Universidad de San Carlos de Guatemala
Centro Universitario de Occidente
División de Ingeniería
Carrera: Ingeniería en Ciencias y Sistemas
Curso: Laboratorio, Lenguajes Formales y de Programación
Auxiliar: Carlos Pac
Docente: Ing. Bryan Monzón

Trabajo: Manual Técnico, práctica 1

Estudiante: Jefferson Salvador Gómez Menchú
jeffersongomez201830892@cunoc.edu.gt

Registro académico: 201830892

Quetzaltenango 06-03-2025

Objetivos

Objetivo General

Comprender los principios básicos de Java Script, HTML y CSS, y aplicarlos para crear una solución de software de un analizador léxico.

Objetivos específicos

- Identificar una solución acorde a una necesidad.
- Realizar una toma de requerimientos basado en una necesidad previamente identificada.
- Implementar una solución a través de programación web.
- Explicar el código utilizado tanto en Javascript como en HTML

Explicación de la Arquitectura y Tecnologías utilizadas

Arquitectura:

1. **FrontEnd (HTML)**
    - Index.html: maquetación del dashboard principal y sus componentes
    - Styles.css: diseño y personalizacion de los componentes modulares (divs, container, headers, body, footer, tables) (no se hizo uso de boostrap.

2. **BackEnd (JavaScript)**
    - **Estructura de archivos:** Donde se encuentran el punto de entrada que es el app.js, donde están las rutas debidas para conectar con el FrontEnd.

Tencologías utilizadas:

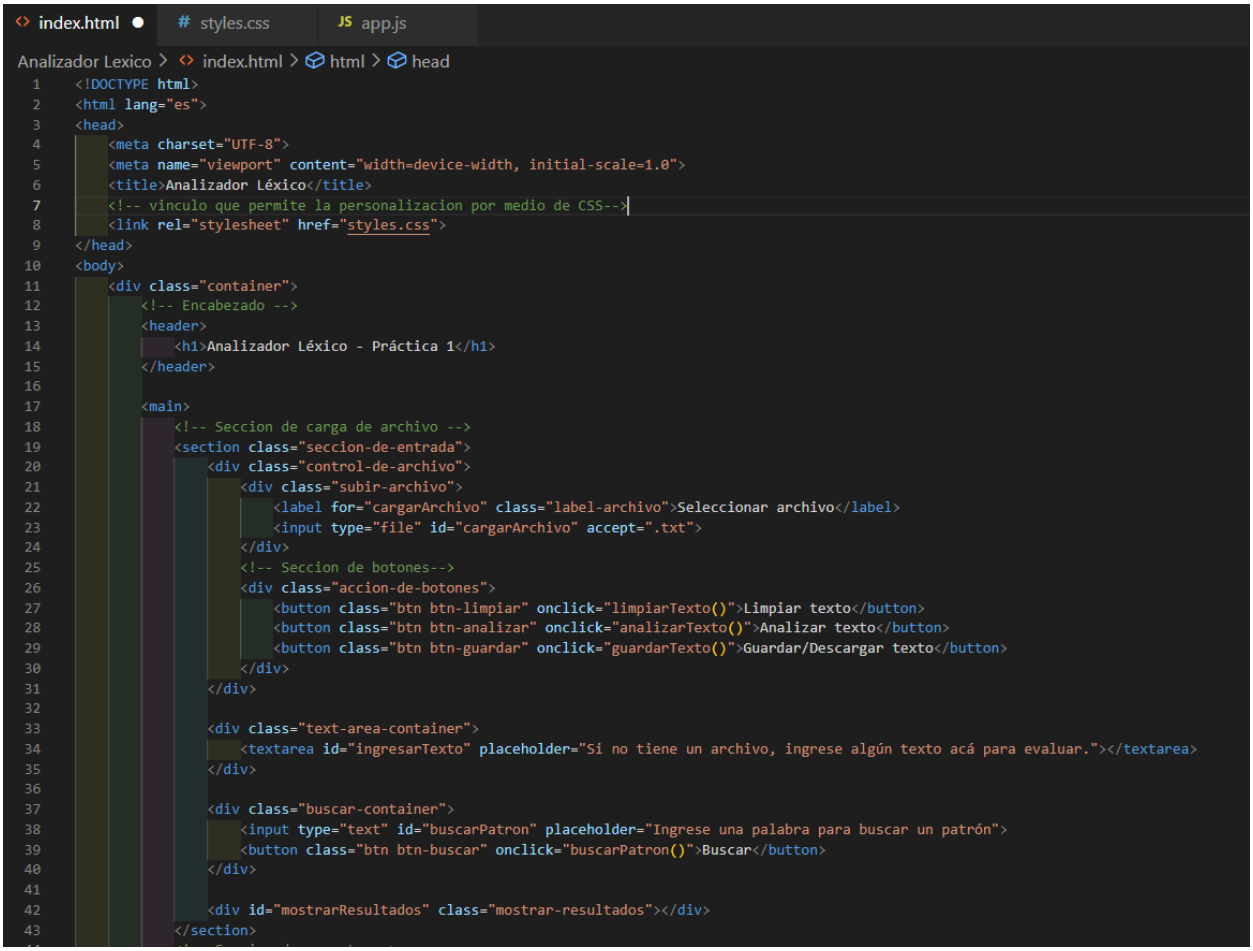
1. **FrontEnd:**
    - HTML 5
    - CSS
    - LiveServer

2. BackEnd:
    - Javascript

FrontEnd:

En esta parte del codigo se maqueta lo que es el encabezado, la carga de archivos y la seccion de botones, cada apartado tiene su respectivo div, esto con el objetivo que durante la personalizacion no haya tanto problema con los elementos y cada uno sea independiente.

Se crea el laber que nos permitira la carga de archivos dentro de nuestra aplicación web, así como también se maquetan los botones que nos ayudaran con las funcionalidades completas de la página.



En el siguiente apartado se muestran como se encuentra la maquetacion del área de reportes, con sus respectivas tablas

```html
<!-- Seccion de reportes-->
<section class="seccion-reportes">
    <div class="area-reporte">
        <h2>Reporte de Errores</h2>
        <div class="table-container">
            <table id="reporteErrores">
                <thead>
                    <tr>
                        <th>Error</th>
                        <th>Descripción</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>
    </div>

    <div class="area-reporte">
        <h2>Reporte de Tokens</h2>
        <div class="table-container">
            <table id="reporteTokens">
                <thead>
                    <tr>
                        <th>Token</th>
                        <th>Tipo</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </div>
    </div>
```

Por último aparece el footer, donde va la informacion del desarrollador y también el vinculo que permite la conexión con javascript

```
        <!-- Footer-->
        <footer>
            <p> 2025 Jefferson Gómez [201830892]- Lenguajes Formales y de Programación</p>
        </footer>
    </div>
<!-- Vinculo que permite conexion con javascript-->
    <script src="app.js"></script>
</body>
</html>
```

Estilos css:

En este apartado unicamente se usaron estilos basicos para poder dar personalizacion a la página y que sea un poco más dinamica y amigable con el usuario, en este apartado se encuentra las configuraciones globales o principales, se deja colores predeterminados para usar más adelante, asi como los bordes, sombras, y fondo. También se tienen en cuenta fuentes de tipo de letra que nos puedes ayudar a estilizar de mejor manera la aplicación

```
:root {
    --primary-color: #3498db;
    --secondary-color: #2c3e50;
    --text-color: #333;
    --light-text: #ecf0f1;
    --border-color: #bdc3c7;
    --metallic-light: #e0e0e0;
    --metallic-dark: #7f8c8d;
    --card-bg: rgba(255, 255, 255, 0.9);
    --shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    --transition: all 0.3s ease;
}

 * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
    line-height: 1.6;
    color: var(--text-color);
    background: linear-gradient(135deg, var(--metallic-light) 0%, var(--metallic-dark) 100%);
    background-attachment: fixed;
    min-height: 100vh;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
}

h1,
h2 {
    color: var(--secondary-color);
    margin-bottom: 20px;
}
```

Acá se observa la manera en que está estilizada tanto el área del header, y la seccion de cargar archivo, con los margenes, color de fondo, display y justificacion del contenido

```css
header {
    margin-bottom: 30px;
    padding: 20px;
    background-color: var(--card-bg);
    border-radius: 10px;
    box-shadow: var(--shadow);
}

main {
    display: grid;
    gap: 30px;
}


.seccion-de-entrada {
    background-color: var(--card-bg);
    border-radius: 10px;
    padding: 20px;
    box-shadow: var(--shadow);
}

.control-de-archivo {
    display: flex;
    flex-wrap: wrap;
    gap: 15px;
    margin-bottom: 20px;
    justify-content: space-between;
}

.subir-archivo {
    position: relative;
    overflow: hidden;
}

.label-archivo {
    display: inline-block;
    padding: 10px 15px;
    background-color: var(--secondary-color);
    color: var(--light-text);
    border-radius: 5px;
```

Acá se observa la estilización de los botones que nos serviran para aplicar funcionalidad en la aplicación. Se observa la configuracion del textArea

```css
.btn {
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-weight: 600;
    transition: var(--transition);
    color: var(--light-text);
}

.btn-limpiar {
    background-color: #34495e;
}

.btn-analizar {
    background-color: #34495e;
}

.btn-guardar {
    background-color: #084823;
}

.btn-buscar {
    background-color: #34495e;
}

.btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

.text-area-container {
    margin-bottom: 20px;
}

textarea {
    width: 100%;
    height: 200px;
    padding: 15px;
```

Acá se tiene la personalizacion de la seccion de reportes, al igual que sus tablas, para que muestre la inforacion idonea y luzca estetico

```css
.seccion-reportes {
    display: grid;
    gap: 20px;
}

.area-reporte {
    background-color: var(--card-bg);
    border-radius: 10px;
    padding: 20px;
    box-shadow: var(--shadow);
}

.table-container {
    overflow-x: auto;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 10px;
}

table,
th,
td {
    border: 1px solid var(--border-color);
}

th {
    background-color: var(--secondary-color);
    color: var(--light-text);
    padding: 12px 15px;
    text-align: left;
}
```

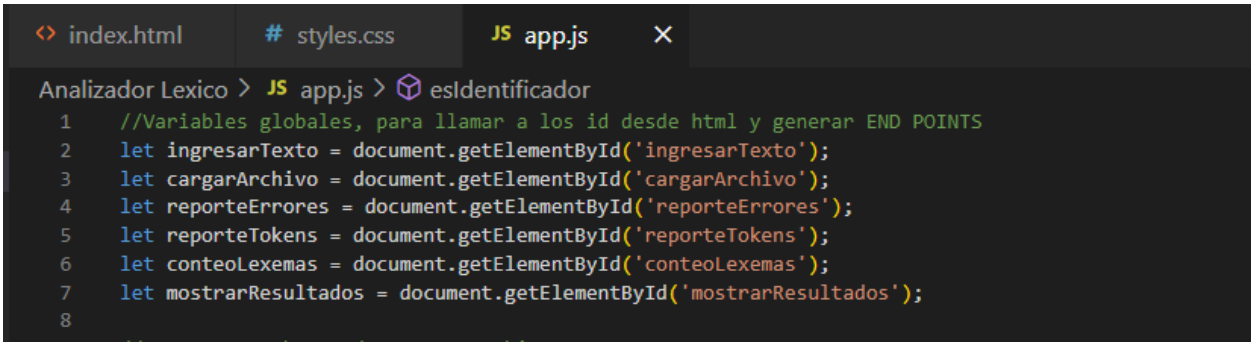
Por ultimo se ve el diseño del footer, y de las secciones que requieren un diseño responsivo para que los divs que los almacenan no sufran ningun movimiento cambio brusco despues de la maquetacion

```css
  /* Footer */
footer {
    text-align: center;
    margin-top: 40px;
    padding: 20px;
    color: #121212;
    font-size: 0.9rem;
    font-weight: bold;
}

  /* Responsive Design */
@media (min-width: 768px) {
    .seccion-reportes {
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    }
}

@media (max-width: 767px) {
    .control-de-archivo {
        flex-direction: column;
    }

    .accion-de-botones {
    width: 100%;
    justify-content: space-between;
    }

    h1 {
    font-size: 2rem;
    }
}
```

BackEnd:

Endpoints:

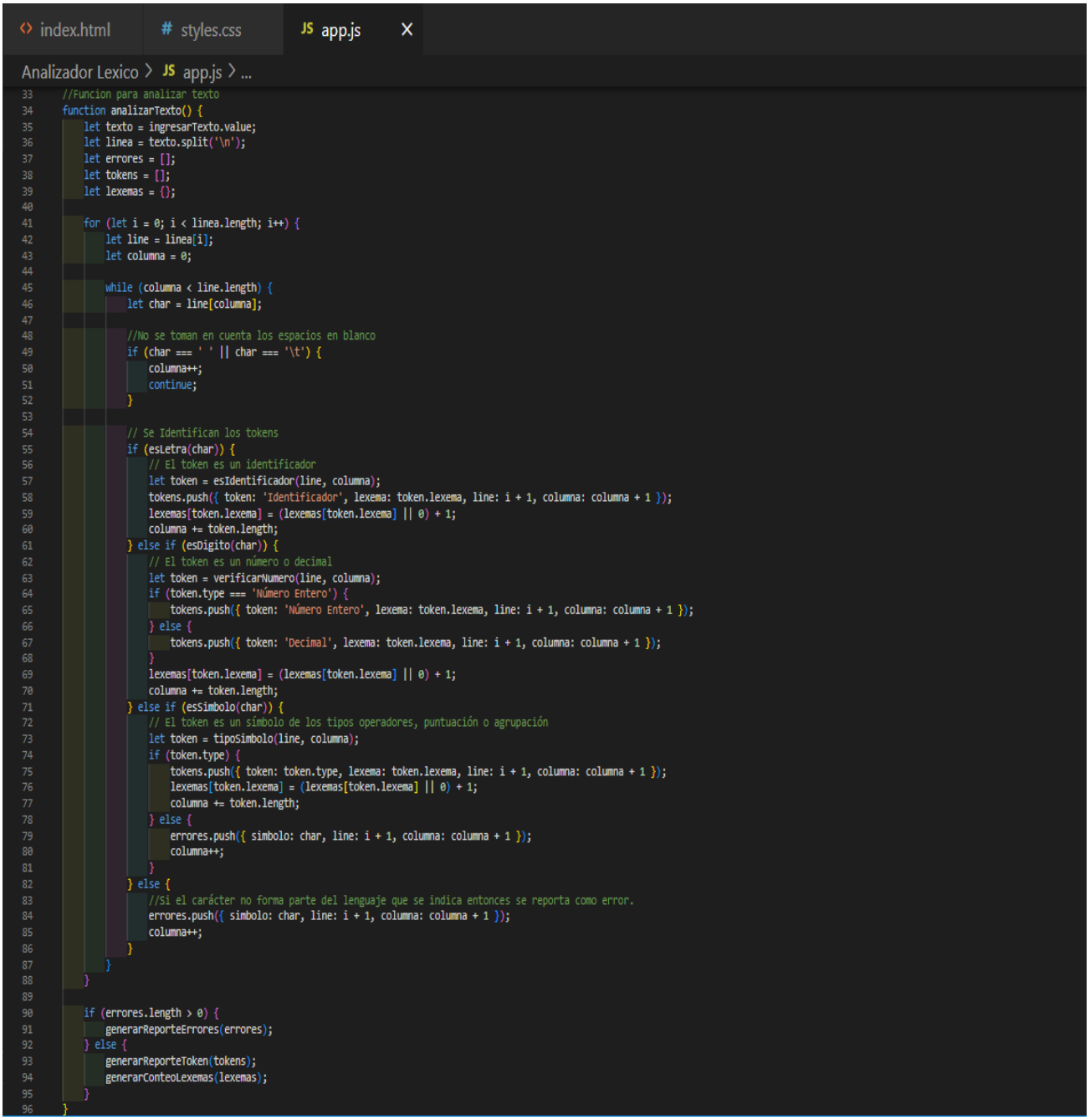
Puntos de acceso que conectan JavaScript con HTML



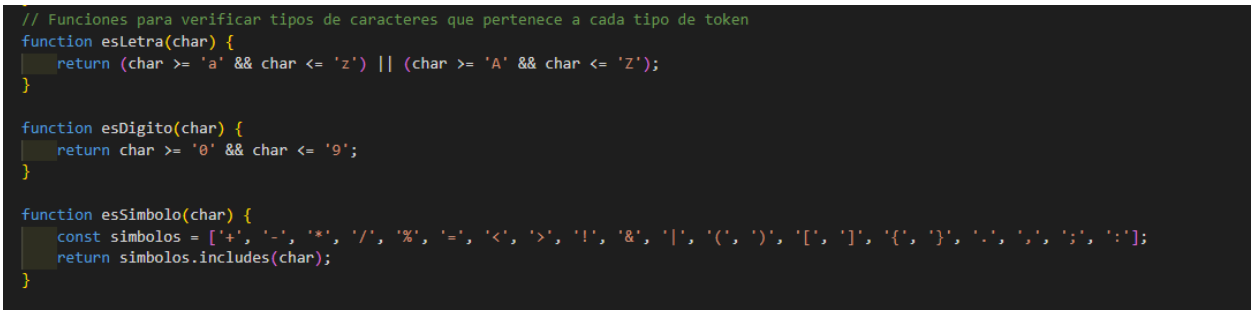
Funciones que permiten la subida de una archivo por medio de un selector de archivos gracias a la implementacion de FileReader y también la funcion de limpiar texto, donde simplemente se emplea todos los valores como vacios, para poder reiniciar el proceso de analizar

```
//Evento para boton de cargar archivo
cargarArchivo.addEventListener('change', function(event) {
    let file = event.target.files[0];
    if (file) {
        let reader = new FileReader();
        reader.onload = function(e) {
            ingresarTexto.value = e.target.result;
        };
        reader.readAsText(file);
    }
});

//Funcion para limpiar texto
function limpiarTexto() {
    ingresarTexto.value = '';
    cargarArchivo.value = '';
    buscarPatron.value = '';
    reporteErrores.innerHTML = '';
    reporteTokens.innerHTML = '';
    conteoLexemas.innerHTML = '';
    mostrarResultados.innerHTML = '';
}
```

Por medio de ciclos anidados podemos establecer la funcion de analizar texto, ya que se establece un for que por medio de una varibale va evaluando linea por línea y carácter por carácter ya que la misma cadena ha sido separada por medio de la funcion split, entonces se puede evaluar cada carácter, y por medio del while anidado se va tomando cada caracter como una columna diferente para poder realizar las evaluaciones necesarias, se toman instancias de funciones que más adelante profundizaremos, esto con el fin de evaluar el tipo de token que es cada carácter o en su defecto si tiene cierto tipo de estructura, la cadena completa se establece como un token determinado. Por ultimo se realiza una validacion para determinar si hay erorres se genera un reporte de errores, de lo contrario genera un reporte de tokens y lexemas.

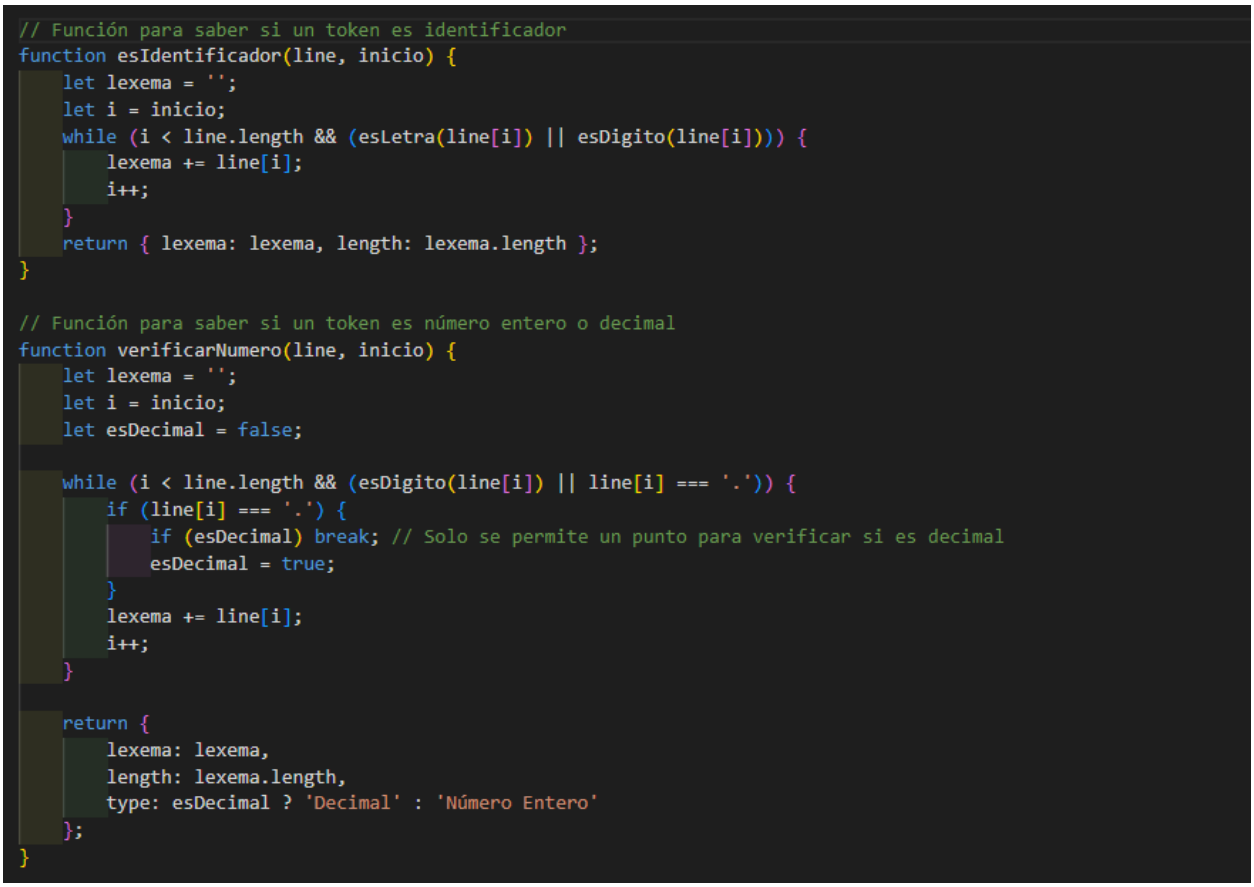


Acá se muestran los caracteres permitidos cada cada tipo de token, de está manera tenemos una mejor precision en nuestro análisis



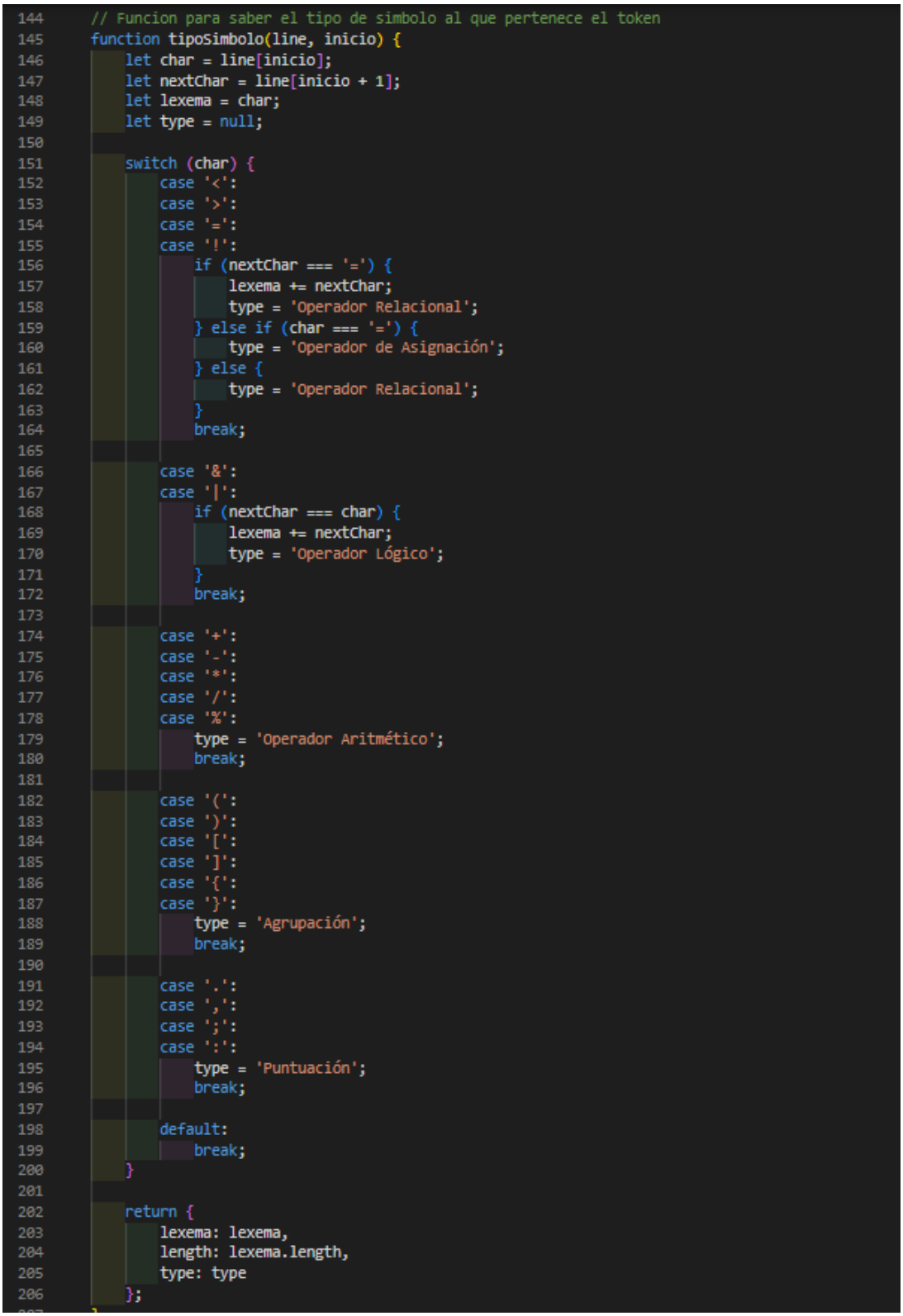
```
// Funciones para verificar tipos de caracteres que pertenece a cada tipo de token
function esLetra(char) {
    return (char >= 'a' && char <= 'z') || (char >= 'A' && char <= 'Z');
}

function esDigito(char) {
    return char >= '0' && char <= '9';
}

function esSimbolo(char) {
    const simbolos = ['+', '-', '*', '/', '%', '=', '<', '>', '!', '&', '|', '(', ')', '[', ']', '{', '}', '.', ',', ';', ':'];
    return simbolos.includes(char);
}
```

Acá se realizan las validaciones respectivas para saber el tipo de token que es el carácter o en su defecto la cadena de datos, por medio de un while que recorre la longitud de la palabra o numero y en caso de que lo sea se incremente un contador, para llevar la contabilidad de los lexemas



```
// Función para saber si un token es identificador
function esIdentificador(line, inicio) {
    let lexema = '';
    let i = inicio;
    while (i < line.length && (esLetra(line[i]) || esDigito(line[i]))) {
        lexema += line[i];
        i++;
    }
    return { lexema: lexema, length: lexema.length };
}

// Función para saber si un token es número entero o decimal
function verificarNumero(line, inicio) {
    let lexema = '';
    let i = inicio;
    let esDecimal = false;

    while (i < line.length && (esDigito(line[i]) || line[i] === '.')) {
        if (line[i] === '.') {
            if (esDecimal) break; // Solo se permite un punto para verificar si es decimal
            esDecimal = true;
        }
        lexema += line[i];
        i++;
    }

    return {
        lexema: lexema,
        length: lexema.length,
        type: esDecimal ? 'Decimal' : 'Número Entero'
    };
}
```

Por el contrario, si el carácter es representado por un simbolo la validacion se realiza por medio de un switch que nos permite una estructura más organizada y una estetica visual, así como una mejor comprension en el proceso de desarrollo de codigo



```
// Funcion para saber el tipo de simbolo al que pertenece el token
function tipoSimbolo(line, inicio) {
    let char = line[inicio];
    let nextChar = line[inicio + 1];
    let lexema = char;
    let type = null;

    switch (char) {
        case '<':
        case '>':
        case '=':
        case '!':
            if (nextChar === '=') {
                lexema += nextChar;
                type = 'Operador Relacional';
            } else if (char === '=') {
                type = 'Operador de Asignación';
            } else {
                type = 'Operador Relacional';
            }
            break;

        case '&':
        case '|':
            if (nextChar === char) {
                lexema += nextChar;
                type = 'Operador Lógico';
            }
            break;

        case '+':
        case '-':
        case '*':
        case '/':
        case '%':
            type = 'Operador Aritmético';
            break;

        case '(':
        case ')':
        case '[':
        case ']':
        case '{':
        case '}':
            type = 'Agrupación';
            break;

        case '.':
        case ',':
        case ';':
        case ':':
            type = 'Puntuación';
            break;

        default:
            break;
    }

    return {
        lexema: lexema,
        length: lexema.length,
        type: type
    };
```

En este apartado se encuentran los reportes generados unicamente en formato de tabla, con maquetacion de html, con los respectivos requisitos de la práctica y cada dato acorde a lo que se pide.

```
//Funciones para poder generar los reportes de errores, tokens y lexemas

function generarReporteErrores(errores) {
    reporteErrores.innerHTML = '<tr><th>Símbolo</th><th>Fila</th><th>Columna</th></tr>';
    errores.forEach(error => {
        let fila = `<tr><td>${error.simbolo}</td><td>${error.line}</td><td>${error.columna}</td></tr>`;
        reporteErrores.innerHTML += fila;
    });
    reporteTokens.innerHTML = '';
    conteoLexemas.innerHTML = '';
}

function generarReporteToken(tokens) {
    reporteTokens.innerHTML = '<tr><th>Token</th><th>Lexema</th><th>Fila</th><th>Columna</th></tr>';
    tokens.forEach(token => {
        let fila = `<tr><td>${token.token}</td><td>${token.lexema}</td><td>${token.line}</td><td>${token.columna}</td></tr>`;
        reporteTokens.innerHTML += fila;
    });
    reporteErrores.innerHTML = '';
}

function generarConteoLexemas(lexemas) {
    conteoLexemas.innerHTML = '<tr><th>Lexema</th><th>Cantidad</th></tr>';
    for (let lexema in lexemas) {
        let fila = `<tr><td>${lexema}</td><td>${lexemas[lexema]}</td></tr>`;
        conteoLexemas.innerHTML += fila;
    }
}
```

En el apartado de patrones se redirige al html para poder conocer el valor que se buscara y de igual forma que al realizar las validaciones, se procede a buscar carácter por carácter, se declara una variable que va a ser la encargada de subrayar los caracteres encontrados y que coincidan por medio de la funcion “.replace”, que contiene informacion gracias a la funcion RegExp, que nos permite la busqueda a niveles más especificos dentro de los datos de informacion que tenemos en el archivo de texto que fue ingresado.

```
//Funcion para poder buscar patrones en base a una cadena (palabra/numero), y subrayar las respuestas en caso de que haya alguna
function buscarPatron() {
    let patron = document.getElementById('buscarPatron').value;
    let texto = ingresarTexto.value;
    let regex = new RegExp(patron, 'g');
    let matches = texto.match(regex);
    let cont = matches ? matches.length : 0;

    let linea = texto.split('\n');
    let textoSubrayado = '';

    // Resaltar coincidencias en cada línea
    linea.forEach(linea => {
        let lineaSubrayada = linea.replace(regex, match => `<span style="background-color: gray;">${match}</span>`);
        textoSubrayado += `<div>${lineaSubrayada}</div>`;
    });

    // Mostrar resultados
    let resultHTML = `<p><strong>Patron buscado:</strong> ${patron}</p>`;
    resultHTML += `<p><strong>No. Repeticiones:</strong> ${cont} veces</p>`;
    resultHTML += `<div><strong>Texto resaltado:</strong><br>${textoSubrayado}</div>`;

    mostrarResultados.innerHTML = resultHTML;

}
```

Por último tenemos la funcion de guardar y descargar el archivo de entrada de texto si asi se desea, para está funcion se hizo uso de Blob, que permite la descarga de archivos, así como el conocimiento de su tamaño y de la funcion “URL.createObjectURL”, cuya funcion es extraer la informacion del Blob y de está manera descargar el archivo a la computadora. Despues de hacer click en el botón establecido, luego simplemente con referencias, se le brinda un nombre predeterminado para que quede almacenado en nuestra computadora.

```
//Funcion para poder guardar y descargar la entrada de texto en caso que asi se quiera

function guardarTexto() {
    let texto = ingresarTexto.value;
    let blob = new Blob([texto], { type: 'text/plain' });  //Permite la descarga del archivo editado o generado en el editor de texto
    let url = URL.createObjectURL(blob);
    let a = document.createElement('a');
    a.href = url;
    a.download = 'texto_analizado_correctamente.txt';
    a.click();
    URL.revokeObjectURL(url);
}
```