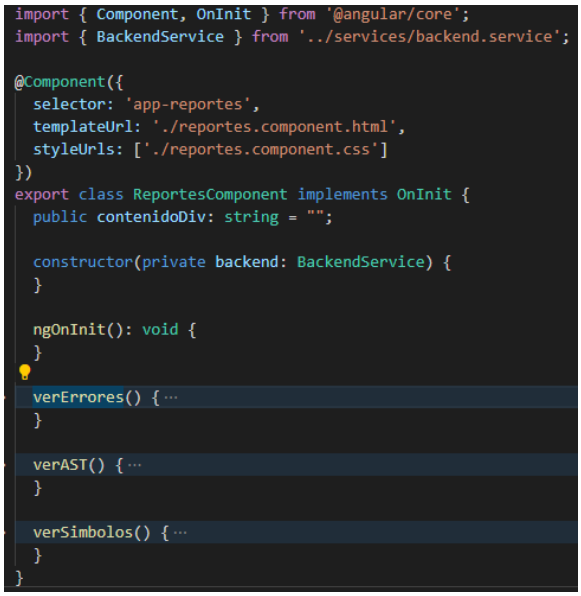
Reportes

Para el HTML de reportes se utilizó el siguiente diseño:

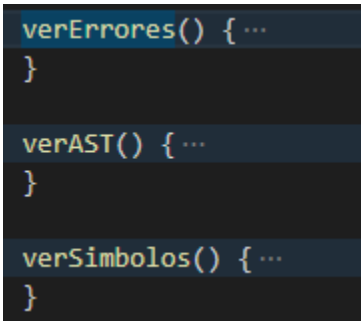
```
<div class="d-grid gap-3" style="grid-template-columns: 1fr 1fr 1fr; margin-top: 20px; margin-bottom: 20px;">
    <button class="btn btn-secondary" (click)="verErrores()">Errores</button>
    <button class="btn btn-secondary" (click)="verSimbolos()">Tabla de Simbolos</button>
    <button class="btn btn-secondary" (click)="verAST()">AST</button>
</div>

<div id="graph" [innerHTML]="contenidoDiv" style="text-align: center; width: auto; height: 400px; border: 1px
```

HTLM componente Editor

Para su funcionalidad se utilizó el archivo ts que se crea por defecto.



Parte del código TS del componente reportes



Funciones creadas para el componente Editor

- **Función verErrores:** Esta función se acciona al presionar el botón ver Errores, lo cual ocasionara que se vea en pantalla una tabla que contiene todos los errores léxicos, sintácticos y semánticos encontradas durante el análisis del archivo.
- **Función verAST:** Esta función se acciona al presionar el botón AST, lo cual ocasionara que se vea en pantalla un grafo con todas las instrucciones reconocidas durante el análisis sintáctico del archivo.
- **Función verSimbolos:** Esta función se acciona al presionar el botón Tabla Simbolos, lo cual ocasionara que se vea en pantalla una tabla con las características de las variables y funciones creadas durante la ejecución del código final.

Para la conexión entre el backend y el frontend se utilizaron los servicios de angular. El archivo creado fue backend.service.ts.

```
URL:string='http://127.0.0.1:4000';

ejecutar(codigo: any){
  return this.http.post(`${this.URL}/Codigo`,codigo)
}

obtenerAnalisis(){
  return this.http.get(`${this.URL}/Codigo`)
}

reporteError(){
  return this.http.get(`${this.URL}/Error`)
}

reporteAST(){
  return this.http.get(`${this.URL}/AST`)
}

reporteSimbolos(){
  return this.http.get(`${this.URL}/Simbolos`)
}
```

Funciones para realizar peticiones al Backend