Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Catedrático: Ing. Kevin Adiel Lajpop Ajpacaja
Tutor académico: Moises Gonzalez Fuentes

SECCION C

Segundo semestre 2022

MANUAL TÉCNICO
Proyecto 1

MFMScript

Nombre: Moises David Maldonado de León **Carné:** 202010833

Guatemala, 3 de noviembre de 2022

Introducción

En el presente documento, se describe la estructura y los demás aspectos técnicos del proyecto MFMScript, el cual implementa un analizador léxico y sintáctico para la obtención de información y creación de reportes agradables al usuario mediante un arquitectura cliente-servidor. Se conocerá no solo el detalle de estructura del programa sino también el proceso previo, es decir, las expresiones regulares utilizadas en cuanto a la parte del análisis léxico y también la gramática utilizada definida e implementada para el análisis sintáctico. También se conocerá la lógica para la implementación del árbol sintáctico.

OBJETIVOS

Generales

Familiarizar al lector con la lógica planteada para el funcionamiento del sistema realizado mediante el uso del lenguaje de programación TypeScript y Javascript (Para herramienta Jison) así como unas cuantas explicaciones de HTML5 y CSS, para la parte del servido y el cliente respectivamente, así como la estructura de datos para el AST.

Específicos

- Mostrar la estructura del programa realizado.
- Dar a conocer con detalle los métodos y variables empleadas, así como su funcionalidad dentro del programa.
- Mostrar la tabla de tokens, expresiones regulares.
- Mostrar la gramática creada e implementada para la correcta ejecución del programa.

Alcances

El programa se realizó con el fin de funcionar como una base para que nuevos usuarios puedan aprender a programar y tener conocimiento de todas las generalidades de un lenguaje de programación. Además, fue creado con el propósito de crear un entorno agradable donde el usuario puede ingresar interactuar y posteriormente verificar si existen errores en las entradas y a su vez, con una sola acción visualizar la información deseada.

Especificaciones

Requisitos del Hardware

- Mínimo 1GB de memoria RAM
- Procesador Intel Core 2 Duo 2 o superior
- Espacio en disco duro: Mínimo 1 GB

Requisitos del Software

- El programa fue realizado utilizando NodeJs y aplicando la tecnología “express” así como el lenguaje TypeScript y JavaScript para la herramienta Jison.
- El IDE utilizado fue VisualStudioCode.
- Funcional en sistemas operativos como Windows 10 o MAC OS
- Tener instaladas o con acceso a las siguientes librerías y paquetes: Graphviz.

Lógica del programa

Servidor

Carpeta	Descripción
Build	**Tiene los archivos de TypeScript generados en la carpeta src compilados en lenguaje JavaScript.**
Node_Modules	**Carpeta que contiene todos los componentes necesarios para el correcto funcionamiento del servidor. Éstas son dependencias que se agregan al ejecutar el comando “npm install” dentro de la carpeta server.**
Src	**Carpeta principal que contiene los componentes del servidor**

Src

Carpeta	Descripción
arbol	**Contiene las clases de nodo y árbol. Se implementó un árbol n-ario como estructura de datos para la creación del AST. Además, genera el archivo dot de este último.**
AST	**Contiene la clase AST, la cual se encarga de crear el AST, así como la tabla de símbolos y errores. También guarda los archivos dot de errores y símbolos.**
Controllers	**Contiene la clase con todos los métodos para analizar, crear archivo y mandar cada imagen generada.**
Gramática	**Contiene el archivo Jison así como su respectivo compilado en JavaScript**
Routes	**Contiene la clase que contiene las rutas donde se realizan las peticiones para ejecutar los métodos respectivos.**
Index.ts	**Archivo principal que inicializa las rutas y el servidor.**

Árbol – Clase Nodo {}

Método/Función	Descripción
aumentarHijo ()	**Método que agrega hijos al nodo.**
getValor()	**Función que retorna el valor almacenado en el nodo.**
setValor()	**Método para cambiar el valor almacenado en el nodo.**
verNodo()	**Método que muestra en consola la información del nodo y la cantidad de hijos (Método solo para pruebas)**

Árbol – Clase AST {}

Método/Función	Descripción
insertarRaiz ()	**Función que establece el nombre del nodo raíz y retorna el nodo.**
verHijosRecursivo()	**Método que muestra en pantalla, los hijos de un nodo, como es recursivo mostrará todos los hijos de los hijos hasta que termine.**
InsertarRecursivo()	**Método que insertar el nodo, este recorre el árbol ya que solicita el nodo padre. Es recursivo ya que visita cada nodo hasta encontrar al padre.**
Graficar()	**Método que genera una imagen png con el árbol.**

AST – Clase ASTC {}

Método/Función	Descripción
data ()	**Método que recibe un arreglo de instrucciones, implementa un for para recorrer cada instrucción, la cual es pasada como parámetro para le método Instrucciones.**
Instrucciones()	**Método recursivo que recibe un arreglo de 1 instrucción, la primera posición indica el tipo de instrucción. Mediante un switch case se verifica el tipo y en cada caso se van agregando creando subárboles que serán agregados al nodo raíz. Es recursivo para los métodos, funciones, ciclos y condicionales.**
Graficar()	**Método que funciona como puente para llamar el método de graficar de la clase AST{}**
creaTablaSimbolos()	**Método que recibe un arreglo con la lista de símbolos. Este genera una imagen png con la información.**
creaTablaErrores()	**Método que recibe un arreglo con la lista de errores. Este genera una imagen png con la información.**

Controllers – Clase AnalizadorController{}

Función	Descripción
read ()	**Función asíncrona que recibe un texto, ejecuta el análisis léxico y sintáctico, así como la generación de AST y tabla de símbolos. De existir errores se creará la tabla de errores. Retorna respuesta en formato JSON.**
createFile()	**Función asíncrona que recibe un texto y crea un archivo con extensión olc con la información recibida. Retorna respuesta en formato JSON.**
getAST(), getSimbolos(), getErrores()	**Estas funciones asíncronas retornan, cada una de ellas, una url donde se encuentra la imagen respectiva. El cliente la recibe e instancia la imagen mediante una etiqueta <img>**

Gramática

Archivo	Descripción
Gramática.jison	**Archivo que contiene el análisis léxico y sintáctico, desde este archivo se implementa código JavaScript para el llenado de arreglo que emula la tabla de símbolos y errores (Estos se exportan como módulos para el acceso desde el analizadorController.ts).**
Gramática.js	**Para crear el archivo debe ejecutar el comando en la ruta de la carpeta server: npm run jison. Esto compila el archivo jison y genera este archivo el cual es llamado como módulo para invocar el método parse().**

Routes – Clase analizadorRutas{}

Método	Descripción
config ()	**Método que contiene las rutas secundarias y dónde se llaman a los métodos de la clase analizadorController. Se utilizaron peticiones POST y GET.**

Index.ts – Clase server{}

Método	Descripción
Config()	**Método que imprenta distintos módulos y librerías para la configuración de la aplicación. En código se dejó comentarios para cada implementación.**
Routes()	**Método donde se crea la ruta principal la cual invoca a su vez las rutas secundarias.**
Start()	**Método principal que inicializa la aplicación.**

Cliente

Carpeta	Descripción
.angular	**Contiene archivos de angular y caché. No modificar nada**
Node_Modules	**Carpeta que contiene todos los componentes necesarios para el correcto funcionamiento del frontedn. Éstas son dependencias que se agregan al ejecutar el comando "npm install" dentro de la carpeta cliente.**
Src	**Carpeta principal que contiene los componentes del frontend**

Src

Carpeta	Descripción
App	**Carpeta principal, contiene subcarpetas con los componentes de la aplicación.**
Assets	**Carpeta para cargar imágenes locales. Sin uso dentro de la aplicación.**
Enviroments	**Contiene archivos de la configuración inicial de Angular.**

App

Carpeta	Descripción
Components	**Contiene archivos para la navegación de elementos presentes en el encabezo de la página.**
Models	**Contiene archivo con la interfaz que guardará el texto ingresado en el área de texto así como un titulo para guardar documento.**
Plantillas	**Contiene subcarpetas con archivos para agregar headers y footers a todas las páginas creadas.**
Servicios	**Contiene el archivo JavaScript en donde se llama a la url del servidor. También se implementan 2 métodos para hacer la petición de analizar y crear archivo**
Vistas	**Contiene subcarpetas con los componentes de cada vista. Reportes, tabla-Err y tabla-s son vistas que se despliegan cuyo contenido son las imágenes generadas en el servidor.**

Flujo del programa

Análisis Sintáctico

El archivo de gramática se encuentra en la carpeta de Documentación como Gramática.txt

Repositorio: https://github.com/Hrafnyr/OLC1-202010833