UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
LENGUAJES FORMALES Y DE PROGRAMACION
CATEDRÁTICO: INGA. ASUNCIÓN MARIANA SIC SOR
TUTOR ACADÉMICO: ELDER ANIBAL PUM ROJAS

MANUAL TÉCNICO

ELEAZAR NEFTALÍ COLOP COLOP
CARNÉ: 3198935960914
SECCIÓN: A

GUATEMALA, 19 DE DICIEMBRE DEL 2,024

ÍNDICE

ÍNDICE **1**

INTRODUCCIÓN **1**

OBJETIVOS **1**

1. GENERAL 1

2. ESPECÍFICOS 1

ALCANCES DEL SISTEMA **1**

ESPECIFICACIÓN TÉCNICA **1**

● REQUISITOS DE HARDWARE 1

● REQUISITOS DE SOFTWARE 1

DESCRIPCIÓN DE LA SOLUCIÓN **2**

LÓGICA DEL PROGRAMA **2**

❖ NOMBRE DE LA CLASE 2

INTRODUCCIÓN

Este manual técnico está diseñado para proporcionar una guía detallada sobre el funcionamiento y la estructura de nuestro proyecto. Aquí se documenta la lógica del programa, los requisitos técnicos, y cómo operar o modificar el sistema para futuros desarrollos.

OBJETIVOS

1. GENERAL

Proveer una referencia técnica comprensible y clara que facilite la comprensión y manipulación del proyecto por parte de otros programadores.

2. ESPECÍFICOS

2.1. Documentar las funcionalidades principales del programa y cómo están implementadas.

2.2. Describir los componentes del sistema, sus interacciones y el propósito de cada módulo.

2.3. Proveer pasos detallados para instalar y ejecutar el sistema en un entorno de desarrollo.

ALCANCES DEL SISTEMA

Este manual tiene como objetivo explicar los aspectos técnicos del proyecto, incluyendo su instalación, configuración y operación. El enfoque principal está en el análisis léxico, generación de reportes y representación gráfica, asegurando que otro desarrollador pueda entender y trabajar con el código fácilmente.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**

  - Procesador: Dual Core o superior
  - Memoria RAM: 4 GB mínimo
  - Espacio en disco: 500 MB disponible

- **REQUISITOS DE SOFTWARE**

  - Node.js (v16 o superior)
  - NPM (incluido con Node.js)
  - Editor de texto recomendado: Visual Studio Code
  - Graphviz (para generación de grafos, instalar con `npm install graphviz`)

DESCRIPCIÓN DE LA SOLUCIÓN

El proyecto está estructurado con un enfoque modular utilizando JavaScript. Se basa en programación orientada a objetos (POO) para mantener las clases y funciones organizadas y evitar la sobrecarga de métodos en una sola clase. El sistema permite cargar archivos, analizarlos léxicamente, generar reportes en diferentes formatos y realizar análisis de operaciones para generar representaciones gráficas.

LÓGICA DEL PROGRAMA

El sistema inicia con un menú interactivo que permite al usuario navegar por las diferentes funcionalidades del proyecto. Las principales operaciones incluyen:

Módulos Principales

1. **FileLoader:**
    - Cargar y leer archivos.
    - Intentar parsear el contenido como JSON.
2. **AnalizadorLexico:**
    - Analizar texto cargado para identificar tokens y errores léxicos.
    - Generar tablas de tokens y errores.
3. **GeneradorDeReportes:**
    - Crear reportes en formato JSON y HTML para tokens y errores.
4. **OperacionesParser:**
    - Analizar operaciones específicas en el archivo cargado.
5. **GraphGenerator:**
    - Generar grafos representativos basados en los datos procesados.

Menú Principal

El menú principal permite al usuario realizar las siguientes operaciones:

- **Cargar Archivo:** Permite al usuario ingresar la ruta de un archivo y cargar su contenido en el sistema.
- **Analizar Archivo:** Realiza un análisis léxico sobre el contenido cargado.
- **Generar Archivos JSON:** Produce reportes JSON de tokens y errores.
- **Generar Reportes HTML:** Crea reportes visuales en formato HTML para errores y tokens.
- **Analizar Operaciones:** Identifica y procesa operaciones dentro del contenido del archivo.
- **Generar Grafo:** Genera una representación gráfica a partir de datos JSON procesados.

Flujo General

1. El usuario inicia el sistema con un mensaje de bienvenida.
2. Navega por el menú interactivo para seleccionar las operaciones que desea realizar.
3. Las operaciones seleccionadas llaman a los módulos correspondientes para procesar los datos.
4. Los resultados se presentan en la consola o se generan como archivos en el sistema de archivos.

INSTRUCCIONES DE INSTALACIÓN

1. Instale Node.js desde su sitio oficial.
2. Instale Graphviz ejecutando npm install graphviz.
3. Clone el repositorio del proyecto.
4. Navegue al directorio del proyecto y ejecute npm install para instalar dependencias.
5. Ejecute el programa con el comando node index.js.

DISEÑO DE CLASES Y MÓDULOS

El proyecto se organiza en módulos independientes:

- **loaders:** Contiene la clase FileLoader para manejar la carga de archivos.
- **analyzer:** Incluye AnalizadorLexico y los generadores de reportes.
- **operations:** Incluye OperacionesParser y GraphGenerator para análisis y representación gráfica.

Este diseño permite extender fácilmente las funcionalidades del sistema sin alterar las existentes.

Para realizar este proyecto nos basamos en nuestras expresión regular y autómata:

Documentación sobre expresión regular, autómata.

[a-zA-Z][a-zA-Z0-9]*|[0-9]+(\.[0-9]+)?|"[^"]*"|[()+\-*/{}[\]:;,.]|[\t\n\r]|.+#

- **Identificadores y palabras reservadas:**
  [a-zA-Z][a-zA-Z0-9]*
  Representa cadenas que comienzan con una letra y pueden tener letras o dígitos.
- **Números enteros y decimales:**
  [0-9]+(\.[0-9]+)?
  Captura números enteros y opcionalmente decimales.
- **Cadenas de texto:**
  "[^"]*"
  Representa cadenas encerradas entre comillas dobles.
- **Símbolos individuales:**
  [()+\-*/{}[\]:;,.]
  Reconoce caracteres como paréntesis, corchetes, operadores aritméticos, etc.
- **Espacios en blanco:**
  [\t\n\r]
  Reconoce tabulaciones, espacios, saltos de línea y retornos de carro.

- **Caracteres no reconocidos:**
  .+
  Cualquier otro carácter que no cumpla con las reglas anteriores.

Estados

- **S0:** Estado inicial.
- **S1:** Construcción de identificadores o palabras reservadas.
- **S2:** Construcción de números enteros.
- **S3:** Construcción de números decimales.
- **S4:** Construcción de cadenas de texto.
- **S5:** Reconocimiento de símbolos individuales.
- **S6:** Manejo de caracteres no reconocidos (errores léxicos).
- **Sf:** Estado de aceptación final (fin del token).

Tabla de Transiciones

Estado Actual	**Carácter**	**Estado Siguiente**	**Acción**
S0	Letra (a-z, A-Z)	S1	Construir lexema para identificadores o palabras reservadas.
S0	Dígito (0-9)	S2	Construir número entero.
S0	.	S3	Comenzar número decimal.
S0	"	S4	Iniciar construcción de cadena.
S0	{, }, [,], :, ,	S5	Reconocer símbolo individual.
S0	Otro carácter no reconocido	S6	Generar error léxico.
S1	Letra o dígito	S1	Continuar construyendo identificador.
S1	Otro	Sf	Aceptar identificador o palabra reservada.

S2	Dígito	S2	Continuar construyendo número entero.
S2	.	S3	Transitar a número decimal.
S2	Otro	Sf	Aceptar número entero.
S3	Dígito	S3	Continuar construyendo número decimal.
S3	Otro	Sf	Aceptar número decimal.
S4	Carácter diferente de "	S4	Continuar construyendo cadena.
S4	"	Sf	Aceptar cadena.
S5	N/A	Sf	Aceptar símbolo individual.
S6	N/A	Sf	Generar error léxico.
Sf	N/A	S0	Reiniciar para analizar el siguiente token.

IMAGEN DEL AUTOMATA