MANUAL TÉCNICO

CompScript

29 DE ABRIL DE 2022.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA.
Organización de lenguajes y compiladores 1, Sección “B”.

Introducción:

El siguiente proyecto tiene como finalidad ser de utilidad para los estudiantes de introducción a la programación y computación 1 de la carrera de ingeniería en ciencias y sistemas de la universidad de San Carlos de Guatemala.

Fue desarrollado utilizando herramientas como: React, JavaScript, TypeScrip y el uso de librerías externas para el editor de código: Monaco editor, el cual es el mismo editor empleado en el motor de VsCode.

Se manejan temas como la abstracción y herencia de clases a otras clases que comparten algunos comportamientos tales como los bloques de sentencias if, else, while, switch, for, funciones, etc.

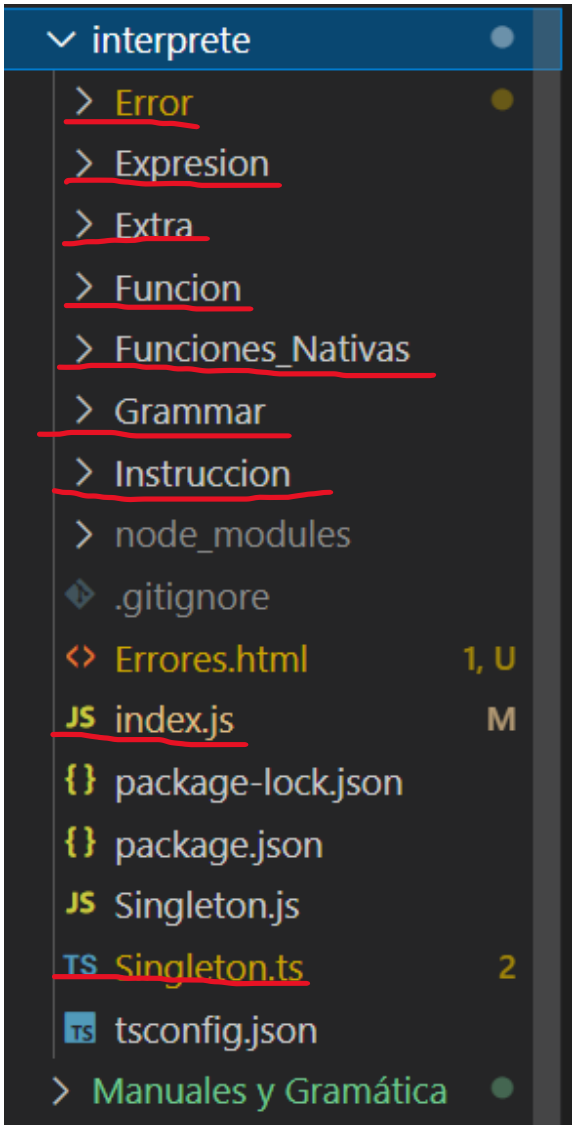
El programa consta de dos servidores principales, Backend y Frontend. En la parte del Backend se empleó la librería express y en la parte del Frontend se empleó la librería de React.

Descripción de funcionalidades:

Backend: La parte del backend es la que se encarga del análisis de todo el código recibido desde el frontend, es como tal el cerebro de la aplicación, y consta de los siguientes archivos:

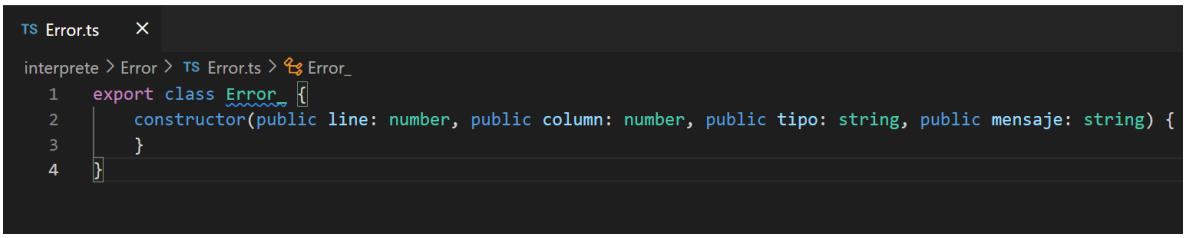
Una carpeta principal llamada ” interprete”, que es la encargada de contener todo el backend.

Al desplegar el menú contenido dentro de la carpeta “interprete” se observarán varias carpetas y varios archivos que son los encargados del correcto funcionamiento de la aplicación. Nos centraremos en los marcados a continuación:



Carpeta Error: Contiene un archivo con extensión .Ts y es la encargada de recibir los errores. Contiene los atributos siguientes:

- Tipo (Léxico, Sintáctico, Semántico)
- Mensaje (Descripción del error)
- Línea
- Columna



Carpeta expresión:

Esta carpeta contiene los siguientes archivos para el correcto funcionamiento de todas las expresiones evaluadas o guardadas en el proyecto:

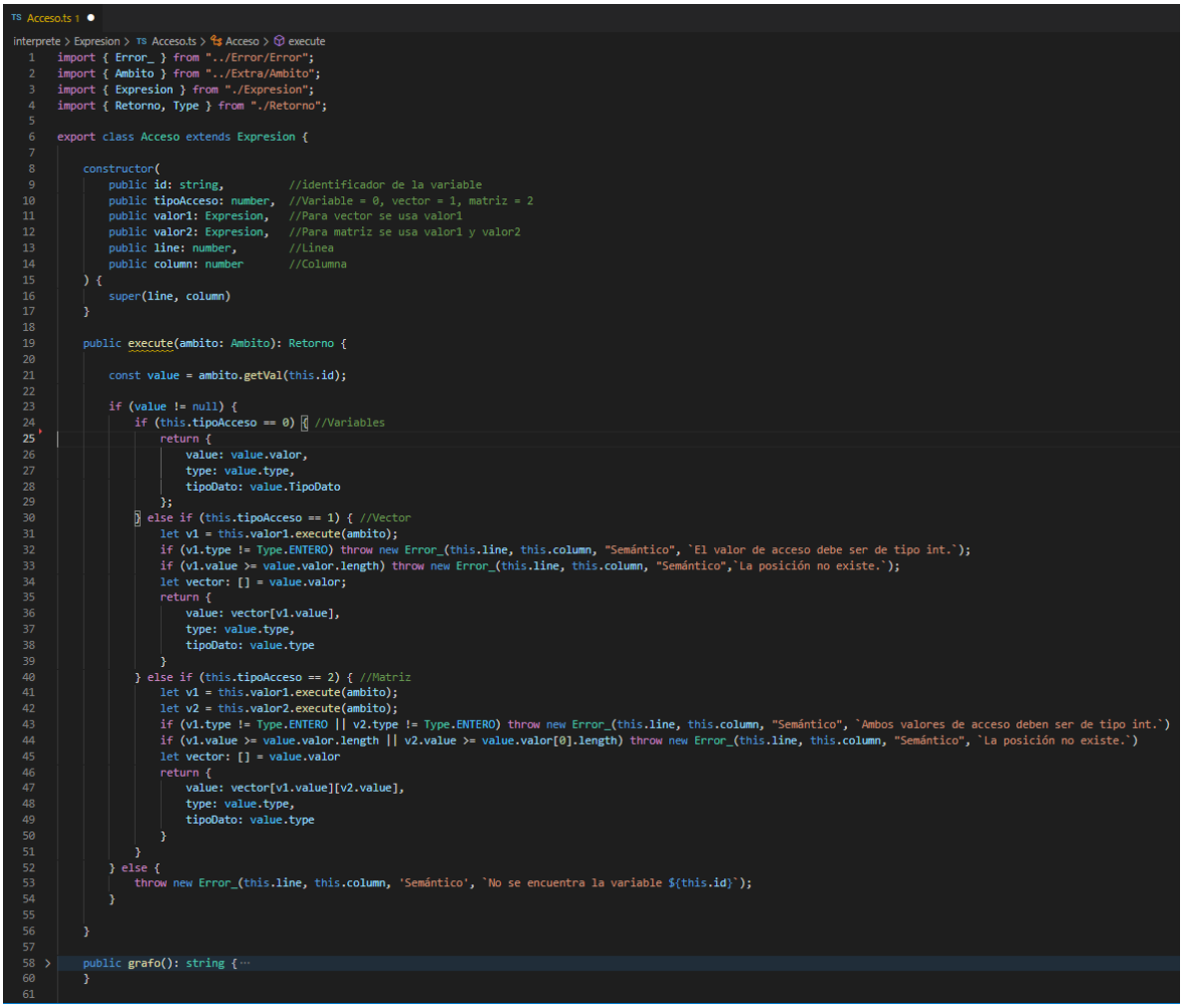
- Acceso
- Aritmética
- Expresión
- Literal
- Lógica
- Relacional
- Retorno

Acceso:

Es la encargada del acceso a las variables ya declaradas y retorna el valor de dicha variable. Recibe como parámetros:

- Id (Identificador).
- tipoAcceso (variable, vector, matriz).
- valor1 (utilizado para el vector y la matriz).
- valor2 (utilizado para la matriz).

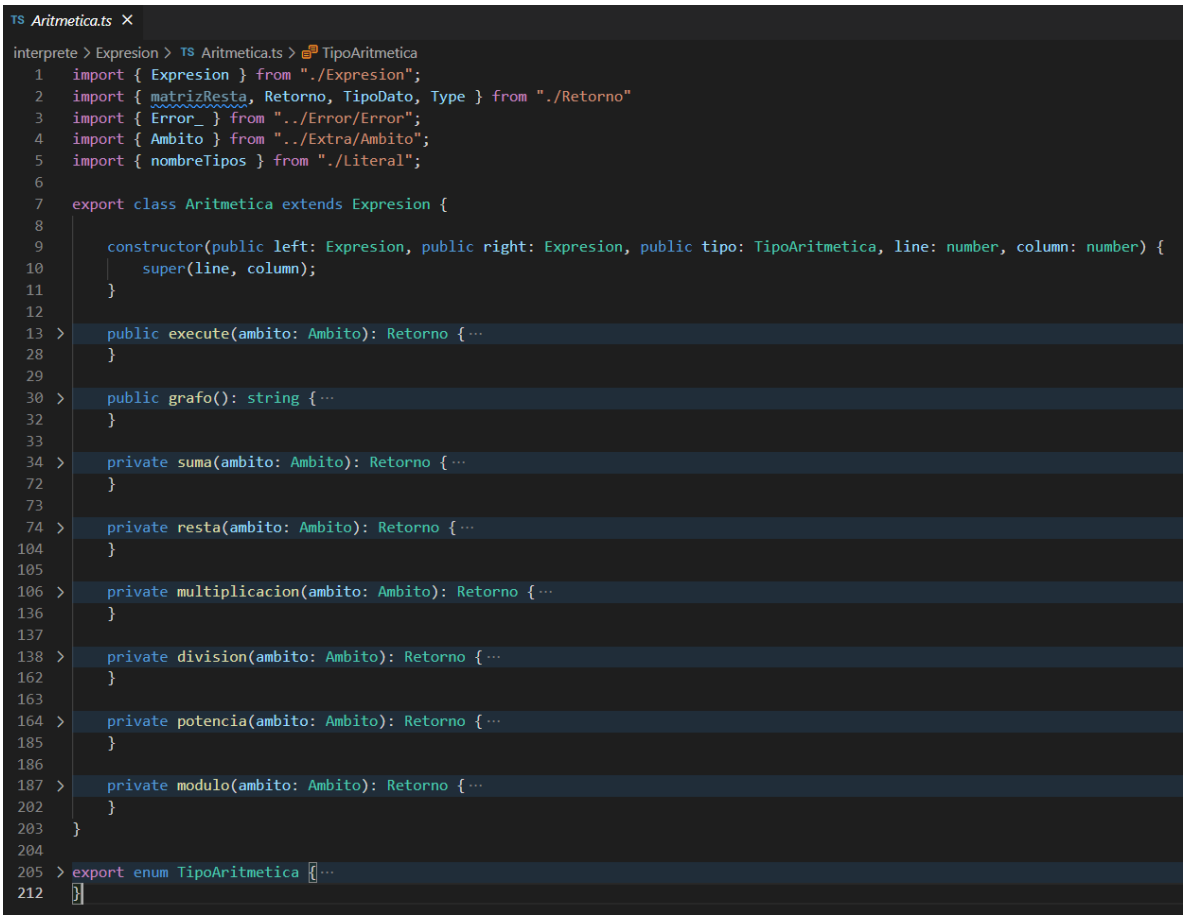
Consiste en verificar si la variable existe para luego retornarla, en el caso de los vectores y matrices funciona igual;



Aritmética:

Es la encargada de realizar todas las operaciones aritméticas tales como:

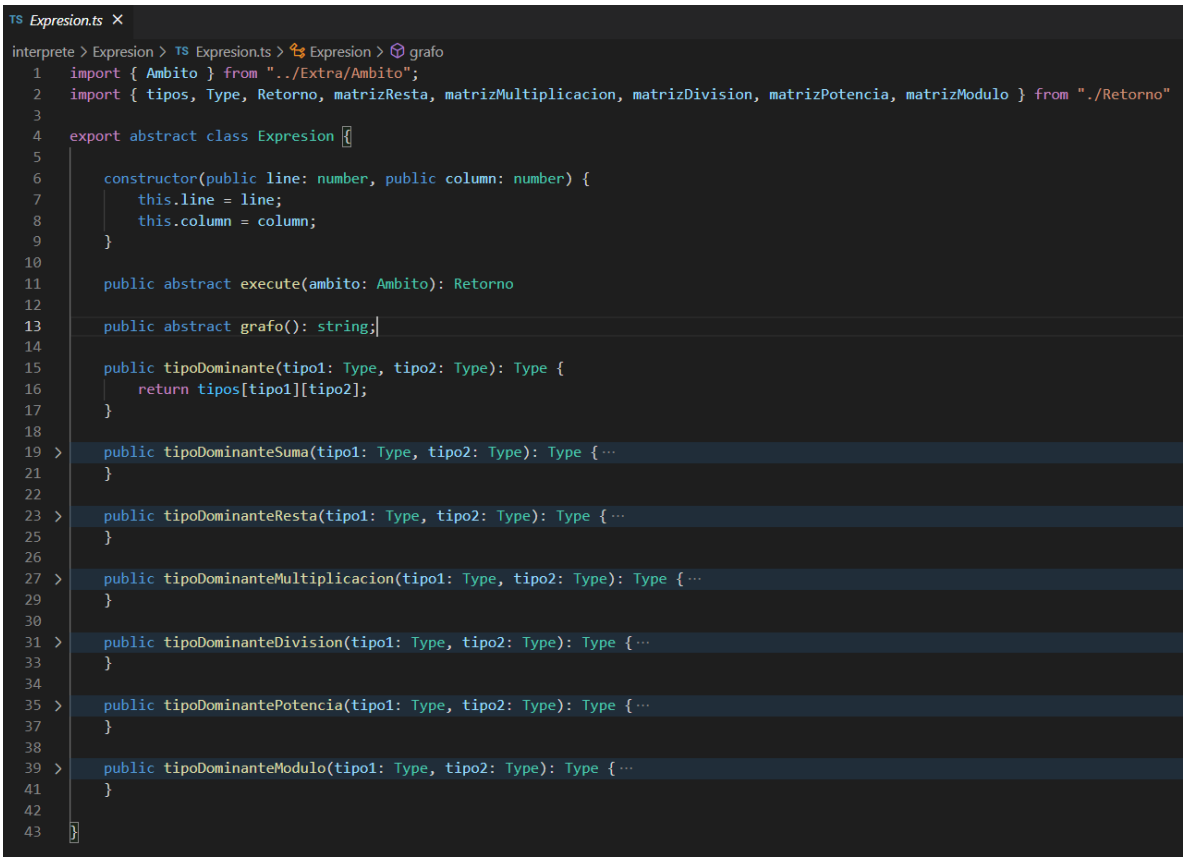
- Suma.
- Resta.
- Multiplicación.
- División.
- Potencia.
- Módulo.



Expresion:

Esta es una clase abstracta y es la encargada de heredar a todas las demás clases contenidas en la carpeta Expresion el método llamado execute, el cual inicializa los atributos y retorna: valor, tipo y el tipo de estructura. Recibe los siguientes parámetros:

- Line (Línea del archivo).
- Column (Columna del archivo.



Literal:

Es la encargada de obtener los valores de cada literal creado en el programa y retornar dichos valores para ser utilizados posteriormente. Recibe los siguientes parámetros:

- Value (valor del literal).
- Tipo (Tipo de literal).
- Line
- Column

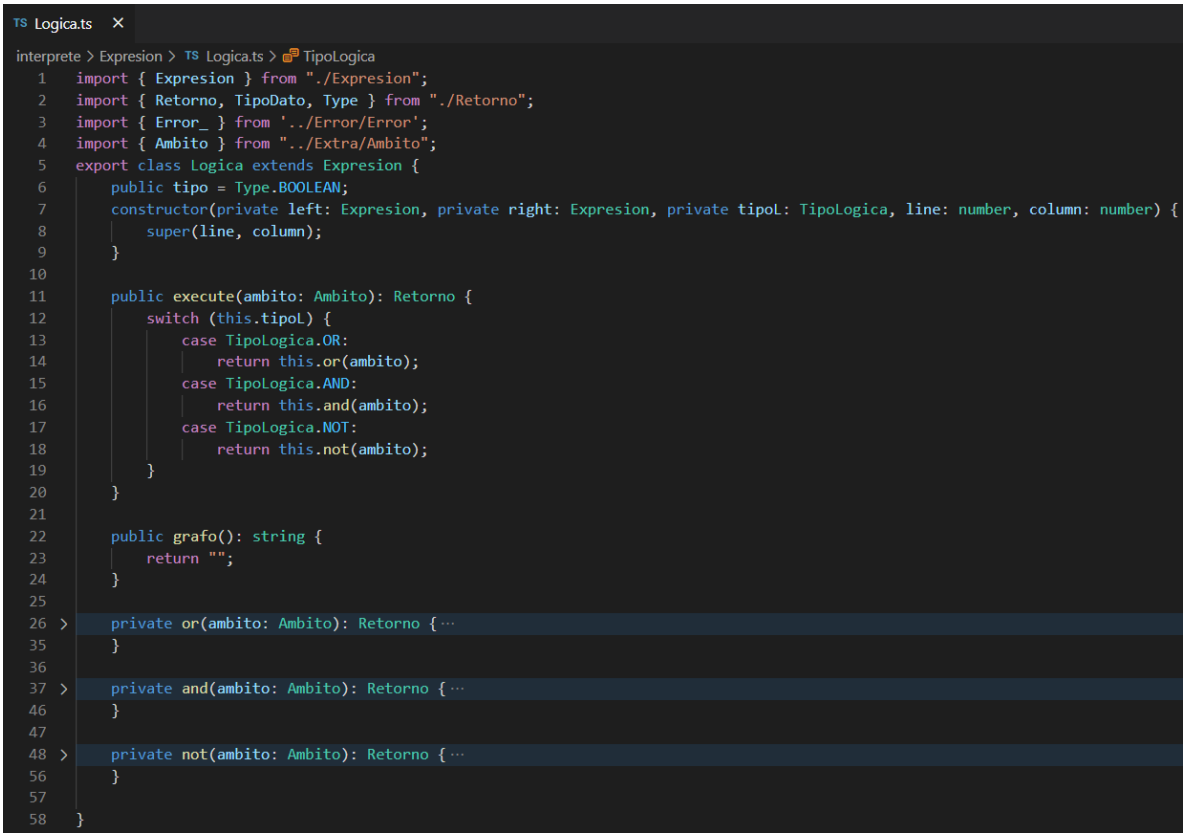
Lógica:

Es la clase encargada de realizar todas las operaciones lógicas tales como:

- OR (“O” lógico).
- AND (“Y” lógico).
- NOT (“NO” lógico).

Recibe los siguientes parámetros:

- Left (parte izquierda a evaluar).
- Rigth (parte derecha a evaluar).
- tipoL (tipo de expresión lógica).
- Line (línea).
- Column (Columna).



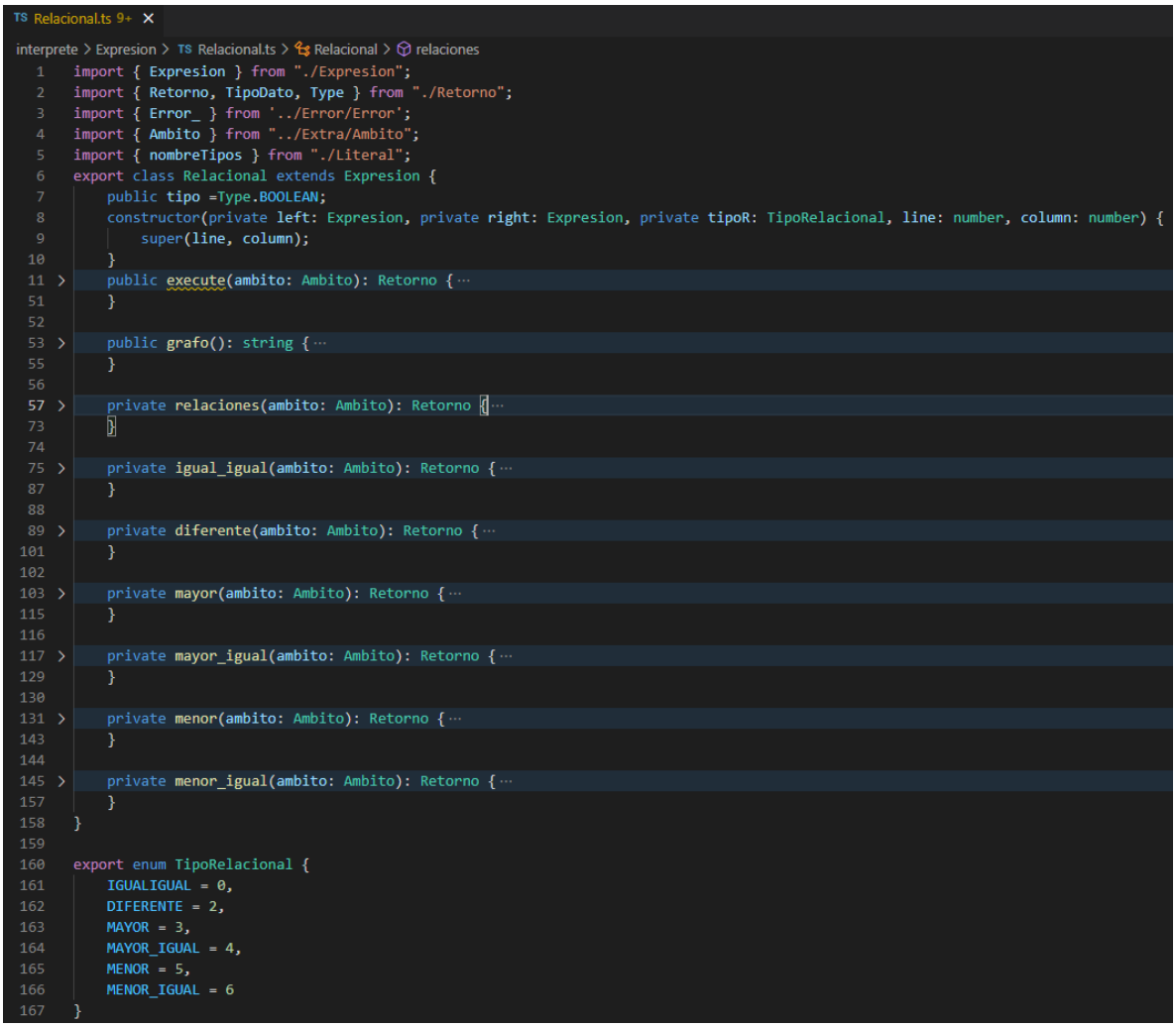
Relacional:

Es la encargada de realizar todas las operaciones relacionales tales como:

- Igual que
- Diferente de
- Mayor
- Mayor o igual
- Menor
- Menor o igual

Recibe como parámetros lo siguiente:

- Left (parte izquierda a evaluar).
- Right (parte derecha a evaluar).
- tipoR (tipo de operación relacional).
- Line.
- Column.



Retorno:

Es la encargada de exportar las matrices de tipos para cada operación a realizar dentro del programa, para poder permitir ciertas operaciones solo con ciertas expresiones.

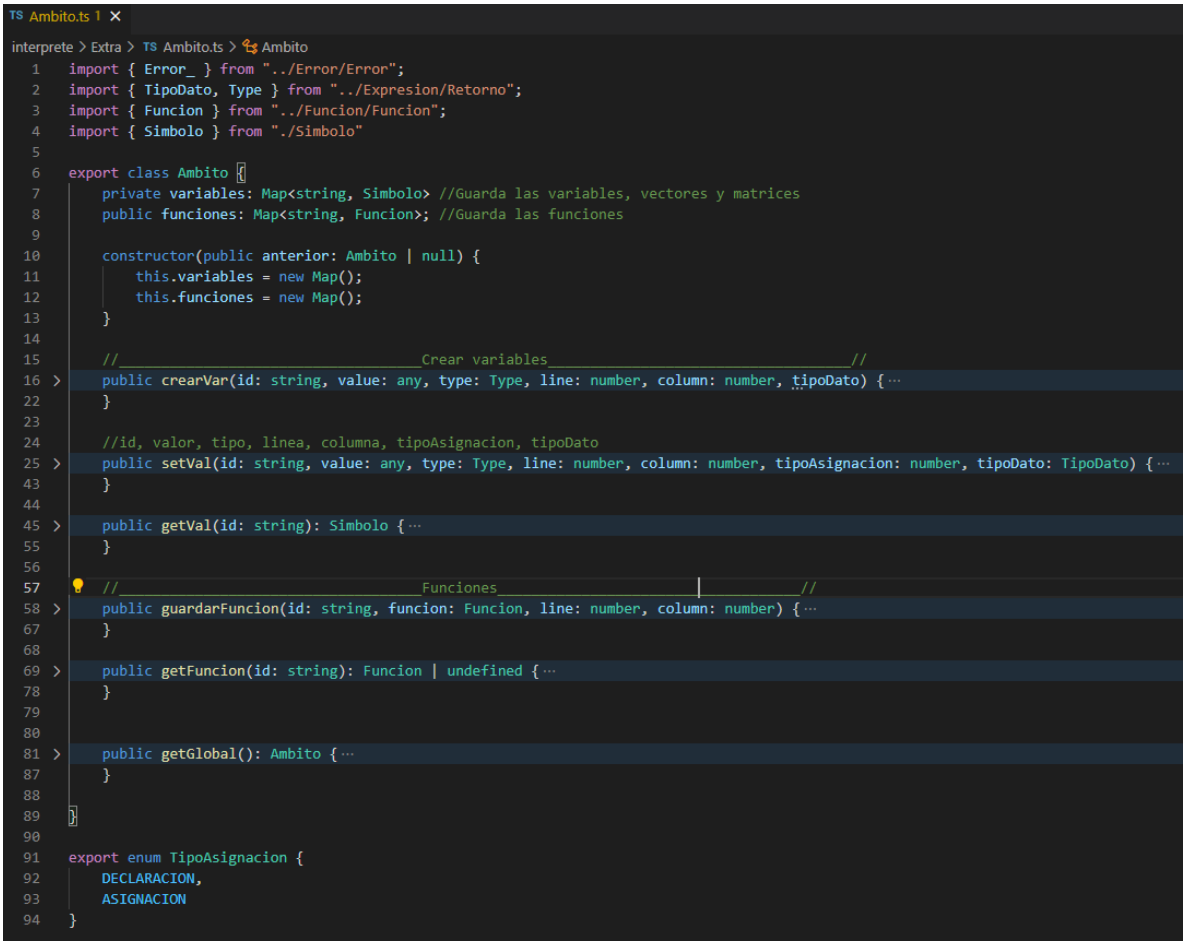
Carpeta Extra:

Esta carpeta contiene los siguientes archivos:

- Ámbito
- Símbolo

Ámbito:

Es el encargado de guardar todas las variables y funciones creadas durante la ejecución del programa. Hace las respectivas validaciones a la hora de guardar variables tales como verificar que otra variable con el mismo nombre no existe en el mismo entorno de ejecución.



Símbolo:

Es la encargada de almacenar en tiempo de ejecución todo lo que se ejecuta en la clase Ámbito. Es un objeto como tal, que sirve para guardar variables, funciones y vectores.

Recibe los siguientes parámetros:

- Valor
- Id
- Type
- TipoDato

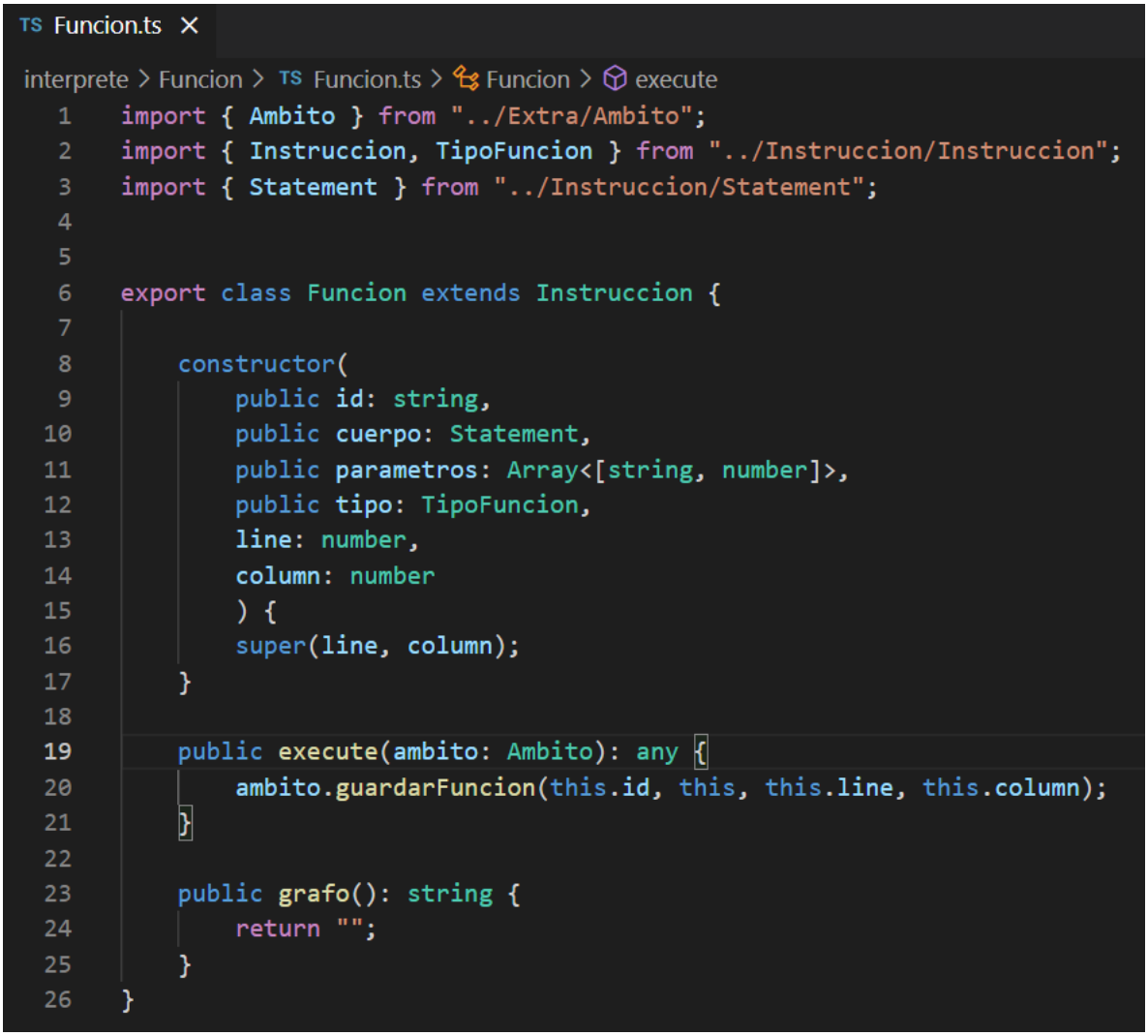
Carpeta Función:

Esta carpeta contiene los siguientes archivos:

- Función
- Llamada función
- Run

Función:

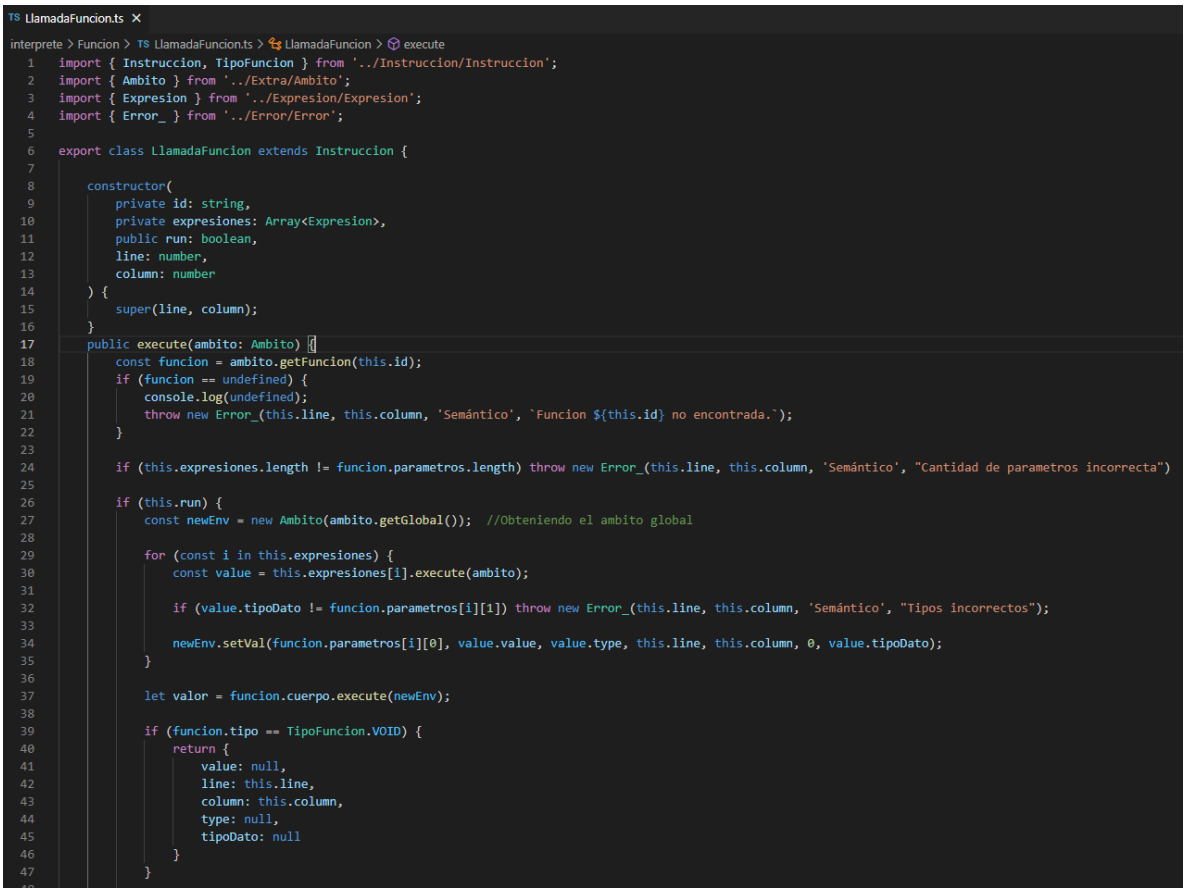
Esta clase no realiza ningún trabajo, solo se encarga de guardarse en el entorno global a ella misma.



LlamadaFuncion:

Es la encargada de buscar si existe la función a la que se está llamando y ejecutarla, también verifica que la cantidad de parámetros y el tipo de dato de cada parámetro sea el correcto. Recibe los siguientes parámetros:

- Id
- Expresiones (parámetros)
- Run (Si se debe ejecutar o no)
- Line
- Column



```ts
import { Instruccion, TipoFuncion } from '../Instruccion/Instruccion';
import { Ambito } from '../Extra/Ambito';
import { Expresion } from '../Expresion/Expresion';
import { Error_ } from '../Error/Error';

export class LlamadaFuncion extends Instruccion {

    constructor(
        private id: string,
        private expresiones: Array<Expresion>,
        public run: boolean,
        line: number,
        column: number
    ) {
        super(line, column);
    }
    public execute(ambito: Ambito) {
        const funcion = ambito.getFuncion(this.id);
        if (funcion == undefined) {
            console.log(undefined);
            throw new Error_(this.line, this.column, 'Semántico', `Funcion ${this.id} no encontrada.`);
        }

        if (this.expresiones.length != funcion.parametros.length) throw new Error_(this.line, this.column, 'Semántico', "Cantidad de parametros incorrecta")

        if (this.run) {
            const newEnv = new Ambito(ambito.getGlobal());  //Obteniendo el ambito global

            for (const i in this.expresiones) {
                const value = this.expresiones[i].execute(ambito);

                if (value.tipoDato != funcion.parametros[i][1]) throw new Error_(this.line, this.column, 'Semántico', "Tipos incorrectos");

                newEnv.setVal(funcion.parametros[i][0], value.value, value.type, this.line, this.column, 0, value.tipoDato);
            }

            let valor = funcion.cuerpo.execute(newEnv);

            if (funcion.tipo == TipoFuncion.VOID) {
                return {
                    value: null,
                    line: this.line,
                    column: this.column,
                    type: null,
                    tipoDato: null
                }
            }
```

Frontend: La parte del frontend es la parte visual de la aplicación, para esta parte del programa se utilizó React js, con la cual se desarrollaron los componentes siguientes:

- Editor
- Consola

Editor:

Para el editor se utilizó el proporcionado por monaco/react. El cual es el mismo o similar al que usa la herramienta de edición de código VsCode.

Consola:

Esta parte fue realizada con un areaText y css.