Universidad de San Carlos de Guatemala

Facultad de ingeniera

Escuela de Ciencias y Sistemas

Organización de lenguajes y compiladores 2

QUETZAL OLC2

Luis Fernando Morales Garcia

201503612

Joel Obdulio Xicara Rios

201403975

LENGUAJE DE PROGRAMACIÓN UTILIZADO

Para el desarrollo del programa se utilizo como base el lenguaje TypeScript para luego hacer un traspilado y obtener el código en JavaScript para que la herramienta de análisis funcione de forma correcta.

El desarrollo inicial se debe ejecutar en las clases TS y una vez completado este proceso se debe correr los siguientes comandos para poder traspilar de TS a JS:

1. tsc
2. npm run build

DESARROLLO DE LA INTERFAZ

La interfaz grafica fue desarrollado con HTML y alcenada en un repositorio de Github para poder hacer uso de Github Pages y contar con un dominio de Github Page para presentar el programa de Quetzal.

HERRAMIENTA PARA EL ANÁLISIS

En esta ocasión se utilizo la herramienta de Bison para lo cual es necesario tener instalado:

- npm en su versión 6.9.0
- NodeJs en su versión V8.10.0 para poder proceder con su instalación y correcto funcionamiento

Para la instalación de Jison es necesario ejecutar el comando npm install jison -g para que quede de forma global.

PATRON INTERPRETE

Para el desarrollo de la lógica dell funcionamiento del programa Quetzal se utilizo el patron interprete por medio del cual creamos una clase abstracta llamada instrucción de la cual se va a heredar dos métodos, un método para interpretas y otro para traducir, estos métodos estarán presentes en todas clases en las que implementos la instrucción. Esta clase maneja 4 atributos:

- Fila
- Columna
- Árbol
- Entorno

```
import { AST } from "../AST/AST";
import { Entorno } from "../AST/Entorno";

export interface Instruccion {
    fila: number;
    columna: number;

    interpretar(tree: AST, table: Entorno): any;
    traducir(tree: AST, table: Entorno):any;
}
```

GRAMATICA

Utilizando la herramienta de jison se debe crear una clase .jison en donde se escribirán todas las reglas y producciones que contendrá nuestro lenguaje, este archivo se encuenta en la carpeta raíz de la carpeta en la se encuentra el código JS.

Para realizar el análisis de algún archivo es necesario instanciar la clase AST dentro de el index.js para que el frontend pueda mandarle el texto de entrada al parser y para esto se hace uso de la funcion parse que provee jison y se declara un entorno global que se el primero.

```
console.log("Analizando");
var textoIngresado = document.getElementById('txCodigo').value;

const result = parse(textoIngresado);
const instrucciones = result['instrucciones'];
const errores = result['errores'];

const ast = new AST(instrucciones);
const entornoGlobal = new Entorno(null);
ast.setTSglobal(entornoGlobal);
```

CLASES PRINCIPALES

CLASE AST

Esta clase cuenta con diversos métodos necesarios para poder realizar el interprete y la traducción de forma correcta ya que cuenta con un objeto de instrucciones que hereda de la clase de Instrucción asi como objetos de tipo Funion y Strucs para algunas instrucciones y para menejar cada uno de los ambietes se crea un objeto de tipo Entorno, Adicional se cuenta con variables auxiliares para el proceso de traducción tales como apuntadores y objetos para el Heap y el Stack.

```
export class AST {
    instrucciones: Array<Instruccion>
    public funciones: Array<Funcion>;   //Pull de Funciones
    public structs: Array<Struct>;      //Pull de Strcuts
    excepciones: Array<Excepcion>;      //Pull de Excepciones
    consola: string;
    dot: string;
    contador: number;
    TSGlobal: Entorno;

    contadores: Array<string>;
    contadorTemporal:number;
    posicionContador:number;
    stack:Array<number>;
    heap:Array<number>;
    temporalesAux: Array<TemporalAux>;
    apuntadorStack:number;
    apuntadorHeap:number;

    contadorEtiquetas:number;
    tabla: Array<TemporalAux>;

    casos:Array<Case3d>

    funciones3D:Array<String>;
    main3D:String;
```

CLASE ENTORNO

Esta clase cuenta con dos atributos los cuales son la tabla en la que se almacenan todas las variables que pertenezcan a ese entorno y por ultimo un atributo de tipo Entorno que hace referencia a la tabla que le precede.

```
export class Entorno {
    tabla: { [id: string]: Simbolo };
    anterior: Entorno;

    constructor(anterior = null) { // Revisar ->
        this.tabla = {}; // Diccionario vacio, es
        this.anterior = anterior;
    }
```

CLASE EXCEPCION

Si existe algún error durante el análisis es posible almacenar estos en errores gracias a una clase llamada excepción la cual nos permite guardar el tipo de error, agregar una descripción mas exacta del error asi como almacenar en fila y columna se dio el error.

```
export class Excepcion {
    tipo: string;
    descripcion: string;
    fila: number;
    columna: number;

    constructor(tipo: string, descripcion: string, fila: number, columna: number) {
        this.tipo = tipo;
        this.descripcion = descripcion;
        this.fila = fila;
        this.columna = columna;
    }
```

CLASE SÍMBOLO

Esta clase se utiliza para crear símbolos los cuales serán utilizados durante el análisis del interprete ya que al declarar alguna variable o funcion se puede almacenar su información especifica como el nombre del objeto, el tipo y su valor, adicional también detalles como la fila y columna y una bandera que indica si es arreglo y su tipo de arreglo.

```
export class Simbolo {
    indentificador: string;
    tipo: Tipo;
    fila: number;
    columna: number;
    valor: any;
    tipoArreglo:Tipo;
    tSt:any;

    constructor(identificador: string, tipo: Tipo, fila: number, columna: number, valor: any) {
        this.indentificador = identificador;
        this.tipo = tipo;
        this.fila = fila;
        this.columna = columna;
        this.valor = valor;
        this.tipoArreglo = null;
        this.tSt="";
    }
```

CLASE TIPO

Esta clase contiene listas de los diferentes tipos y operadores que el software será capaz de reconocer y serán de ayuda para las diferentes expresiones posibles.

```
export enum Tipo {
    INT,
    DOUBLE,
    BOOL,
    CHAR,
    STRING,
    ARRAY,
    STRUCT,
    NULL,
    VOID
}

export enum OperadorAritmetico {
    MAS,
    MENOS,
    POR,
    DIV,
    MOD,
    UMENOS,
    CONCATENAR,
    REPETIR,
    MASMAS,
    MENOSMENOS,
}
```

```
export enum OperadorRelacional {
    MENORQUE,
    MAYORQUE,
    MENORIGUAL,
    MAYORIGUAL,
    IGUALIGUAL,
    DIFERENTE
}

export enum OperadorLogico {
    AND,
    OR,
    NOT
}
```

DIRECTORIOS

La herramienta esta divide en tres directorios restantes los cuales contienes todo el desarrollo de las instrucciones posibles, el desarrollo de los operaciones aritméticas y lógicas posibles y por ultimo un directorio con funciones nativas del programa.