Adrian Samuel Molina Cabrera

201903850

Manual Técnico

En el index.ts se ejecuta el parser que lee el codigo que viene desde la interfaz gráfica.

```
app.post('/analizar', (req, res) => {
  const { entrada, consola } = req.body;
  if (!entrada) {
    return res.redirect('/');
  }
  try {
    const tree = parser.parse(entrada);
    const tabla = new Table(null);

    tree.instrucciones.map((m: any) => {
      try {
        const res = m.execute(tabla, tree);
      } catch (error) {
        const error2 = new Excepcion('Sintactico',
          `Irrecuperable`, 0, 0);
        tree.consola.push(error2.toString());
      }
      if (res instanceof Break || res instanceof Retorno) {
        const error = new Excepcion('Semantico',
          `Sentencia break fuera de un ciclo`,
          res.line, res.column);
        tree.excepciones.push(error);
        tree.consola.push(error.toString());
      } else if (res instanceof Continue) {
        const error = new Excepcion('Semantico',
          `Sentencia continue fuera de un ciclo`,
          res.line, res.column);
        tree.excepciones.push(error);
        tree.consola.push(error.toString());
      }
    });
```

El AST y la tabla de símbolos se crea con el archivo graficar.ts, este tiene como parámetro la tabla generada luego de leer el código ingresado al igual que el nodo principal del árbol generado.

```
> export function graphAST(raiz: NodoAST): void {···
  }

  var c: number;
  var grafo: String;

> function getDot(raiz: NodoAST): String {···
  }

> function recorrerAST(padre: String, nPadre: NodoAST): void {···
  }

> export function graphTabla(tabla: Array<Simbolo>): void {···
  }

> function escribirHtml(tabla: Array<Simbolo>, documento: String): String {···
  }
```

La clase Símbolo sirve para guardar todas las variables del programa, su tipo, su id, su valor, también se guardan vectores, listas, métodos y funciones.

```
import {Tipo, tipos} from "../other/tipo";

export class Simbolo {
    tipo: Tipo;
    id: String;
    valor: Object;
    line : Number;
    column : Number;
    tipo2 : Tipo;


    constructor(tipo: Tipo, id: String, valor: Object, tipo2: Tipo,line:Number, column:Number) {
        this.tipo = tipo;
        this.id = id;
        this.valor = valor;
        this.line = line;
        this.column = column;
        this.tipo2 = tipo2;
    }
}
```

Los símbolos se guardan en la tabla de símbolos mediante un diccionario, para que sea mas fácil y rápido obtener las variables.

```
import {Simbolo} from "./Simbolo";

export class Table{
    Anterior: Table;
    Variables: Map<String, Simbolo>;

    constructor(Anterior: Table){
        this.Anterior = Anterior;
        this.Variables = new Map<String, Simbolo>();
    }

    setVariable(simbol: Simbolo){ ···
    }

    getVariable(id: String): Simbolo{ ···
    }
}
```

La clase del árbol almacena todo lo que se utilizará en el código, las variables, los errores, las instrucciones y lo que se mostrará en consola.

```
import {Nodo} from "../Abstract/Nodo";
import {Excepcion} from "../other/Excepcion";
import { Simbolo } from "./Simbolo";

export class Tree {
    instrucciones: Array<Nodo>
    excepciones: Array<Excepcion>
    consola: Array<String>
    Variables: Array<Simbolo>;

    constructor(instrucciones: Array<Nodo>) {
        this.instrucciones = instrucciones;
        this.excepciones = new Array<Excepcion>();
        this.consola = new Array<String>();
        this.Variables = new Array<Simbolo>();
    }
}
```

Todo el arbol es a base de nodos, la clase es abstracta para que todos tuvieran el método execute, el cual sirve para que se ejecute uno tras otro.

```
import {Table} from "../Simbols/Table";
import {Tree} from "../Simbols/Tree";
import {Tipo} from "../other/tipo";
import {NodoAST} from "../Abstract/NodoAST";
export abstract class Nodo{
    line: Number;
    column: Number;
    tipo: Tipo;

    abstract getNodo():NodoAST;

    abstract execute(table: Table, tree: Tree): any;

    constructor(tipo: Tipo, line: Number, column: Number) {
        this.tipo = tipo;
        this.line = line;
        this.column = column;
    }
}
```

También se creó un método NodoAST, para que hiciera la misma tarea que execute, pero enfocado a la creación del grafo ast.

```
import LinkedList from 'ts-linked-list';

export class NodoAST {
    hijos: LinkedList<NodoAST>;
    valor: String;

    constructor(valor: String) {
        this.hijos = new LinkedList();
        this.valor = valor;
    }


    public setHijos(hijos: LinkedList<NodoAST>): void {···
    }

    public agregarHijo(hijo: any): void {···
    }

    public agregarHijos(hijos: LinkedList<NodoAST>): void {
        hijos.forEach(hijo => this.hijos.append(hijo));
    }

    public agregarPrimerHijo(hijo: any): void {···
    }

    public getValor(): String {
        return this.valor;
    }

    public setValor(cad: String): void {
        this.valor = cad;
    }
```

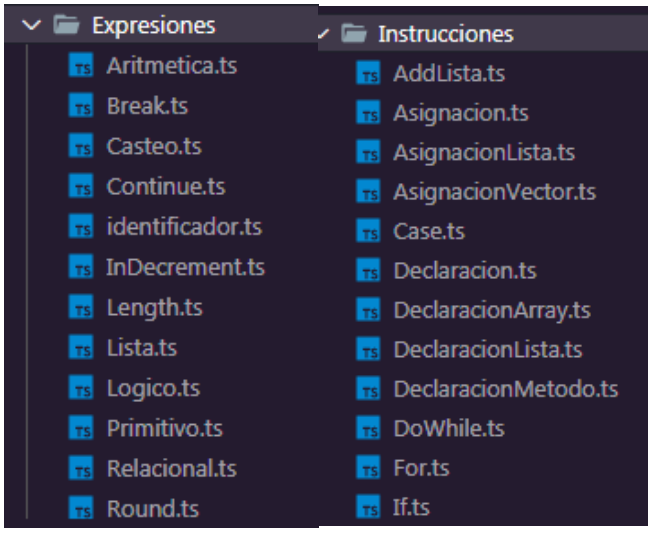
El archivo gramar es la base de todo el proyecto, es el analizador léxico y sintáctico.

```
ini
    : instrucciones EOF             {$$ = new Tree($1); return $$;}
;

instrucciones
    :instrucciones instruccion      {$$ = $1; $1.push($2);}
    |instruccion                    {$$=[$1];}
;

instruccion
    :declaracionVar                 {$$ = $1;}
    |metodos                        {$$ = $1;}
    |llamada                        {$$ = $1;}
    |sentencia_if                   {$$ = $1;}
    |sentencia_switch               {$$ = $1;}
    |sentencia_while                {$$ = $1;}
    |sentencia_for                  {$$ = $1;}
    |sentencia_dowhile              {$$ = $1;}
    |sentencia_print                {$$ = $1;}
    |ID INC PTCOMA                  {$$ = new InDecrement($1, "++", @1.first_line, @1.first_column);}
    |ID DEC PTCOMA                  {$$ = new InDecrement($1, "--", @1.first_line, @1.first_column);}
    |CONTINUE PTCOMA                {$$ = new Continue(@1.first_line, @1.first_column);}
    |BREAK PTCOMA                   {$$ = new Break(@1.first_line, @1.first_column);}
    |sentencia_return               {$$ = $1;}
    |EXEC llamada                   {$$ = $2;}
;
```

Todas las instrucciones y expresiones son nodos.



Tenemos un enum de tipos para poder llevar un mejor orden con los nodos.

```
export enum tipos {
    ENTERO,
    DECIMAL,
    NUMERO,
    CARACTER,
    STRING,
    BOOLEANO,
    LISTA,
    ARRAY,
    VOID,
    METODO,
    FUNCION,
    VARIABLE
}

export function esEntero(numero: number) {
    if (numero % 1 == 0) {
        return tipos.ENTERO;
    } else {
        return tipos.DECIMAL;
    }
}
export class Tipo {
    tipo: tipos;


    constructor(tipo: tipos) {
        this.tipo = tipo;
    }
```

La clase excepcion sirve para llevar el conteo de errores generados en la compilacion del programa.

```
export class Excepcion{
    tipo: String;
    descripcion: String;
    line: Number;
    column: Number;

    constructor(tipo: String, descripcion: String, line: Number, column: Number) {
        this.tipo = tipo;
        this.descripcion = descripcion;
        this.line = line;
        this.column = column;
    }

    toString(){
        return `Error ${this.tipo} en la linea ${this.line} y columna ${this.column}, ${this.descripcion}`;
    }
}
```