# Manual técnico

ANALIZADOR LÉXICO, DE UN TORNEO DE FUTBOL

Mishell Monroy

PROYECTO 1, LENGUAJES FORMALES DE PROGRAMACIÓN 202404409

# Índice

# Introducción

Este manual, explica la creación del código, para la elaboración de un analizador léxico para un torneo de futbol, lo que hace es leer un archivo de texto, por medio de una interfaz gráfica en HTML, ahí se sube el archivo y el usuario puede elegir qué operación realizar, el objetivo es analizar dicho archivo, utilizando tokens, para crear palabras reservadas y poder crear un analizador léxico, donde crea el lexema con tokens válidos y no válidos, los cuales se reportan por medio de una tabla, adicional se crean métodos, para graficar el partido y crear reportes usando la información del archivo de texto.

## **Objetivos**:

Brindar una idea clara de cómo fue creado el programa, para que cualquier otra persona pueda interpretar de una buena manera el código creado, incluso es viable ver este manual para cuando se le desea hacer una actualización el programa, se sabe dónde se debe de hacer, debido a que en este documento se explica de forma clara y ordenada las funciones del código.

## **Dirigido**

Está dirigido a todas aquellas personas que deseen modificar o actualizar el programa, también para quienes deseen un programa como este, para que sepan cómo deben leer el código y como es su funcionamiento, incluso está dirigido a el programador en sí, ya que de esta forma puede saber de una manera más clara y ordenada que es y donde está el código que desea implementar o actualizar.

# Especificación técnica

## Requisitos de hardware

- Disco duro capaz de correr el programa, a partir de 250 GB
- Procesador Core i5 o superior
- 4GB de memoria RAM como mínimo.

## Requisitos de Software

- Debe tener instalado algún editor de código fuente, para usar el programa directamente, Computadora de escritorio o portátil, se recomienda utilizar Visual Studio Code.
- Node, version v22.17.1
- Instalar previamente el Java

# Lógica del programa

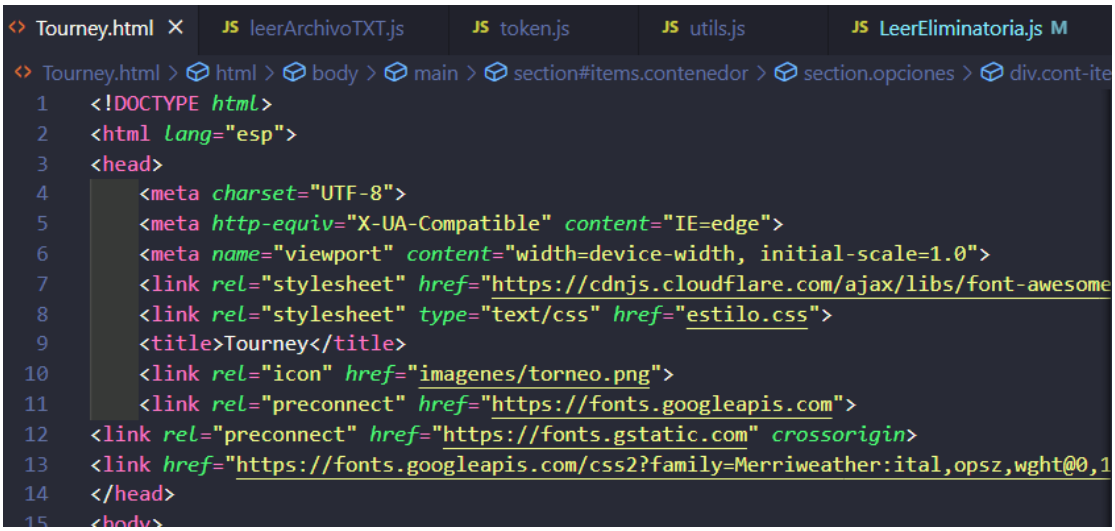
## Diccionario de palabras especiales:

I. FileReader: sirve para poder leer el archivo txt desde la página HTML
II. addEventListener: sirve para expresar una acción realizada en el HTML, en este caso, expresa el “click” en cierto botón, para realizar alguna acción especifica.
III. createObjectURL: sirve para generar una URL única y temporal que apunta directamente a un objeto en memoria, como un archivo seleccionado por el usuario (un objeto Blob o File).
IV. Blob: es un tipo de URL, especifica que se puede usar en atributos como src de etiquetas <img>, <video> o <audio> para mostrar contenido binario sin necesidad de subirlo a un servidor.
V. alert: se usa para que el navegador despliegue una alerta al realizar cierta acción.

# Interfaz gráfica

## HTML

La interfaz gráfica es por medio de un archivo .html, en este caso se llama “Tourney.html” y desde acá es donde se realizan todas las interacciones con el usuario. Se utilizó CSS, para modificar el estilo del HTML, enlazándolo por medio del rel=”stylesheet” href=”estilo.css”




Se usan 4 botones, para poder:

1. subir un archivo txt
2. analizar el archivo txt
3. exportar reportes
4. exportar grafico con graphiz

para poder utilizar los scripts, se crean ciertos divs, donde se colocará lo creado con JS

```
<div class="reportes_analisis">
    <h2 class="subtitulo">Analisis Léxico</h2>
    <div id="analisisReporte"></div> <!-- para mostrar el reporte del analisis -
    <button id="dowlandLexico" class="boton2 ">Descargar Analisis</button>
</div>

<div id="reportes">
    <h2 class="subtitulo">Reportes Generados</h2>
    <div id="reporte"></div> <!-- para mostrar el reporte generado -->
    <div id="tablaResumen"></div>
<div id="tablaFases"></div>
</div>
```

## Lógica del programa

Para este programa se utilizó HTML, CSS y JS, se usó un paquete Json, para poder programar de forma modular, usando type"module"

```
{
    "name": "lfp-proyecto1-202404409",
    "version": "1.0.0",
    "description": "Proyecto 1 - habilitar módulos ES (type: module)",
    "main": "main.js",
    "type": "module",
```

## Archivos JS utilizados

1. tokens.js
2. lexico.js
3. Analizador_Texto.js
4. utils.js
5. LeerEliminatoria.js

**Token.js**

Aquí se define un diccionario con los atributos validos, lexemas, y el constructor del token.

El constructor del token defina que atributos e información lleva cada token, información que será necesaria para crear el analizador lexico.

```
export class token{
    constructor(tipo, lexema, linea, columna){
        this.tipo = tipo;
        this.lexema = lexema;
        this.linea = linea;
        this.columna = columna;
    }
}
```

Diccionario para las palabras reservadas: aquí están todas las palabras que el txt usa para identificar fases, torneos, estas palabras son las que se usan para recaudar la información del txt

```
//creacion de diccionario para palabras reservadas
export const palabrasReservadas = [
    "TORNEO",
    "EQUIPOS",
    "ELIMINACION",
    "equipo",
    "jugador",
    "partido",
    "resultado",
    "goleador"
];
```

Diccionario de caracteres validos: son esas palabras que son validas al leer el archivo, si viene alguna diferente, se marca como error léxico.

```
//diccionario de atributos validos
export const AtributosValidos = [
    "nombre",
    "sede",
    "posicion",
    "equipos",
    "edad",
    "numero",
    "cuartos",
    "semifinal",
    "final",
    "vs",
    "goleadores",
    "minuto"
];
```

Diccionario de símbolos: símbolos que vienen en el txt, que son válidos, y que sirven para leer el archivo.

```
//diccionario de simbolos
export const Simbolos = {
    "{" : "Lave Izquierda",
    "}" : "Llave Derecha",
    "[" : "Corchete Izquierdo",
    "]" : "Corchete Derecho",
    "(" : "Parentesis Izquierdo",
    ")" : "Parentesis Derecho",
    ":" : "Dos Puntos",
    ";" : "Punto y Coma",
    "," : "Coma",
    "." : "Punto"
}
```

**Lexico.js**

Aquí se defina la clase Lexico() y analizar(), en la clase Lexico() se genera el constructor de como viene el texto, esto sera usado para analizar el archivo.

```
import { token, palabrasReservadas, AtributosValidos, Simb

export class Lexico {
    constructor(texto) {
        this.texto = texto;
        this.position = 0;
        this.linea = 1;
        this.columna = 1;
        this.estado = "INICIO";
        //this.currentChar = this.texto[this.position];
        this.tokens = [];
        this.errores = [];
    }
```

Analizar() utiliza la información de Lexico() para analizar el texto, validando los tokens línea por línea y definiendo si hay algún error y que tipo de token son.

```
analizar() {
    while (this.position < this.texto.length) { //an
        let char = this.texto[this.position];

        //Estado AFD=inicio
        if(this.estado === "INICIO"){
            //ignorar espacios y saltos de linea
            if (char === ' ' || char === '\t') {
                this.avanzar();
                continue; //siguiente caracter
            }
            //salto de linea
            if (char === '\n') {
                this.linea++;
                this.columna = 1;
                this.position++;
                continue;
            }
```

**Analizador_Texto.js**

Aquí se crea toda la interacción con los botones,  se definen los atributos que se usarán para crear el DOM, que es el archivo que será leído, para poder analizar el texto y crear una tabla con todos los tokens validos e invalidos, desde acá se crea el “descargar reporte” donde se puede descargar un html, con la tabla de tokens, y el “descargar graphviz” donde se puede descargar un txt con el grafico de el torneo.

```
import { Lexico } from './lexico.js';
import { extraerAtributos, generarDot } from './utils.js';

//elementos para manipular el DOM
const txtFile = document.getElementById('txtFile'); // id correcto
const textArea = document.getElementById('textArea'); // nombre consi
const btnAnalizar = document.querySelector('.analizar');
const analisisReporte = document.getElementById('analisisReporte');
const reporteBtn = document.getElementById('btnReporte');
const btnDowland = document.getElementById('dowlandLexico');
const btnDowlandGraphiz = document.getElementById('dowlandGraphiz');

let ultimoReporte =""; // Variable para almacenar el último reporte g

//cargar archivo de texto
if (txtFile) {
    txtFile.addEventListener('change', (e) => {
        const file = e.target.files[0];
        if (!file) return;

        const reader = new FileReader();
        reader.onload = function (e) {
            textArea.value = e.target.result;
        };
        reader.readAsText(file);
```

**Utils.js**

En esta clase se separó un método para extraer los atributos “Torneo” y “sede” del archivo txt para luego poder mostrarlos en el gráfico, que también se declara acá, esto hace modular el código y mas limpio, así en la clase correspondiente, solo se hace llamar a esta función.

```
// Función para extraer pares atributo-valor desde la lista de tokens
export function extraerAtributos(tokens) {
    const atributos = {};
    for (let i = 0; i < tokens.length - 1; i++) {
        const actual = tokens[i];
        const siguiente = tokens[i + 1];

        if (actual.tipo === "ATRIBUTO" &&
            (siguiente.tipo === "CADENA" || siguiente.tipo === "NUMERO")) {
            atributos[actual.lexema] = siguiente.lexema;
        }
    }
    return atributos;
}
```

Creación del graphviz:

```
// Función para generar el código DOT de Graphviz con los atributos extraído
export function generarDot(atributos) {
    const nombre = atributos["nombre"] || "Nombre no encontrado";
    const sede = atributos["sede"] || "Sede no encontrada";

    return `
digraph Torneo {
    graph [labelloc="t", fontsize=20];
    node [shape=box, style=filled, color="#f0f0f0", fontname="Arial"];

    Torneo [label="Nombre: ${nombre}"];
    Sede [label="Sede: ${sede}"];
    Torneo -> Sede [label="se juega en", color="#0077cc"];
}
`;
}
```

## **LeerEliminatoria**.js

En esta clase se lee el archivo txt, para poder extraer la información correspondiente y mostrarla en el HTML.

Se usa FileReader para poder capturar el archivo y leerlo, a continuación se separan los datos dentro, para poder crear 3 tablas que generen los reportes del partido.

```
let archivoSeleccionado = null // Variable global para almacenar el archivo selecci

//capturar el archivo seleccionado
document.querySelector("#txtFile").addEventListener("change", function(event) {
    archivoSeleccionado = event.target.files[0];
});

// Al presionar el botón, se analiza el archivo
document.querySelector(".presionar").addEventListener("click", function() {
    if (archivoSeleccionado) {
        leerTXT(archivoSeleccionado);
    } else {
        alert("Primero selecciona un archivo.");
    }
});
```

Se crea un HTML interno, para poder ser visto desde el HTML principal y adicionalmente se crea un método para descargar el archivo en un tipo.html

```
function mostrarYDescargarReporte({ torneo, fases, estadisticas, goleadores }) {
    let html = `
        <h1>${torneo.nombre}</h1>
        <p><strong>Sede:</strong> ${torneo.sede}</p>
        <p><strong>Equipos participantes:</strong> ${torneo.equipos}</p>

        <h2> Fases del Torneo</h2>
        <table border='1'><tr><th>Fase</th><th>Partido</th><th>Resultado</th><th>Gar
    `;

    fases.forEach(f => {
        html += `<tr><td>${f.fase}</td><td>${f.equipo1} vs ${f.equipo2}</td><td>${f.
    });

    html += `</table>
        <h2> Estadísticas por Equipo</h2>
        <table border='1'><tr><th>Equipo</th><th>Jugados</th><th>Ganados</th><th>Per
    `;
```