



---

# 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, TypeScript y el uso de librerías externas para el editor de código: Monaco editor, el cual es el mismo editor empleado en el motor de VS Code.

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.

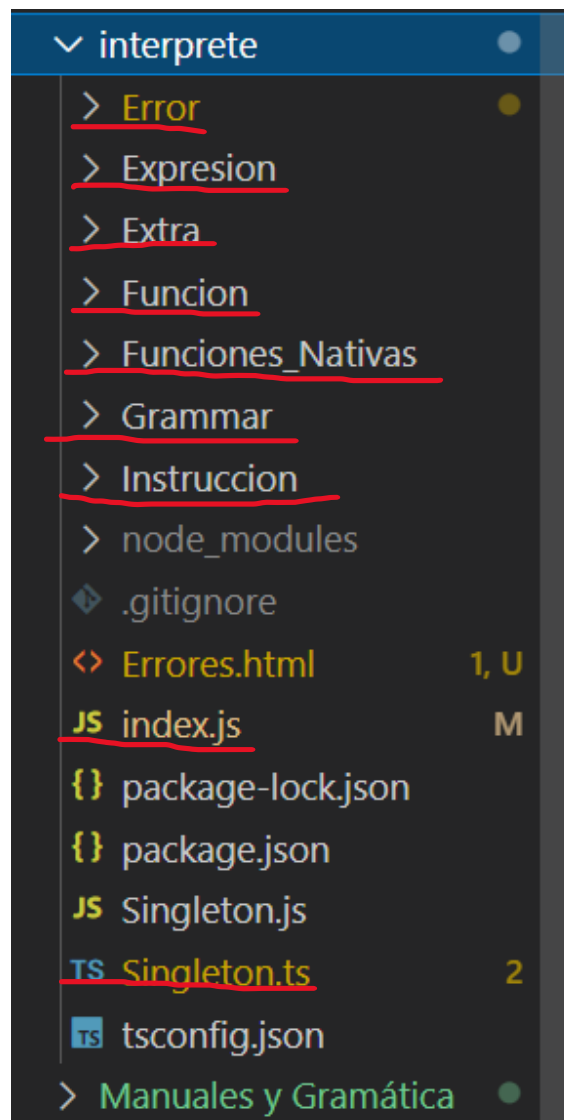
## Descripción de funcionalidades:

**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



```
TS Error.ts  X
interprete > Error > TS Error.ts > Error_
1  export class Error {
2      constructor(public line: number, public column: number, public tipo: string, public mensaje: string) {
3      }
4  }
```

### **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;

```
TS Accesos 1
Interprete > Expresion > TS Accesos > Acceso > execute
1 import { Error_ } from "../Error/Error";
2 import { Ambito } from "../Extra/Ambito";
3 import { Expresion } from "../Expresion";
4 import { Retorno, Type } from "../Retorno";
5
6 export class Acceso extends Expresion {
7
8     constructor(
9         public id: string,           //Identificador de la variable
10        public tipoAcceso: number,    //Variable = 0, vector = 1, matriz = 2
11        public valor1: Expresion,     //Para vector se usa valor1
12        public valor2: Expresion,     //Para matriz se usa valor1 y valor2
13        public line: number,         //Linea
14        public column: number        //Columna
15    ) {
16        super(line, column)
17    }
18
19    public execute(ambito: Ambito): Retorno {
20
21        const value = ambito.getVal(this.id);
22
23        if (value != null) {
24            if (this.tipoAcceso == 0) { //Variables
25                return {
26                    value: value.valor,
27                    type: value.type,
28                    tipoData: value.TipoData
29                };
30            } else if (this.tipoAcceso == 1) { //Vector
31                let v1 = this.valor1.execute(ambito);
32                if (v1.type != Type.ENTERO) throw new Error_(this.line, this.column, "Semántico", "El valor de acceso debe ser de tipo int.");
33                if (v1.value >= value.valor.length) throw new Error_(this.line, this.column, "Semántico", "La posición no existe.");
34                let vector: [] = value.valor;
35                return {
36                    value: vector[v1.value],
37                    type: value.type,
38                    tipoData: value.type
39                };
40            } else if (this.tipoAcceso == 2) { //Matriz
41                let v1 = this.valor1.execute(ambito);
42                let v2 = this.valor2.execute(ambito);
43                if (v1.type != Type.ENTERO || v2.type != Type.ENTERO) throw new Error_(this.line, this.column, "Semántico", "Ambos valores de acceso deben ser de tipo int.");
44                if (v1.value >= value.valor.length || v2.value >= value.valor[0].length) throw new Error_(this.line, this.column, "Semántico", "La posición no existe.");
45                let vector: [] = value.valor;
46                return {
47                    value: vector[v1.value][v2.value],
48                    type: value.type,
49                    tipoData: value.type
50                };
51            }
52        } else {
53            throw new Error_(this.line, this.column, "Semántico", "No se encuentra la variable ${this.id}");
54        }
55    }
56 }
57
58 > public grafo(): string { ...
59 }
60
61
```

## Aritmética:

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

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

```
TS Aritmetica.ts X
interprete > Expresion > TS Aritmetica.ts > TipoAritmetica
1 import { Expresion } from "../Expresion";
2 import { matrizResta, Retorno, TipoDato, Type } from "../Retorno"
3 import { Error_ } from "../Error/Error";
4 import { Ambito } from "../Extra/Ambito";
5 import { nombreTipos } from "../Literal";
6
7 export class Aritmetica extends Expresion {
8
9     constructor(public left: Expresion, public right: Expresion, public tipo: TipoAritmetica, line: number, column: number) {
10         super(line, column);
11     }
12
13 > public execute(ambito: Ambito): Retorno { ...
28     }
29
30 > public grafo(): string { ...
32     }
33
34 > private suma(ambito: Ambito): Retorno { ...
72     }
73
74 > private resta(ambito: Ambito): Retorno { ...
104     }
105
106 > private multiplicacion(ambito: Ambito): Retorno { ...
136     }
137
138 > private division(ambito: Ambito): Retorno { ...
162     }
163
164 > private potencia(ambito: Ambito): Retorno { ...
185     }
186
187 > private modulo(ambito: Ambito): Retorno { ...
202     }
203 }
204
205 > export enum TipoAritmetica { ...
212 }
```

## 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).

```
TS Expresion.ts X
interprete > Expresion > TS Expresion.ts > Expresion > grafo
1 import { Ambito } from "../Extra/Ambito";
2 import { tipos, Type, Retorno, matrizResta, matrizMultiplicacion, matrizDivision, matrizPotencia, matrizModulo } from "../Retorno"
3
4 export abstract class Expresion {
5
6     constructor(public line: number, public column: number) {
7         this.line = line;
8         this.column = column;
9     }
10
11     public abstract execute(ambito: Ambito): Retorno
12
13     public abstract grafo(): string;
14
15     public tipoDominante(tipo1: Type, tipo2: Type): Type {
16         return tipos[tipo1][tipo2];
17     }
18
19 > public tipoDominanteSuma(tipo1: Type, tipo2: Type): Type { ...
21     }
22
23 > public tipoDominanteResta(tipo1: Type, tipo2: Type): Type { ...
25     }
26
27 > public tipoDominanteMultiplicacion(tipo1: Type, tipo2: Type): Type { ...
29     }
30
31 > public tipoDominanteDivision(tipo1: Type, tipo2: Type): Type { ...
33     }
34
35 > public tipoDominantePotencia(tipo1: Type, tipo2: Type): Type { ...
37     }
38
39 > public tipoDominanteModulo(tipo1: Type, tipo2: Type): Type { ...
41     }
42
43 }
```

## 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

```
TS Literal.ts 1 X
Interprete > Expresion > TS Literal.ts > Literal > grafo
1  import { Ambito } from "../Extra/Ambito";
2  import { Expresion } from "../Expresion";
3  import { Retorno, TipoDato, Type } from './Retorno';
4
5  export class Literal extends Expresion {
6
7      constructor(private value: any, private tipo: TipoLiteral, line: number, column: number) {
8          super(line, column);
9      }
10
11     public execute(ambito: Ambito): Retorno {
12
13         if (this.tipo == TipoLiteral.ENTERO) {           //Si es int
14             return {
15                 value: Number(this.value),
16                 type: Type.ENTERO,
17                 tipoDato: TipoDato.ENTERO
18             };
19         } else if (this.tipo == TipoLiteral.CADENA) {    //Si es string
20             return {
21                 value: this.value.toString(),
22                 type: Type.CADENA,
23                 tipoDato: TipoDato.CADENA
24             };
25         } else if (this.tipo == TipoLiteral.BOOLEAN) {   //Si es boolean
26             if (this.value.toString().toLowerCase() == "true") {
27                 return {
28                     value: true,
29                     type: Type.BOOLEAN,
30                     tipoDato: TipoDato.BOOLEAN
31                 };
32             } else {
33                 return {
34                     value: false,
35                     type: Type.BOOLEAN,
36                     tipoDato: TipoDato.BOOLEAN
37                 };
38             }
39         } else if (this.tipo == TipoLiteral.DOUBLE) {   //Si es double
40             return {
41                 value: Number(this.value).toFixed(2),
42                 type: Type.DOUBLE,
43                 tipoDato: TipoDato.DOUBLE
44             };
45         } else if (this.tipo == TipoLiteral.CARACTER) { //Si es char
46             return {
47                 value: this.value,
48                 type: Type.CARACTER,
49                 tipoDato: TipoDato.CARACTER
50             };
51         }
52     }
53
54     public grafo(): string { ...
55     }
56
57
58 }
```



## 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).
- Righth (parte derecha a evaluar).
- tipoL (tipo de expresión lógica).
- Line (línea).
- Column (Columna).

```
TS Logica.ts  X
interprete > Expression > TS Logica.ts > Tipologica
1  import { Expression } from "../Expression";
2  import { Retorno, TipoDato, Type } from "../Retorno";
3  import { Error_ } from "../Error/Error";
4  import { Ambito } from "../Extra/Ambito";
5  export class Logica extends Expression {
6      public tipo = Type.BOOLEAN;
7      constructor(private left: Expression, private right: Expression, private tipoL: TipoLogica, line: number, column: number) {
8          super(line, column);
9      }
10
11     public execute(ambito: Ambito): Retorno {
12         switch (this.tipoL) {
13             case TipoLogica.OR:
14                 return this.or(ambito);
15             case TipoLogica.AND:
16                 return this.and(ambito);
17             case TipoLogica.NOT:
18                 return this.not(ambito);
19         }
20     }
21
22     public grafo(): string {
23         return "";
24     }
25
26 > private or(ambito: Ambito): Retorno { ...
35     }
36
37 > private and(ambito: Ambito): Retorno { ...
46     }
47
48 > private not(ambito: Ambito): Retorno { ...
56     }
57
58 }
```

## 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.

```
TS Relacionalts 9+ X
Interprete > Expresion > TS Relacionalts > Relacional > relaciones
1 import { Expresion } from "../Expresion";
2 import { Retorno, TipoDato, Type } from "../Retorno";
3 import { Error_ } from "../Error/Error";
4 import { Ambito } from "../Extra/Ambito";
5 import { nombreTipos } from "../Literal";
6 export class Relacional extends Expresion {
7     public tipo = Type.BOOLEAN;
8     constructor(private left: Expresion, private right: Expresion, private tipoR: TipoRelacional, line: number, column: number) {
9         super(line, column);
10    }
11    > public execute(ambito: Ambito): Retorno { ...
51    }
52
53    > public grafo(): string { ...
55    }
56
57    > private relaciones(ambito: Ambito): Retorno { ...
73    }
74
75    > private igual_igual(ambito: Ambito): Retorno { ...
87    }
88
89    > private diferente(ambito: Ambito): Retorno { ...
101    }
102
103    > private mayor(ambito: Ambito): Retorno { ...
115    }
116
117    > private mayor_igual(ambito: Ambito): Retorno { ...
129    }
130
131    > private menor(ambito: Ambito): Retorno { ...
143    }
144
145    > private menor_igual(ambito: Ambito): Retorno { ...
157    }
158 }
159
160 export enum TipoRelacional {
161     IGUALIGUAL = 0,
162     DIFERENTE = 2,
163     MAYOR = 3,
164     MAYOR_IGUAL = 4,
165     MENOR = 5,
166     MENOR_IGUAL = 6
167 }
```

## 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.

```
TS Retorno.ts X
interprete > Expresion > TS Retorno.ts > [?] tipos
1  export enum Type {
2      ENTERO = 0, //ENTERO
3      DOBLE = 1, //DOUBLE
4      BOOLEAN = 2, //BOOLEAN
5      CARACTER = 3, //CARACTER
6      CADENA = 4, //CADENA
7  }
8
9  export enum TipoDato {
10     ENTERO = 0,
11     DOBLE = 1,
12     BOOLEAN = 2,
13     CARACTER = 3,
14     CADENA = 4,
15     VECTOR = 5
16 }
17
18 > export type Retorno = { ...
22 }
23
24 //Matriz tipo dominante
25 > export const tipos = [ ...
41 ]
42
43 > export const matrizResta = [ ...
59 ]
60
61 > export const matrizMultiplicacion = [ ...
77 ]
78
79 > export const matrizDivision = [ ...
95 ]
96
97 > export const matrizPotencia = [ ...
113 ]
114
115 > export const matrizModulo = [ ...
131 ]
132
133 > export const matrizNegacionUnaria = [ ...
139 ]
```

## 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.

```
TS Ambito.ts | X
interprete > Extra > TS Ambito.ts > Ambito
1 import { Error_ } from "../Error/Error";
2 import { TipoDato, Type } from "../Expresion/Retorno";
3 import { Funcion } from "../Funcion/Funcion";
4 import { Simbolo } from "../Simbolo"
5
6 export class Ambito {
7     private variables: Map<string, Simbolo> //Guarda las variables, vectores y matrices
8     public funciones: Map<string, Funcion>; //Guarda las funciones
9
10    constructor(public anterior: Ambito | null) {
11        this.variables = new Map();
12        this.funciones = new Map();
13    }
14
15    // Crear variables //
16 > public crearVar(id: string, value: any, type: Type, line: number, column: number, tipoDato) { ...
22    }
23
24    //id, valor, tipo, linea, columna, tipoAsignacion, tipoDato
25 > public setVal(id: string, value: any, type: Type, line: number, column: number, tipoAsignacion: number, tipoDato: TipoDato) { ...
43    }
44
45 > public getVal(id: string): Simbolo { ...
55    }
56
57 // Funciones //
58 > public guardarFuncion(id: string, funcion: Funcion, line: number, column: number) { ...
67    }
68
69 > public getFuncion(id: string): Funcion | undefined { ...
78    }
79
80
81 > public getGlobal(): Ambito { ...
87    }
88
89 }
90
91 export enum TipoAsignacion {
92     DECLARACION,
93     ASIGNACION
94 }
```

## 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

```
TS Simbolo.ts X
interprete > Extra > TS Simbolo.ts > Simbolo
1  import { Retorno, Type } from '../Expresion/Retorno';
2
3  export class Simbolo {
4
5      constructor(
6          public valor: any,           //valor
7          public id: string,           //identificador
8          public type: any,            //tipo
9          public TipoDato: any         //tipo estructura
10         ) {
11
12     }
13
14 }
```

## 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.

```
TS Funcion.ts X
interprete > Funcion > TS Funcion.ts > Funcion > execute
1  import { Ambito } from "../Extra/Ambito";
2  import { Instruccion, TipoFuncion } from "../Instruccion/Instruccion";
3  import { Statement } from "../Instruccion/Statement";
4
5
6  export class Funcion extends Instruccion {
7
8      constructor(
9          public id: string,
10         public cuerpo: Statement,
11         public parametros: Array<[string, number]>,
12         public tipo: TipoFuncion,
13         line: number,
14         column: number
15     ) {
16         super(line, column);
17     }
18
19     public execute(ambito: Ambito): any {
20         ambito.guardarFuncion(this.id, this, this.line, this.column);
21     }
22
23     public grafo(): string {
24         return "";
25     }
26 }
```

## 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 LlamadaFuncion.ts X
interprete > Funcion > TS LlamadaFuncion.ts > LlamadaFuncion > execute
1 import { Instruccion, TipoFuncion } from '../Instruccion/Instruccion';
2 import { Ambito } from '../Extra/Ambito';
3 import { Expresion } from '../Expresion/Expresion';
4 import { Error_ } from '../Error/Error';
5
6 export class LlamadaFuncion extends Instruccion {
7
8     constructor(
9         private id: string,
10         private expresiones: Array<Expresion>,
11         public run: boolean,
12         line: number,
13         column: number
14     ) {
15         super(line, column);
16     }
17
18     public execute(ambito: Ambito) {
19         const funcion = ambito.getFuncion(this.id);
20         if (funcion == undefined) {
21             console.log(undefined);
22             throw new Error_(this.line, this.column, 'Semántico', 'Funcion ${this.id} no encontrada.');
```

```
23
24         if (this.expresiones.length != funcion.parametros.length) throw new Error_(this.line, this.column, 'Semántico', "Cantidad de parametros incorrecta")
25
26         if (this.run) {
27             const newEnv = new Ambito(ambito.getGlobal()); //Obteniendo el ambito global
28
29             for (const i in this.expresiones) {
30                 const value = this.expresiones[i].execute(ambito);
31
32                 if (value.tipoDato != funcion.parametros[i][1]) throw new Error_(this.line, this.column, 'Semántico', "Tipos incorrectos");
33
34                 newEnv.setVal(funcion.parametros[i][0], value.value, value.type, this.line, this.column, 0, value.tipoDato);
35             }
36
37             let valor = funcion.cuerpo.execute(newEnv);
38
39             if (funcion.tipo == TipoFuncion.VOID) {
40                 return {
41                     value: null,
42                     line: this.line,
43                     column: this.column,
44                     type: null,
45                     tipoDato: null
46                 }
47             }
48         }
49     }
50 }
```

**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.

