INFORME PROYECTO FINAL

Informe análisis léxico

Materia:

Compiladores

Profesor:

Jorge Leonardo Gamarra Ospina

Estudiantes:

María Alejandra Reina

Jessica Tascón

Sebastian Ramirez

Universidad Remington

Sede-Tuluá

TABLA DE CONTENIDO

1. Introducción

Un compilador es una herramienta fundamental en el proceso de desarrollo de software. Se encarga de traducir el código fuente de un programa escrito en un lenguaje de programación específico a un código ejecutable que la computadora puede entender. En otras palabras, convierte el código legible por humanos en instrucciones que la máquina puede procesar y ejecutar.

Para comprender mejor su funcionamiento, podemos tomar como ejemplo un programa escrito en JavaScript, un lenguaje ampliamente utilizado en el desarrollo web. Este tipo de programas contiene instrucciones que deben ser interpretadas por la computadora, pero JavaScript, como muchos lenguajes de alto nivel, no es directamente comprensible por el hardware. El compilador cumple la función de traducir ese código fuente a un lenguaje que la máquina pueda entender y ejecutar. Durante este proceso, analiza la estructura del programa, verifica su corrección y genera un archivo ejecutable o código optimizado que puede ser interpretado eficientemente. Así, el compilador actúa como un puente entre el programador y el sistema que ejecutará las instrucciones.

Dentro de este proceso, una de las etapas más importantes es el análisis léxico. Esta fase tiene como entrada el código fuente del lenguaje aceptado por el compilador y como salida proporciona al analizador sintáctico una secuencia de tokens, que son unidades léxicas significativas como palabras clave, identificadores, operadores, números, entre otros. El análisis léxico permite estructurar y simplificar el código, y es clave para que las etapas posteriores del compilador funcionen correctamente.

Con este proyecto, se buscó desarrollar un analizador léxico básico como parte del funcionamiento de un compilador. El objetivo fue comprender de manera práctica cómo se identifican los tokens, cómo se manejan los errores léxicos, y cómo se construye una herramienta que traduzca texto plano en una estructura más formal que sirva de base para las siguientes fases de la compilación. También se pretendió fortalecer el entendimiento de los procesos internos de los compiladores y aplicar conocimientos de programación en la implementación de esta fase específica.

2. Marco Teórico

Definición técnica de análisis léxico

El **análisis léxico** es la primera fase del proceso de compilación, encargada de leer el código fuente y transformarlo en una secuencia de unidades significativas llamadas *tokens*. Su objetivo principal es identificar patrones léxicos definidos mediante expresiones regulares, eliminando caracteres irrelevantes como espacios en blanco y comentarios. Este proceso actúa como puente entre el código fuente y el análisis sintáctico, facilitando la interpretación del lenguaje por parte del compilador o intérprete.

Tokens, lexemas, expresiones regulares y autómatas finitos

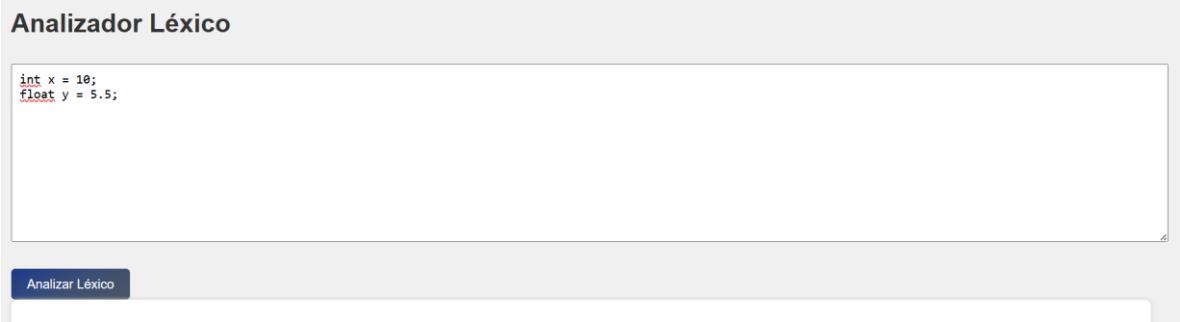
- **Tokens:** Son las unidades mínimas significativas del lenguaje que el analizador léxico reconoce. Cada token pertenece a una categoría léxica, como identificadores, palabras clave, operadores, literales, etc.
- **Lexemas:** Es la cadena de caracteres específica en el código fuente que coincide con el patrón de un token. Por ejemplo, en la línea int edad = 20;, edad es un lexema que pertenece al token *IDENTIFICADOR*.
- **Expresiones Regulares:** Son una forma formal de describir patrones léxicos. Se utilizan para definir la sintaxis de los tokens. Por ejemplo, la expresión regular [a-zA-Z_][a-zA-Z0-9_]* puede representar un identificador en muchos lenguajes.
- **Autómatas Finitos:** Son modelos matemáticos utilizados para reconocer cadenas que pertenecen a un lenguaje regular. El analizador léxico convierte las expresiones regulares en autómatas finitos deterministas (DFA) o no deterministas (NFA) para validar los lexemas en el código fuente.

Tecnologías

- HTML, CSS y JavaScript para la web interactiva.
- Node.js para el levantar servidor local.
- GitHub Pages para despliegue en línea.
- **JSHint**: herramienta para analizar la calidad del código JS.

3. Desarrollo

- Casos de prueba: solicitamos análisis léxico de las siguientes líneas de código.



- Respuesta de tokens detectados correctamente por la amplia variedad de tokens léxicos definidos en el código fuente.

Tabla de Tokens

Token	Tipo	Línea
int	PALABRA_RESERVADA	1
x	IDENTIFICADOR	1
=	OPERADOR	1
10	NÚMERO_ENTERO	1
;	DELIMITADOR	1
float	PALABRA_RESERVADA	2
y	IDENTIFICADOR	2
=	OPERADOR	2
5.5	NÚMERO_DECIMAL	2
;	DELIMITADOR	2

- ¿Tuviste errores o dificultades técnicas? ¿Cómo los solucionaste?

Si, como en todo programa, sin embargo, encontramos la forma de implementar mayor variedad de tokens para la tabla de tokens, con el fin de tener un programa más preciso y de mejor calidad.

¿Cómo implementamos el análisis léxico?

Implementé el analizador léxico en **JavaScript puro**, sin librerías externas, usando expresiones regulares para identificar tokens en el código fuente. La lógica está encapsulada en funciones, lo que permite una estructura modular y reutilizable.

Descripción del lenguaje o gramática que estás analizando

Mi analizador reconoce un lenguaje de programación similar a C o JavaScript con soporte para:

```
const palabrasReservadas = new Set(["int", "float", "if", "while", "return", "string", "void", "let", "const", "else"]);
const operadores = new Set(["+", "-", "*", "/", "=", ">", "<", "==","!","?","&&","||"]);
const delimitadores = new Set([";", "{", "}", "(", ")"]);
```

Estructura del código

Funciones: la principal es analizar Código(código), que recibe el código fuente como entrada y devuelve una lista de tokens.

Paso 1: Definimos los tokens.

```
const palabrasReservadas = new Set(["int", "float", "if", "while", "return", "string", "void", "let", "const", "else"]);
const operadores = new Set(["+", "-", "*", "/", "=", ">", "<", "==","!","?","&&","||"]);
const delimitadores = new Set([";", "{", "}", "(", ")"]);
```

Paso 2: al dar clic en el botón se obtiene el fragmento de código solicitado por el usuario y se limpian las tablas

```
//Esta funcion se ejecuta cuando el usuario haga click en analisis lexico
function analizarLexico() {
    const codigo = document.getElementById("codigo").value; //Obtenemos la informacion ingresada por el usuario
    const { listaTokens, tablaSimbolos } = analizarCodigo(codigo);// Usamos la funcion analizar codigo

    // Limpia la tabla de tokens
    const tablaTokensBody = document.getElementById("tablaTokens").querySelector("tbody");
    tablaTokensBody.innerHTML = "";
    listaTokens.forEach(t => {
        //Mostrar tabla de tokens en el html
        const fila = `<tr><td>${t.token}</td><td>${t.tipoToken}</td><td>${t.numeroLinea}</td></tr>`;
        tablaTokensBody.innerHTML += fila;
    });

    // Limpia la tablaSimbolos
    const tablaSimbolosBody = document.getElementById("tablaSimbolos").querySelector("tbody");
    tablaSimbolosBody.innerHTML = "";
    // Recorre la tabla de simbolos y muestra los identificadores
    Object.entries(tablaSimbolos).forEach(([nombre, info]) => {
        const fila = `<tr><td>${nombre}</td><td>${info.Tipo}</td><td>${info.Token}</td><td>${info.Lineas.join(", ")}</td></tr>`;
        tablaSimbolosBody.innerHTML += fila;
    });
}
```

Paso 3: Define los tokens. posteriormente se envían al html.

```
//Definir los tokens
if (palabrasReservadas.has(token)) {
    tipoToken = 'PALABRA_RESERVADA';
    if (["int", "float", "string"].includes(token)) {
        tipoActual = token;
    } else if (token === "void") {
        tipoActual = "void";
    }
} else if (operadores.has(token)) {
    tipoToken = 'OPERADOR';
    tipoActual = null;
} else if (delimitadores.has(token)) {
    tipoToken = 'DELIMITADOR';
    tipoActual = null;
} else if (/".*?"/.test(token)) {
    tipoToken = 'CADENA';
} else if (/\d+\.\d+/.test(token)) {
    tipoToken = 'NÚMERO_DECIMAL';
} else if (/\d+/.test(token)) {
    tipoToken = 'NÚMERO_ENTERO';
} else {
    tipoToken = 'IDENTIFICADOR';
    if (!tablaSimbolos[token]) {
        tablaSimbolos[token] = { Tipo: tipoActual || 'Desconocido', Token: 'Variable', Lineas: [] };
    }
    tablaSimbolos[token].Lineas.push(numeroLinea);
    if (tipoActual) {
        tablaSimbolos[token].Tipo = tipoActual;
        tipoActual = null;
    }
}
```

Flujo del programa

¿Cómo se usa?

1. El usuario ingresa código en un textarea HTML.
2. Al hacer clic en el botón "Analizar", se llama a analizarCodigo.
3. El resultado se muestra como una lista de tokens clasificados.

4. Conclusión

Con este proyecto aprendimos cómo funciona el análisis léxico y su papel fundamental dentro de un compilador. Entendimos cómo identificar y clasificar tokens a partir del código fuente, y fortalecimos nuestros conocimientos en programación.

Nuestro analizador léxico tiene algunas limitaciones, como la detección básica de errores y una cobertura limitada del lenguaje JavaScript.

Si hubiéramos contado con más tiempo o herramientas, mejoraríamos la detección de errores, ampliaríamos la variedad de tokens reconocidos y buscaríamos integrar una interfaz más amigable.

5. Autores

- María Alejandra Reina
  mariaalejandrareina122@gmail.com
  https://github.com/Mari-R122

- Jessica Tascón
  jessicamarcela12@hotmail.com
  https://github.com/JESSICATASCON

- Sebastian Ramirez Parra
  Sebasramirez830@gmail.com
  https://github.com/sebas830

Repositorio: https://github.com/sebas830/ProyectoFinalCompiladores