MANUAL

-

TECNICO

FUNCION PARA ANALIZAR EL CODIGO EN EL EDITOR

```
const handleExecute = () => {

  let cadena_codigo = editableText;
  let analizador = new Analizador(cadena_codigo, "editor");

  let ast: AST = analizador.Analizar();


  if(ast != undefined) {
    setNonEditableText(ast.getSalida());
  }else{
    setNonEditableText("Error al analizar");
  }

};
```

FUNCIÓN PARA GENERAR EL ARBOL DE LAS INSTRUCCIONES Y EXPRESIONES,

```
private graphiz(){
    this.nodes="";
    this.connections="";
    this.nodes += `S_Arbol[label="Arbol" fillcolor="${this.fillcolor}" style=filled];\n`
    for (let x = 0; x < this.sentencias.length; x++) {
        this.initGraphiz(this.sentencias[x])
    }
    let body = `digraph G { \nbgcolor=transparent \n${this.nodes + this.connections}}`;
    //console.log(body)
    variables.treeContent.setContent(body)
}
private initGraphiz(sentencia: Nodo){
    this.recursiveGraphiz(sentencia.constructor.name,sentencia,"S_Arbol");
}
private recursiveGraphiz(nombre: string,sentencia: any,nodoPadre: string){

    let idNode=this.idNode;

    if (nombre==="DeclararVariable"){
        if(sentencia.exp){
            this.nodes += `S_DeclararVariable${idNode}[label="DeclararVariable" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `${nodoPadre} -> S_DeclararVariable${idNode} [color="${this.color}"];\n`;

            const clave: string = Object.keys(TipoPrimitivo).find(key => TipoPrimitivo[key] === sentencia.exp.tipo.tipo) || '';

            this.nodes += `S_DeclararVariable_tipo${idNode}[label="${clave}" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_tipo${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_DeclararVariable_id${idNode}[label="${sentencia.id}" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_id${idNode} [color="${this.color}"];\n`;
```

```
        this.nodes += `S_DeclararVariable_igual${idNode}[label="=" fillcolor="${this.fillcolor}" style=filled];\n`
        this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_igual${idNode} [color="${this.color}"];\n`;

        this.recursiveGraphiz(sentencia.exp.constructor.name,sentencia.exp,"S_DeclararVariable"+idNode);

        this.nodes += `S_DeclararVariable_ptcoma${idNode}[label=";" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_ptcoma${idNode} [color="${this.color}"];\n`;
        this.idNode+=1;


    }else{
        const clave: string = Object.keys(TipoPrimitivo).find(key => TipoPrimitivo[key] === sentencia.tipo.tipo) || '';
        this.nodes += `S_DeclararVariable${idNode}[label="DeclararVariable" fillcolor="${this.fillcolor}" style=filled];\n`
        this.nodes += `S_DeclararVariable_tipo${idNode}[label="${clave}" fillcolor="${this.fillcolor}" style=filled];\n`
        this.nodes += `S_DeclararVariable_id${idNode}[label="${sentencia.id}" fillcolor="${this.fillcolor}" style=filled];\n`
        this.nodes += `S_DeclararVariable_ptcoma${idNode}[label=";" fillcolor="${this.fillcolor}" style=filled];\n`

        this.connections += `${nodoPadre} -> S_DeclararVariable${idNode} [color="${this.color}"];\n`;
        this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_tipo${idNode} [color="${this.color}"];\n`;
        this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_id${idNode} [color="${this.color}"];\n`;
        this.connections += `S_DeclararVariable${idNode} -> S_DeclararVariable_ptcoma${idNode} [color="${this.color}"];\n`;
        this.idNode+=1;
    }
    this.idNode+=1;
}else if(nombre==="LlamadaFuncion"){
    this.nodes += `S_LlamadaFuncion${idNode}[label="LlamadaFuncion" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_LlamadaFuncion_nombre${idNode}[label="${sentencia.nombre}" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_LlamadaFuncion_pi${idNode}[label="(" fillcolor="${this.fillcolor}" style=filled];\n`

    this.connections += `${nodoPadre} -> S_LlamadaFuncion${idNode} [color="${this.color}"];\n`;
    this.connections += `S_LlamadaFuncion${idNode} -> S_LlamadaFuncion_nombre${idNode} [color="${this.color}"];\n`;
    this.connections += `S_LlamadaFuncion${idNode} -> S_LlamadaFuncion_pi${idNode} [color="${this.color}"];\n`;

    this.auxiliarRecursiveGraphiz(sentencia.lista_exp[0],"S_LlamadaFuncion"+idNode)
```

```
    this.nodes += `S_LlamadaFuncion_pd${idNode}[label=")" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_LlamadaFuncion_ptcoma${idNode}[label=";" fillcolor="${this.fillcolor}" style=filled];\n`

    this.connections += `S_LlamadaFuncion${idNode} -> S_LlamadaFuncion_pd${idNode} [color="${this.color}"];\n`;
    this.connections += `S_LlamadaFuncion${idNode} -> S_LlamadaFuncion_ptcoma${idNode} [color="${this.color}"];\n`;
    this.idNode+=1;
}else if(nombre==="OperacionAritmetica" || nombre==="OperacionRelacional"){
    this.idNode+=1;
    this.recursiveGraphiz(sentencia.exp1.constructor.name,sentencia.exp1,nodoPadre);

    this.nodes += `S_Simbolo${idNode}[label="${sentencia.signo}" fillcolor="${this.fillcolor}" style=filled];\n`
    this.connections += `${nodoPadre} -> S_Simbolo${idNode} [color="${this.color}"];\n`;
    this.idNode+=1;
    this.recursiveGraphiz(sentencia.exp2.constructor.name,sentencia.exp2,nodoPadre);


    this.idNode+=1;

}else if(nombre==="AccesoVariable"){
    if(sentencia.key){
        this.nodes += `S_AccesoVariable${idNode}[label="${sentencia.nombreVar}" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_ci1${idNode}[label="[" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_ci1${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_posv${idNode}[label="${sentencia.posicionVector}" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_posv${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_cd1${idNode}[label="]" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_cd1${idNode} [color="${this.color}"];\n`;
    }else if(sentencia.key2){
        this.nodes += `S_AccesoVariable${idNode}[label="${sentencia.nombreVar}" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_ci1${idNode}[label="[" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_ci1${idNode} [color="${this.color}"];\n`;
```

```
        this.nodes += `S_AccesoVariable_id${idNode}[label="${sentencia.posicionVector}" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_id${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_cd1${idNode}[label="]" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_cd1${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_AccesoVariable_cd2${idNode}[label="]" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable_cd2${idNode} [color="${this.color}"];\n`;
    }else{
        this.nodes += `S_AccesoVariable${idNode}[label="${sentencia.nombreVar}" fillcolor="${this.fillcolor}" style=filled];\n`;
        this.connections += `${nodoPadre} -> S_AccesoVariable${idNode} [color="${this.color}"];\n`;
    }
    this.idNode+=1;
}else if(nombre==="Valor"){
    this.nodes += `S_Valor${idNode}[label="${sentencia.valor}" fillcolor="${this.fillcolor}" style=filled];\n`
    this.connections += `${nodoPadre} -> S_Valor${idNode} [color="${this.color}"];\n`;
    this.idNode+=1;
}else if(nombre==="DeclararFuncion"){
    const clave: string = Object.keys(TipoPrimitivo).find(key => TipoPrimitivo[key] === sentencia.tipo.tipo) || '';
    this.nodes += `S_DeclararFuncion${idNode}[label="DeclararFuncion" fillcolor="${this.fillcolor}" style=filled];\n`
    this.connections += `${nodoPadre} -> S_DeclararFuncion${idNode} [color="${this.color}"];\n`;

    this.nodes += `S_DeclararFuncion_tipoPrimitivo${idNode}[label="${clave}" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_DeclararFuncion_nombre${idNode}[label="${sentencia.nombre}" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_DeclararFuncion_pi${idNode}[label="(" fillcolor="${this.fillcolor}" style=filled];\n`

    this.connections += `S_DeclararFuncion${idNode} -> S_DeclararFuncion_tipoPrimitivo${idNode} [color="${this.color}"];\n`;
    this.connections += `S_DeclararFuncion${idNode} -> S_DeclararFuncion_nombre${idNode} [color="${this.color}"];\n`;
    this.connections += `S_DeclararFuncion${idNode} -> S_DeclararFuncion_pi${idNode} [color="${this.color}"];\n`;
    this.forRecursiveGraphiz(sentencia.parametros,"S_DeclararFuncion"+idNode,true)
    //parametros
    this.nodes += `S_DeclararFuncion_pd${idNode}[label=")" fillcolor="${this.fillcolor}" style=filled];\n`
    this.nodes += `S_DeclararFuncion_ci${idNode}[label="{" fillcolor="${this.fillcolor}" style=filled];\n`

    this.connections += `S_DeclararFuncion${idNode} -> S_DeclararFuncion_pd${idNode} [color="${this.color}"];\n`;
    this.connections += `S_DeclararFuncion${idNode} -> S_DeclararFuncion_ci${idNode} [color="${this.color}"];\n`;
```

```
        this.idNode+=1;
    }else if(nombre==="Asignacion"){
        this.nodes += `S_Asignacion${idNode}[label="Asignacion" fillcolor="${this.fillcolor}" style=filled];\n`
        this.connections += `${nodoPadre} -> S_Asignacion${idNode} [color="${this.color}"];\n`;

        this.nodes += `S_Asignacion_valor${idNode}[label="${sentencia.id}" fillcolor="${this.fillcolor}" style=filled];\n`
        this.connections += `S_Asignacion${idNode} -> S_Asignacion_valor${idNode} [color="${this.color}"];\n`;
        if(sentencia.exp==="++"){
            this.nodes += `S_Asignacion_mas${idNode}[label="++" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_mas${idNode} [color="${this.color}"];\n`;
        }else if(sentencia.exp==="--"){
            this.nodes += `S_Asignacion_menos${idNode}[label="--" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_menos${idNode} [color="${this.color}"];\n`;
        }else if(sentencia.key){
            this.nodes += `S_Asignacion_ci1${idNode}[label="[" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_ci1${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_Asignacion_num1${idNode}[label="${sentencia.posicion}" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_num1${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_Asignacion_cd1${idNode}[label="]" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_cd1${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_Asignacion_igual${idNode}[label="=" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_igual${idNode} [color="${this.color}"];\n`;

            this.auxiliarRecursiveGraphiz(sentencia.exp,"S_Asignacion"+idNode)

        }else if(sentencia.key2){
            this.nodes += `S_Asignacion_punto${idNode}[label="." fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_punto${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_Asignacion_add${idNode}[label="add" fillcolor="${this.fillcolor}" style=filled];\n`
            this.connections += `S_Asignacion${idNode} -> S_Asignacion_add${idNode} [color="${this.color}"];\n`;

            this.nodes += `S_Asignacion_pi${idNode}[label="(" fillcolor="${this.fillcolor}" style=filled];\n`
```

COMPONENTE PARA MOSTRAR LOS ERRORES

```
import { Err } from "../../Data/Errores";
import "./styles.css"
const ErrorTable= () => {
    const Er = Err.obtenerListaDatos();
  return (
    <div className="container">
        <table className="error-table">
        <thead>
          <tr>
            <th>No.</th>
            <th>Tipo</th>
            <th>Descripción</th>
            <th>Fila</th>
            <th>Columna</th>

          </tr>
        </thead>
        <tbody>
          {

          Er.map((error, index) => (
            <tr key={index}>
              <td>{index + 1}</td>
              <td>{error.tipo}</td>
              <td>{error.dato}</td>
              <td>{error.linea}</td>
              <td>{error.columna}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>

  );
};

export default ErrorTable;
```

COMPONENTE PARA MOSTRAR LA TABLA DE SIMBOLOS.

```
import React from "react";
import { Ambito } from "../../Entorno/Ambito"

const TablaSimbolos = () => {
  const variables = Ambito.returnvariables();
  const metodos = Ambito.returnmetodos();
  let tipo43;
  const tablaSimbolos = [];
  let contador = 0;

  // Agregar variables a la tabla de símbolos
  variables.forEach((variable) => {
    tablaSimbolos.push({
      id: ++contador,
      columna: variable.columna,
      fila: variable.linea,
      tipo1: "variable",
      tipo: obtenerTipo(variable.tipo.tipo),
      descripcion: variable.id,
      ambito: variable.ambito
    });
  });

  // Agregar métodos a la tabla de símbolos
  metodos.forEach((metodo) => {
    tablaSimbolos.push({
      id: ++contador,
      columna: metodo.columna,
      fila: metodo.linea,

      tipo: obtenerTipo(metodo.tipo.tipo),
      tipo1: tipo43,
      descripcion: metodo.nombre,
      ambito: "--"
    });
  });
```

```
function obtenerTipo(tipo) {
  switch (tipo) {
    case 0:
      tipo43 = "funcion"
      return "integer";
    case 1:
      tipo43 = "funcion"
      return "double";
    case 2:
      tipo43 = "funcion"
      return "char";
    case 3:
      tipo43 = "funcion"
      return "string";
    case 4:
      tipo43 = "funcion"
      return "null";
    case 5:
      tipo43 = "funcion"
      return "boolean";
    case 6:
      tipo43 = "metodo"
      return "void";
    case 7:
      tipo43 = "funcion"
      return "vector";
    case 8:
      tipo43 = "funcion"
      return "lista";
    case 9:
      tipo43 = "funcion"
      return "variable";
    default:
      return "";
  }
}
```

```
  return (
    <>
      <h1>TABLA DE SIMBOLOS</h1>
      <table>
        <thead>
          <tr>
            <th>#</th>
            <th>Identificador</th>
            <th>Tipo</th>
            <th>Tipo</th>
            <th>Entorno</th>
            <th>Linea</th>
            <th>Columna</th>
          </tr>
        </thead>
        <tbody>
          {tablaSimbolos.map((simbolo, index) => (
            <tr key={index}>
              <td>{simbolo.id}</td>
              <td>{simbolo.descripcion}</td>
              <td>{simbolo.tipo}</td>
                <td>{simbolo.tipo1}</td>
              <td>{simbolo.ambito}</td>
                <td>{simbolo.fila}</td>
              <td>{simbolo.columna}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </>
  );
};

export default TablaSimbolos;
```

COMPONENTE PARA VISUALIZAR EL ARBOL.

```
import React, { useEffect, useRef, useCallback } from "react";
import { Graphviz } from "graphviz-react";
import { graphviz } from "d3-graphviz";

export default ({ dot, width, height }) => {
  // gen css from props
  const style = {
    width: width || "100%",
    height: height || "100%"
  };
  const graphvizRoot = useRef(null);

  // update style in Graphviz div
  useEffect(() => {
    if (graphvizRoot.current) {
      const { id } = graphvizRoot.current;
      // use DOM id update style
      const el = document.getElementById(id);
      for (let [k, v] of Object.entries(style)) {
        el.style[k] = v;
      }
      graphviz(`#${id}`);
    }
  }, [graphvizRoot]);
  const reset = useCallback(() => {
    if (graphvizRoot.current) {
      const { id } = graphvizRoot.current;
      graphviz(`#${id}`).resetZoom();
    }
  }, [graphvizRoot]);
  return (
    <div
      style={{
        ...style,
        position: "relative"
      }}
    >
```

```
  return (
    <div
      style={{
        ...style,
        position: "relative"
      }}
    >
      {dot !== ""
        ? [
            <Graphviz
              dot={dot}
              options={{
                useWorker: false,
                ...style,
                zoom: true
                //...props
              }}
              ref={graphvizRoot}
            />,
            // <button
            //   onClick={reset}
            //   style={{
            //     position: "absolute",
            //     right: "5%",
            //     top: "5%"
            //   }}
            // >
            //   Reset
            // </button>
          ]
        : null}
    </div>
  );
};
```

COMPONENTES.

- SIDEBAR.
- LAYOUT.
- SIMBOLOS.
- TABLA ERRORES.
- ARBOL.
- TREE.

PAGES.

- EDITORPAGE
- HOMEPAGE
- REPORTSPAGE
- SIMBOLOS
- TREEPAGE

COMPONENTES DEL ANALIZADOR.

- ENTORNO

- EXPRESIONES
- INSTRUCCIONES.

ENTORNO

- AMBITO
- AST
- EXPRESION
- INSTRUCCION
- NODO
- RAIZ

EXPRESIONES

- ACCESOVARIABLE
- LLAMDAFUNCION
- OPERACIONARITMETICA
- OPERACIONRELACIONAL
- VALOR

INSTRUCCIONES

- ASIGNACION
- DECLARARFUNCION
- DECLARARLISTA
- DECLARAVARIABLE
- DECLARARVECTOR
- DOWHILE
- FOR
- IF
- INCREMENTO DECREMENTO
- LENGTH
- PARAMETRO
- PRINT
- ROUND
- SWITCH
- TOLOWER
- TOSTRING
- TOUPPER
- TRUNCATE
- TYPEOF
- WHILE