Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ingeniería en ciencias de sistemas.

Manual técnico

Nombre del estudiante

José Luis Espinoza Jolón - 202202182

Guatemala, septiembre de 17 de 2024

Contenido

Descripción del programa 3

Lenguaje utilizado 3

Estructura del proyecto 4

- Gramática 5
- Entorno 6
- Interprete 7
- structC 8
- Instancia 8
- Remota 9

Diseño del programa 10

- index html 10
- index js 10
- styles css 11

Descripción del programa

desarrolle un intérprete para el lenguaje de programación OakLand, el cual toma inspiración de la sintaxis de Java, aunque esta no es su característica principal. OakLand sobresale por su capacidad para soportar diversos paradigmas de programación, como la orientación a objetos, la programación funcional y la programación procedimental.

Adicionalmente, se deberá crear una plataforma simple pero robusta para permitir la creación, apertura, edición e interpretación de código en OakLand. Este entorno de desarrollo (IDE) se implementará utilizando JavaScript puro y será alojado en Github Pages.

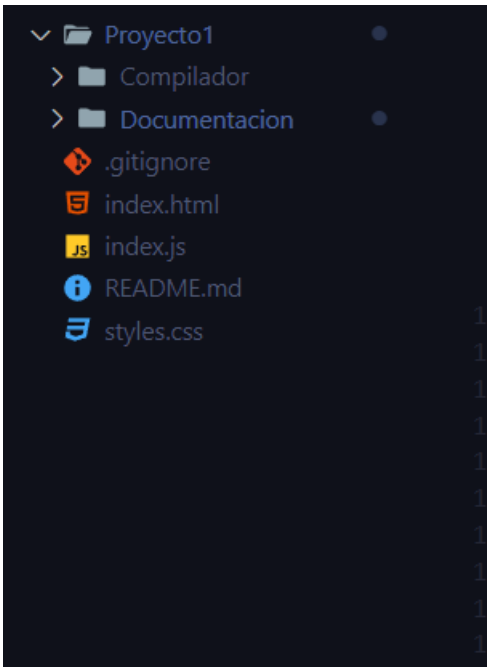
Los analizadores léxico y sintáctico serán desarrollados utilizando la herramienta PeggyJS, para lo cual será necesario escribir una gramática que defina la sintaxis del lenguaje.

Lenguaje utilizado

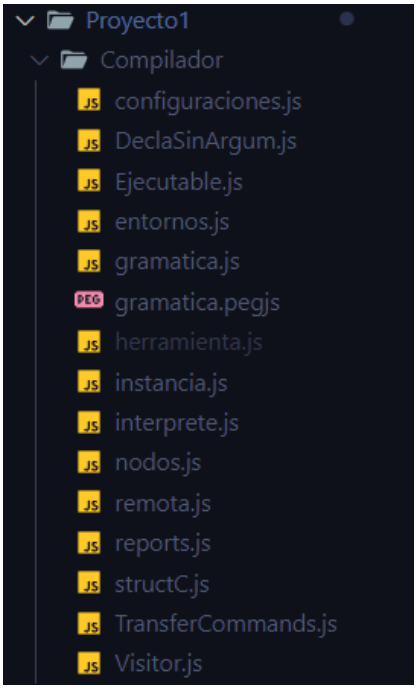
Se utilizo **JavaScript** y para el analizador **PeggyJs**

Estructura del proyecto

La manera que se organizó el proyecto fue de la siguiente manera:

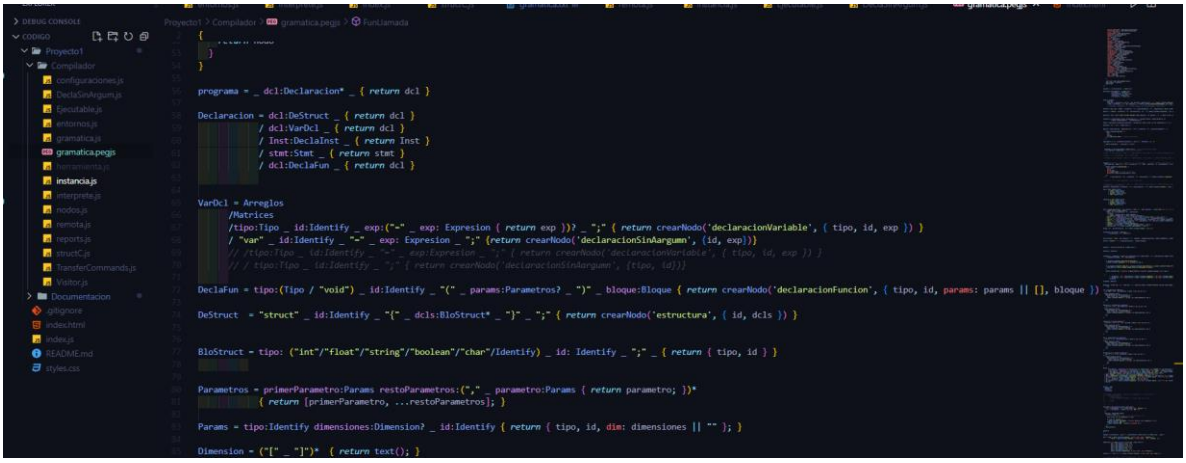


Desplegado todas las carpetas



- Gramática

El archivo que contiene toda la gramática del proyecto se llama **gramática.pegjs**



```
{
}

programa = _ dcl:Declaracion* _ { return dcl }

Declaracion = dcl:DeStruct _ { return dcl }
            / dcl:VarDcl _ { return dcl }
            / Inst:DeclaInst _ { return Inst }
            / stmt:Stmt _ { return stmt }
            / dcl:DeclaFun _ { return dcl }


VarDcl = Arreglos
      /Matrices
      /tipo:Tipo _ id:Identify _ exp:("=" _ exp: Expresion { return exp })? _ ";" { return crearNodo('declaracionVariable', { tipo, id, exp }) }
      / "var" _ id:Identify _ "=" _ exp: Expresion _ ";" {return crearNodo('declaracionSinAargumn', {id, exp})}
      // /tipo:Tipo _ id:Identify _ "=" _ exp:Expresion _ ";" { return crearNodo('declaracionVariable', { tipo, id, exp }) }
      // / tipo:Tipo _ id:Identify _ ";" { return crearNodo('declaracionSinAargumn', {tipo, id})}

DeclaFun = tipo:(Tipo / "void") _ id:Identify _ "(" _ params:Parametros? _ ")" _ bloque:Bloque { return crearNodo('declaracionFuncion', { tipo, id, params: params || [], bloque }) }

DeStruct  = "struct" _ id:Identify _ "{" _ dcls:BloStruct* _ "}" _ ";" { return crearNodo('estructura', { id, dcls }) }


BloStruct = tipo: ("int"/"float"/"string"/"boolean"/"char"/Identify) _ id: Identify _ ";" _ { return { tipo, id } }


Parametros = primerParametro:Params restoParametros:("," _ parametro:Params { return parametro; })*
           { return [primerParametro, ...restoParametros]; }

Params = tipo:Identify dimensiones:Dimension? _ id:Identify { return { tipo, id, dim: dimensiones || "" }; }

Dimension = ("[" _ "]")*  { return text(); }
```

La gramática se realizo por la derecha, que contiene nivel de precedencia por cada producción, La gramática abarca una amplia variedad de construcciones, desde operaciones aritméticas y lógicas hasta declaraciones y estructuras de control como if, while, for, switch, y más.

Permite la declaración de funciones y estructuras (struct), lo que la hace apta para lenguajes con tipado fuerte

Incluye manejo de arreglos y matrices multidimensionales, proporcionando flexibilidad en la gestión de colecciones de datos.

- Entorno

es una clase que representa un espacio de trabajo o contexto donde se definen y manejan variables durante la ejecución de un programa o interpretación de un código. Es esencial para gestionar el alcance de las variables (alcance léxico) y las relaciones entre diferentes niveles de entornos (por ejemplo, dentro de funciones o bloques anidados).

```
export class Entorno {
    /**
     * @param {Entorno} padre
     */
    constructor(padre = undefined) {
        this.valores = {};
        this.padre = padre;
    }

    /**
     * @param {string} nombre
     * @param {any} valor
     */
    setVariable(tipo, nombre, valor,linea ,columna) {

        if(buscarTablarReservada(nombre)){
            //throw new Error(`Error: No se puede declarar la variable ${nombre} porque es una palabra reservada`);
            let errores = new erroresReporte(linea,columna,`Error: No se puede declarar la variable ${nombre} porque es una palabra reservada`);
            erroresCompilacion.push(errores);
            return
        }


        if (this.valores[nombre] != undefined) {
            //throw new Error(`Error: La variable ${nombre} ya está definida`);
            let errores = new erroresReporte(linea,columna,`Error: La variable ${nombre} ya está definida`);
            erroresCompilacion.push(errores);
            return;
```

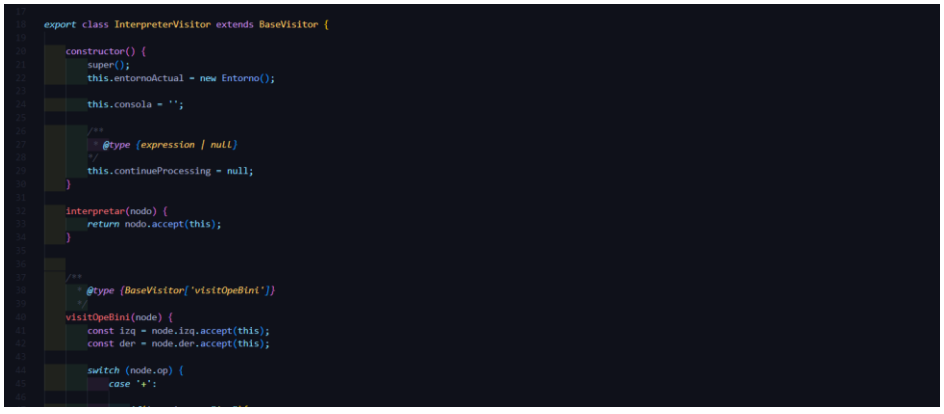
- Interprete

InterpreterVisitor extiende de BaseVisitor y se encarga de interpretar nodos en un árbol de sintaxis abstracta (AST).

- this.entornoActual: Guarda el entorno de ejecución actual (un objeto que maneja las variables y sus valores).
- this.consola: Una cadena que actúa como el buffer de salida para los mensajes de la consola.
- this.continueProcessing: Un marcador que se utiliza para controlar el procesamiento de bucles y otras estructuras iterativas.

Tal ves se realizo toda la funcionalidad del lenguaje, como el for, while ,etc.

```
export class InterpreterVisitor extends BaseVisitor {
    /**
     * @type {BaseVisitor['visitBoolena']}
     */
    visitBoolena(node){
        return {valor:node.valor,tipo:node.tipo}

    }

    /**
     * @type {BaseVisitor['visitCadenaString']}
     */

    visitCadenaString(node){
        return {valor:node.valor ,tipo:node.tipo}
    }


    /**
     * @type {BaseVisitor['visitCaracter']}
     */


    visitCaracter(node){
        return {valor:node.valor, tipo:node.tipo}

    }

    /**
     * @type {BaseVisitor['visitEmbebidas']}
     */
```



```
export class InterpreterVisitor extends BaseVisitor {

    constructor() {
        super();
        this.entornoActual = new Entorno();

        this.consola = '';

        /**
         * @type {expression | null}
         */
        this.continueProcessing = null;
    }

    interpretar(nodo) {
        return nodo.accept(this);
    }


    /**
     * @type {BaseVisitor['visitOpeBini']}
     */
    visitOpeBini(node) {
        const izq = node.izq.accept(this);
        const der = node.der.accept(this);

        switch (node.op) {
            case '+':
```

- structC

Esta clase se realizo el manejo del struct , junto con la clase interprete

```
import { Ejecutable } from "./Ejecutable.js";
import { Instancia } from "./instancia.js";

export class StructC extends Ejecutable {

    constructor(nombre,propiedades){
        super();
        /**
         * @type {string}
         */
        this.nombre = nombre;

        /**
         * @type {Object.<string, Expresion>}
         */
        this.propiedades = propiedades;


    }

    aridad() {
        return Object.keys(this.propiedades).length;
    }

    /**
     * @type {Invocable['invocar']}
     */
    invocar(interprete, args) {
        const nuevaInstancia = new Instancia(this);

        Object.entries(this.propiedades).forEach(([nombre, valor]) => {
```

- Instancia

la clase Instancia permite gestionar las propiedades de una estructura definida por StructC, con métodos para establecer y obtener propiedades, manejando errores cuando se intentan acceder o modificar propiedades que no están definidas.

```
    set(nombre, valor,nodo) {
        //console.log("Instancia.set: ", nombre, valor);
        if (!(nombre in this.structC.propiedades)) {
            //throw new Error(`La propiedad ${nombre} no está definida en la estructura ${this.structC.nombre}`);
            let error = new erroresReporte(nodo.location.start.line, nodo.location.start.column,`La propiedad ${nombre} no está definida en la estructura ${this.structC.nombre}`);
            erroresCompilacion.push(error);
            return;
        }
        this.structC.propiedades[nombre] = valor;
    }

    get(nombre,nodo) {
        if(this.structC.propiedades.hasOwnProperty(nombre)){
            return this.structC.propiedades[nombre];
        }else{
            //throw new Error(`Propiedad no encontrada: ${nombre}`);
            let error = new erroresReporte(nodo.location.start.line,nodo.location.start.column,`Propiedad no encontrada: ${nombre}`);
            erroresCompilacion.push(error);
        }
    }
}
```

- Remota

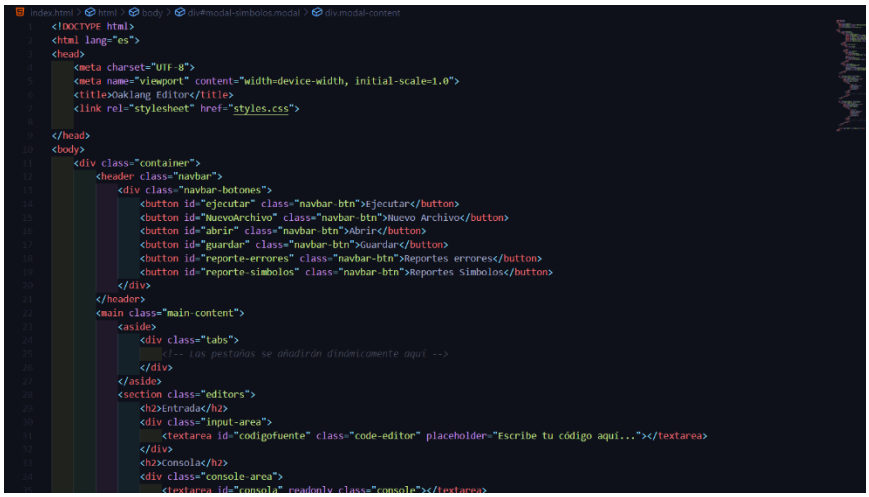
la clase FuncionRemota maneja la invocación de una función en un entorno específico, configurando un nuevo entorno para los parámetros de la función, ejecutando el código de la función, y manejando errores relacionados con el tipo de retorno y las excepciones de control de flujo.

```
import { BreakException } from './TransferCommands.js';
import { ReturnException } from './TransferCommands.js';
import { erroresReporte } from "./reports.js";
import { erroresCompilacion } from "../../index.js";

export class FuncionRemota extends Ejecutable {

    constructor(node,clousure) {
        super();

        /**
         * @type {DeclaracionFuncion}
         */
        this.node = node;
        /**
         * @type {Entorno}
         */
        this.clousure = clousure;
    }

    aridad() {
        return this.node.params;
    }

    /**
     * @type {Invocable['invocar']}
     */
    invocar(interprete, argumentos, node) {
        const nuevoEntorno = new Entorno(this.clousure);

        this.node.params.forEach((parametro, i) => {
            nuevoEntorno.setVariable(parametro.tipo,parametro.id, argumentos[i].valor,node.location.start.line,node.location.start.column);
        });


        const entornoAntesDeEjecutar = interprete.entornoActual;
        interprete.entornoActual = nuevoEntorno;


        try {

            this.node.bloque.accept(interprete);


        } catch (error) {
            interprete.entornoActual = entornoAntesDeEjecutar;
```

Diseño del programa

- index html

En esta parte se realizo todo esqueleto de la pagina web, que contiene botones, textArea



```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Oaklang Editor</title>
    <link rel="stylesheet" href="styles.css">

</head>
<body>
    <div class="container">
        <header class="navbar">
            <div class="navbar-botones">
                <button id="ejecutar" class="navbar-btn">Ejecutar</button>
                <button id="NuevoArchivo" class="navbar-btn">Nuevo Archivo</button>
                <button id="abrir" class="navbar-btn">Abrir</button>
                <button id="guardar" class="navbar-btn">Guardar</button>
                <button id="reporte-errores" class="navbar-btn">Reportes errores</button>
                <button id="reporte-simbolos" class="navbar-btn">Reportes Simbolos</button>
            </div>
        </header>
        <main class="main-content">
            <aside>
                <div class="tabs">
                    <!-- las pestañas se añadirán dinámicamente aquí -->
                </div>
            </aside>
            <section class="editors">
                <h2>Entrada</h2>
                <div class="input-area">
                    <textarea id="codigofuente" class="code-editor" placeholder="Escribe tu código aquí..."></textarea>
                </div>
                <h2>consola</h2>
                <div class="console-area">
                    <textarea id="consola" readonly class="console"></textarea>
```

- index js

En esta parte se realizó la lógica de los botones.

```
import { parse } from './Proyecto1/Compilador/gramatica.js'
import { InterpreterVisitor } from './Proyecto1/Compilador/interprete.js'


const editor = document.getElementById('codigofuente')
const ejecutar = document.getElementById('ejecutar')
const consola = document.getElementById('consola')
const reporteErroresBtn = document.getElementById('reporte-errores')
const reporteSimbolosBtn = document.getElementById('reporte-simbolos')
const modalErrores = document.getElementById('modal-errores')
const modalSimbolos = document.getElementById('modal-simbolos')
const closeButtons = document.getElementsByClassName('close')

export let erroresCompilacion = []
export let tablaSimbolos = []

document.addEventListener('DOMContentLoaded', () => {
    const openBtn = document.getElementById('abrir');
    const saveBtn = document.getElementById('guardar');
    const newFileBtn = document.getElementById('NuevoArchivo');
    const tabsContainer = document.querySelector('.tabs');
    let currentFile = null;
    const fileContents = {};

    // Crear el input de archivo dinámicamente
    const fileInputElement = document.createElement('input');
    fileInputElement.type = 'file';
    fileInputElement.accept = '.oak';
    fileInputElement.style.display = 'none';

    // Manejar el evento de cambio en el input de archivo
    fileInputElement.addEventListener('change', (event) => {
```

- styles css

Aquí solo tiene el diseño de la página.

```css
/* styles.css */

body {
    background-color: #16161a;
    color: #fff;
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

.container {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}

/* Ajuste de la barra de navegación */
.navbar {
    background-color: #72757e;
    padding: 10px 20px; /* Añadir más espaciado interno */
    display: flex;
    justify-content: center; /* Centra los botones */
    align-items: center;
    width: 100%;
}

/* Ajustar la barra de navegación para que no sea tan ancha */
.navbar-botones {
    display: flex;
}

/* Estilo de los botones de la barra de navegación */
.navbar-btn {
    background-color: #72757e;
```