Instituto Tecnológico de México en Celaya

Ing. Sistemas Computacionales

Lenguajes y automatas 2

ACTIVIDAD 4

EQUIPO 4

PROF. RICARDO GONZÁLEZ GONZÁLEZ

MIEMBROS:

- Acevedo Colunga Erick Eduardo
- Plascencia Fonseca Adrian
- Quintana Villicaña Cristian
- Rivera López Rubén Eliezer

17/09/2024

Índice:

Introducción .....

Desarrollo .....

Definición de analizador léxico .....

¿Qué es un generador de analizador léxico .....

Funcionamiento y estructura de un generador .....

Principales generadores de analizador léxico .....

Conclusión .....

Referencias .....

Generadores de Analizadores Léxicos

Introducción

En el desarrollo de lenguajes de programación, los analizadores léxicos juegan un papel fundamental en el proceso de compilación o interpretación del código fuente. Un analizador lexico se encarga de la primera fase del proceso de compilación, transformando una secuencia de caracteres en una secuencia de tokens, que son unidades significativas para el lenguaje. Sin embargo, la creación manual de analizadores léxicos puede ser una tarea compleja y propensa a errores, por lo que surgieron herramientas llamadas generadores de analizadores léxicos. Estas herramientas automatizan la construcción de analizadores léxicos a partir de especificaciones de lenguajes formales, ahorrando tiempo y esfuerzo a los desarrolladores.

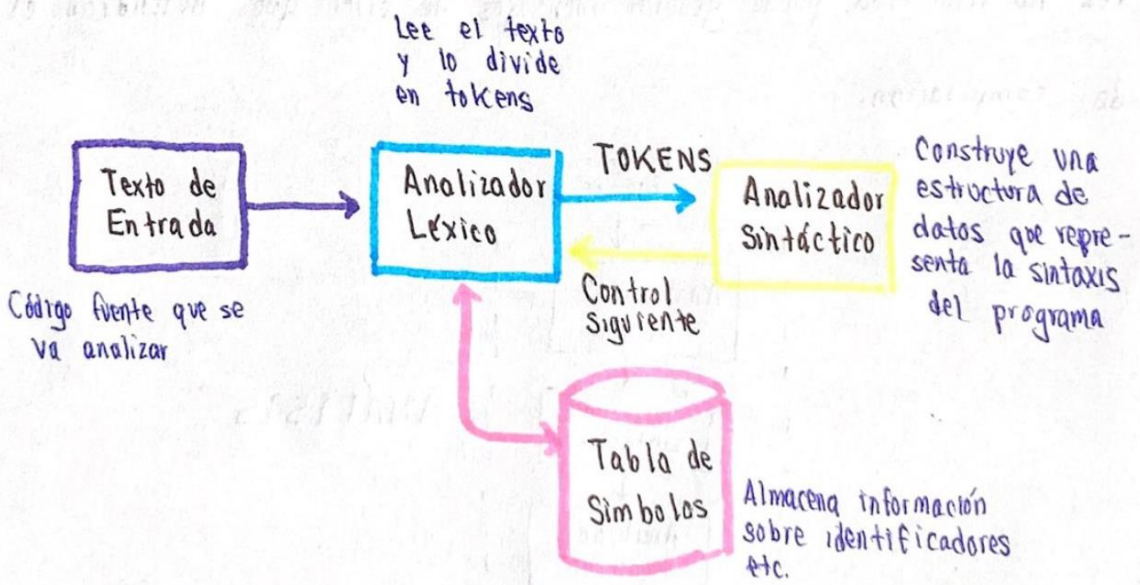
En esta monográfia técnica se abordará el concepto de los generadores de analizadores léxicos desde sus fundamentos teóricos hasta su apli-

cación práctica. Analizaremos cómo estas herramientas contribuyen al diseño y desarrollo de compiladores, las técnicas subyacentes que utilizan, como los autómatas finitos y las expresiones regulares, y cómo se integran en un proceso de compilación completo. También se hara un recorrido por las herramientas más conocidas y su impacto en el diseño de lenguajes de programación, mostrando su relevancia en el campo de la informática y su capacidad para enfrentar los desafíos de los lenguajes formales.

Desarrollo

1- Definición de analizador léxico

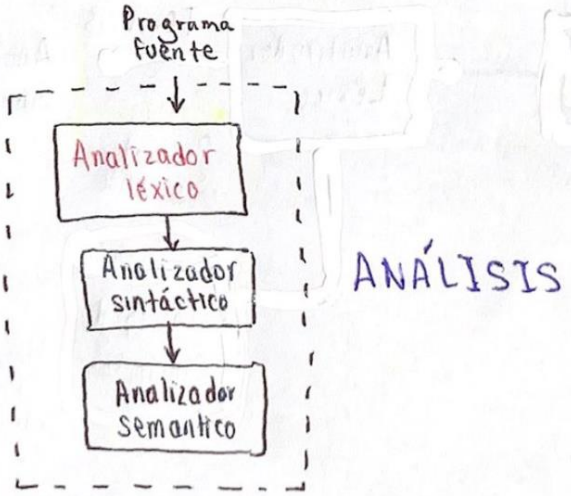
Un analizador léxico es un componente fundamental en los sistemas de procesamiento de lenguajes, especialmente en los compiladores. Su principal tarea es transformar la secuencia de caracteres que componen el código fuente en una serie de unidades mínimas de significado llamadas tokens. Los tokens representan elementos como palabras clave, operadores, identificadores, literales numéricos y otros componentes del lenguaje que serán procesados en fases posteriores, como el análisis sintáctico y semántico.



Durante este proceso, el analizador léxico también elimina elementos que no aportan información semántica, como los espacios en blanco, tabulaciones o comentarios, lo que facilita el análisis de la estructura del código.

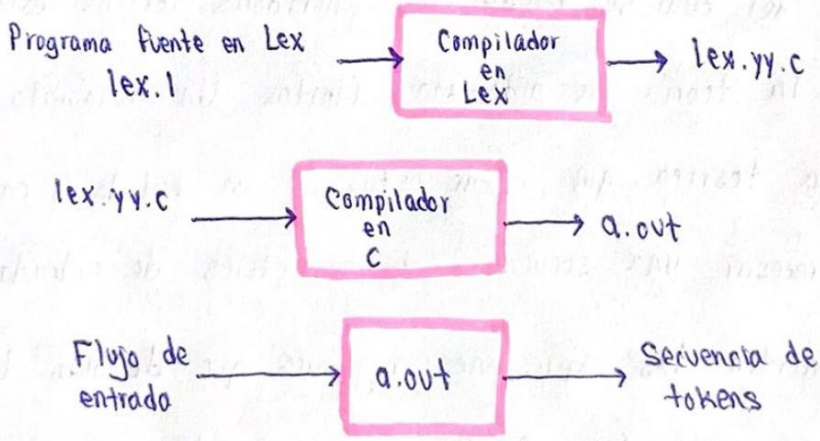
Por ejemplo, en el lenguaje de programación C, el analizador léxico puede identificar tokens como [int] (una palabra clave), [x] (un identificador), o [+] (un operador), asignando un tipo específico a cada uno de ellos

El análisis léxico, al ser la primera etapa en el procesamiento del código fuente, es esencial para garantizar que el texto de entrada cumpla con las reglas básicas del lenguaje y este listo para las fases posteriores. Cualquier error en esta etapa, como una secuencia de caracteres no reconocida, puede generar mensajes de error que detendrán el proceso de compilación.



2- ¿Qué es un generador de analizadores léxicos?

Un generador de analizadores léxicos es una herramienta que permite automatizar la creación de un analizador léxico a partir de una descripción formal del lenguaje de entrada. Los generadores simplifican el desarrollo de compiladores, intérpretes y otros sistemas de procesamiento de texto, ya que permiten a los desarrolladores especificar las reglas léxicas utilizando expresiones regulares u otras notaciones formales en lugar de escribir manualmente el código que realiza la tarea de tokenización.



"Un ejemplo de el proceso de generación y ejecución de un analizador léxico usando Lex"

El proceso para usar un generador de analizadores léxico es:

* Especificación de reglas léxicas: El desarrollador define utilizando expresiones regulares, los patrones que identifican los distintos tipos de tokens que el analizador léxico debe reconocer.

* Generación de código: A partir de las reglas léxicas, el generador produce código fuente en un lenguaje específico (como C, C++, o Java), que implementan un analizador léxico optimizado para procesar el texto de entrada.

3- Funcionamiento y estructura de los generadores de analizadores léxicos

El proceso básico del cual se basan los generadores léxicos esta fundamentado en la teoría de autómatas finitos. Un autómata finito es una máquina teórica que puede estar en un estado y cambiar de estado al procesar una secuencia de caracteres de entrada. Los generadores convierten las expresiones regulares que definen las reglas léxicas en autómatos finitos, que son utilizados para reconocer los tokens en el código fuente.

Un ejemplo: $(a|b)^* abb$

$S = \{0, 1, 2, 3\}$

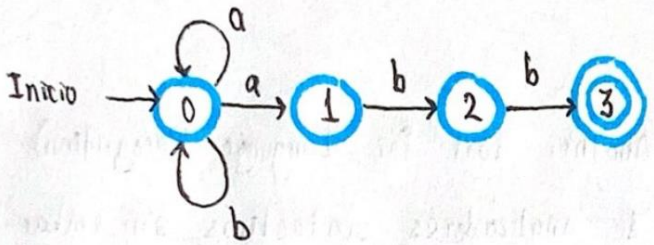
$V = \{a, b\}$

$S_0 = 0$

$F = \{3\}$

Este ejemplo nos muestra un automata finito que se especializa en reconocer la cadena $(a|b)^*abb$.

Tiene 3 estados, siendo el estado 3 el estado final.



4- Principales generadores de analizadores léxicos

A lo largo de las decadas se han desarrollado diversas herramientas para la generación de analizadores léxicos. Entre los que tenemos:

Lex y Flex:

De los primeros generadores de analizadores léxicos, desarrollado en los años 70 para sistemas Unix. Flex (Fast Lexical Analyzer) es una versión más moderna y eficiente de Lex, diseñada para mejorar el rendimiento. Ambas herramientas permiten definir reglas léxicas mediante expresiones regulares, y generar código en C. Se usan junto con Yacc y Bison.

JFlex:

Es un generador de analizadores léxico para el lenguaje de programación Java. Usa expresiones regulares para definir los tokens pero genera su código en Java.

ANTLR:

Por sus siglos en ingles ANTLR (Another Tool for Language Recognition) es conocido por ser un generador de analizadores sintácticos sin embargo tiene la capacidad de generar analizadores léxicos, permite definir reglas léxicas lo que permite la creación de compiladores e intérpretes.

Conclusión

Podemos concluir que los generadores de analizadores léxicos representan una herramienta clave en el desarrollo de compiladores, intérpretes y sistemas de procesamiento de texto. Facilitan la creación automatizada de analizadores léxicos a partir de descripciones formales del lenguaje, como expresiones regulares, reduciendo la carga de trabajo manual y minimizando errores. Gracias a los principios de la teoría de autómatas y los algoritmos de optimización, como la construcción de autómatas finitos deterministas (FDA).

Cabe resaltar las ventajas de los generadores:

* Automatización y rapidez: generación de forma automática de analizadores léxicos optimizados.

* Mantenimiento simplificado: las reglas léxicas pueden ser modificadas fácilmente sin necesidad de cambiar la lógica interna del analizador.

* Eficiencia: son muy eficientes ya que se basan en algoritmos bien establecidos y probados (determinización de autómatas finitos).

Referencias:

- Cliver Jimny Sonco Claveorias. (2020, 5 julio). analizador léxico - generador LEX [Video]. Youtube. https://www.youtube.com/watch?v=jBON-dLKRe4

- Prezi, G.L.O. (2019). GENERADOR DE ANALIZADORES LÉXICOS. prezi.com. https://prezi.com/mghu3fnrdwd7/generador-de-analizadores-lexicos

- Yahir Snchez DjMix. (2023, 7 octubre). Actividad 4.4 y 4.5 Lenguajes y autómatas Edinson Yahir Sánchez. Velazquez. pptx. [Diapositivas]. Slide Share. https://es.slideshare.net/slideshow/actividad-44-y-45-lenguajes-y-automatas-edinson-yahir-snchez-velazquezpptx/262805252

- Juan José Tinajero Vivanco (2023, 7 mayo). Generadores de analizadores léxicos [Video]. Youtube. https://www.youtube.com/watch?v=40728tvQ5gK

- Manuel Castaneda (2020, 16 octubre). Compiladores 141020, generador de analizadore léxicos [Video]. YouTube. https://www.youtube.com/watch?v=tiLwSqVfYbA

Aplicaciones (caso de estudio).

Contenido:

1. Introducción
2. Fundamentos Teóricos
3. Desarrollo del tema:
   3.1 Lenguajes formales: Fundamentos y clasificación.
   3.2 Generadores de analizadores léxicos
   3.3 Caso de estudio: Uso de Flex en la construcción de un compilador.
4. Conclusión

Bibliografía

1 Introducción:

La teoría de lenguajes formales y autómatas es uno de esos temas que, aunque parezca complejo, es fundamental para entender cómo funcionan las computadoras y los sistemas que usamos a diario. Lo fascinante de esta teoría es que nos permite describir, con reglas muy precisas, cómo funcionan los lenguajes de programación, los protocolos de comunicación, y muchas otras estructuras informáticas. En otras palabras, a través de los lenguajes formales podemos entender el "idioma" que hablan las computadoras, y los autómatas son como modelos abstractos que nos muestran cómo estas máquinas procesan esas "instrucciones" para ejecutar cualquier tarea.

Hoy en día, la teoría de lenguajes formales y autómatas sigue siendo esencial en la ingeniería de software, en la inteligencia artificial y en muchos otros campos. Por ejemplo, cuando se compila un programa, lo que realmente está ocurriendo es que el código fuente se está descomponiendo en sus elementos más básicos, que luego son procesos según reglas gramaticales. Este proceso depende mucho de los lenguajes regulares y los lenguajes libres de contexto, y aquí es donde entran en juego los autómatas finitos y los autómatas. Por otro lado, las máquinas de Turing nos muestran hasta dónde puede llegar la capacidad de cómputo de una máquina, algo que sigue siendo un tema de estudio profundo en informática teórica.

En esta monografía, exploraremos cómo los generadores de analizadores léxicos, como Lex y Flex, aprovechan los lenguajes regulares y los autómatas finitos para procesar texto y generar tokens, esenciales para la traducción de lenguajes de programación a código máquina, directo en las tecnologías que usamos a diario. Cada vez que escribimos código, usamos aplicaciones o interactuamos con sistemas inteligentes, hay una estructura matemática detrás que nos permite hacerlo de manera eficiente y segura.

2. Fundamentos Teóricos:

Antes de profundizar en las aplicaciones, es importante repasar algunos conceptos clave:

- Lenguajes formales: Un lenguaje formal es un conjunto de cadenas de símbolos que son válidos de acuerdo con una gramática formal. Los lenguajes formales se clasifican en diferentes tipos según la jerarquía de Chomsky: lenguajes regulares, libres de contexto, sensibles al contexto y recursivamente enumerables.

- Autómatas: Los autómatas son máquinas abstractas que reconocen lenguajes. Entre los más comunes los autómatas finitos, que reconocen lenguajes regulares y los autómatas de pila, que reconocen lenguajes libres de contexto. También existen autómatas más complejos, como las máquinas de Turing, que son un modelo teórico para la computación general.

3. Desarrollo del tema:

3.1 Lenguajes formales: Fundamentos y clasificación

Los lenguajes formales son esenciales para describir y definir estructuras simbólicas de manera precisa y rigurosa. Son la base de los sistemas computacionales, algoritmos y la lógica que gobierna el comportamiento de las máquinas. Para entender mejor las capacidades y limitaciones de estos lenguajes, Noam Chomsky propuso una clasificación, conocida como la jerarquía de Chomsky, que divide los lenguajes en cuatro categorías. Esta clasificación nos ayuda a ver qué tipos de lenguajes pueden ser reconocidos y procesados por diferentes modelos computacionales, como los autómatas.

Lenguajes regulares y análisis léxico:

Los lenguajes regulares son los más simples de esta jerarquía. Son aquellos que pueden ser reconocidos por un autómata finito, un tipo de máquina abstracta con un número limitado de estados. Los lenguajes regulares se describen a menudo con expresiones regulares, que son una manera de representar patrones comunes en textos.

Un ejemplo muy práctico de un lenguaje regular lo encontramos en los motores de búsqueda de patrones en texto, como grep en sistemas unix. Esto es como cuando se tiene un archivo de texto enorme y se necesita encontrar todas las líneas que contienen la palabra "error". Una expresión regular podría ser algo tan sencillo como error, que instruye

a grep para que encuentre todas las coincidencias exactas.
Un analizador léxico utiliza estas expresiones regulares para escanear el código fuente y descomponerlo en unidades significativas llamadas tokens.

Un token puede ser una palabra clave como if, else, while, un identificador como el nombre de una variable, o un símbolo como un operador + o =. Cada uno de estos elementos se define con una expresión regular que describe su patrón. El analizador léxico explora el código fuente y, usando estas reglas, reconoce cada token en función de su forma.

Ejemplo práctico: Esto es como tener un código fuente simple en un lenguaje de programación como C, que contiene la palabra clave if. Un analizador léxico debe reconocer esta palabra clave y diferenciarla de otras cadenas de texto. Una expresión regular que capture este comportamiento sería algo como:

if|else|for|while

Esta expresión regular instruye al analizador léxico a reconocer cualquiera de estas palabras clave como tokens específicos. Así, cuando encuentra if en el código, lo identifica como una estructura de control.

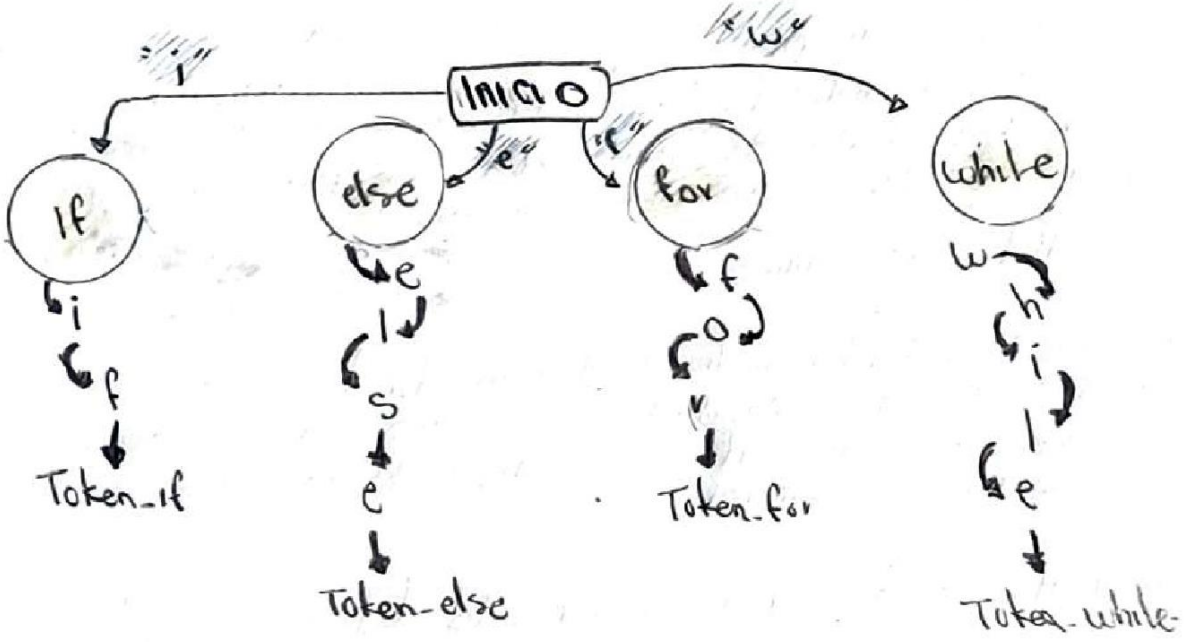
Autómatas finitos y el reconocimiento de lenguajes regulares:

son modelos matemáticos usados para crear analizadores léxicos. Un DFA procesa una cadena de texto y decide si cumple con las reglas de un lenguaje regular. En análisis léxico, esto significa que el DFA lee el código fuente carácter por carácter y determina si cumple con los patrones establecidos.

Cuando el DFA encuentra un grupo de caracteres que coinciden con uno de esos patrones, genera un token. Este proceso es muy rápido porque el DFA sigue un flujo secuencial sin necesidad de retroceder para hacer más verificaciones.

Ejemplo práctico:

Si un DFA está programado para reconocer las palabras clave if, else, for, while, comenzará en un estado inicial. Si lee una letra "i", avanzará a un estado que espera una "f" (para la palabra clave "if"). Si lo encuentra, el DFA genera un token para esa palabra clave.



3.2. Generadores de analizadores léxicos.

Los generadores de analizadores léxicos, como Lex y Flex, son herramientas automatizadas que simplifican el proceso de crear un analizador léxico. En lugar de construir manualmente un analizador léxico desde cero, los desarrolladores pueden usar estas herramientas para definir las reglas léxicas en forma de expresiones regulares, y el generador se encarga de traducir estas expresiones a código fuente que implementa el DFA.

Lex/Flex:

Lex y Flex son dos de los generadores de analizadores léxicos más utilizados. Ambos permiten a los programadores definir las reglas para los tokens mediante expresiones regulares y luego generan automáticamente el código C correspondiente para un analizador léxico eficiente. Este código fuente y reconoce los tokens especificados.

Ejemplo práctico:

Este ejemplo es acerca de un analizador léxico que reconozca identificadores, números enteros y operadores aritméticos.

ejemplo1.l

```
%{
#include <stdio.h>
%}

%%

[0-9]+        { printf("Número: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identificador: %s\n", yytext); }
"+"           { printf("Operador más: +\n"); }
"-"           { printf("Operador menos: -\n"); }
[ \t\n]       ;  /* Ignorar espacios, tabulaciones y saltos de línea */

%%

int main() {
    yylex();  // Función generada por Flex para comenzar el análisis léxico
    return 0;
```

Archivo generado en el analizador léxico Flex

Aquí, las expresiones regulares definen lo siguiente:

- [a-zA-Z][a-zA-Z0-9]*: Identifica identificadores, es decir, secuencias que comienzan con una letra o subrayado (_) y pueden seguir con letras o números.
- [0-9]+: Reconoce números enteros.
- "+"|"-"|"*"|"/": Reconoce operadores aritméticos.

Flex generará automáticamente un analizador léxico en C que podrá procesar cadenas de código fuente y generar los tokens correspondientes.

3.3 Caso de estudio: Uso de flex en la construcción de un compilador.

Para ejemplificar el uso práctico de Flex, se desarrollo un caso de de estudio la construcción de un analizador léxico para un lenguaje de programación simple utilizando esta herramienta.

Definición de reglas léxicas:

La primera etapa en la construcción de un analizador léxico con flex es definir las reglas léxicas utilizando expresiones regulares. Estas reglas nos permitirán reconocer los diferentes componentes de un lenguaje, como identificadores, palabras clave, operadores y números.

- identificadores: [a-zA-Z][a-zA-Z0-9]*
- números enteros: [0-9]+
- Operadores: \+|\-|*|/

Generación de tokens:

Flex toma estas reglas léxicas y genera un DFA que procesa el código fuente. Cada vez que el DFA reconoce una secuencia que coincide con una de las expresiones regulares, genera un token que representa el componente del lenguaje correspondiente.

Por ejemplo, si el código fuente contiene la expresión a = 5+b;, Flex genera los siguientes tokens:

- a → IDENTIFICADOR
- = → OPERADOR
- 5 → ENTERO
- + → OPERADOR
- b → IDENTIFICADOR
- ; → TOKEN DE FIN DE SENTENCIA

Estos tokens se enviarán a la siguiente fase del compilador, el análisis sintáctico, donde se verificará si las secuencias de tokens siguen las reglas gramaticales del lenguaje.

Este proceso muestra cómo Flex automatiza el trabajo de convertir el código fuente en tokens que pueden ser interpretados por un compilador, lo que ahorra una enorme cantidad de trabajo manual y reduce el riesgo de errores.

1: Conclusión

En resumen, la teoría de lenguajes formales y autómatas no es solo una parte fundamental de la informática teórica, sino que tiene aplicaciones prácticas clave, como la construcción de analizadores léxicos. Herramientas como Lex y Flex nos permiten aprovechar los lenguajes regulares y los autómatas finitos para generar analizadores léxicos de manera automática, lo que facilita el desarrollo de compiladores. Estas herramientas simplifican lo que de otro modo sería una tarea muy compleja y propensa a errores, permitiendo que los programadores definan las reglas léxicas de un lenguaje de programación con expresiones regulares. El proceso de descomponer un código fuente en tokens, a través del uso de autómatas finitos y generadores de analizadores, es crucial para convertir lenguajes de alto nivel en algo que las máquinas puedan procesar. A través de este estudio, queda claro que los conceptos teóricos que a menudo parecen abstractos, como los autómatas y los lenguajes formales, tienen un impacto muy tangible y útil en la creación de software que usamos todos los días.

Bibliografía:

- Kamila. (2022, 26 octubre). Lex (generador de analizadores léxicos). TechEdu. https://techlib.net/techedu/lex-generador-de-analizadores-lexicos/

- Organización de lenguajes y compiladores I. 1 Automatización y Optimización de la generación de analizadores léxicos en la industria. (2023, noviembre). ResearchGate. https://www.researchgate.net/publication/375160887_AUTOMATIZACION_Y_OPTIMIZACION_DE_LA_GENERACION_DE_ANALIZADORES_LEXICOS_EN_LA_INDUSTRIA

- The open Group (n.d). Lex - A Lexical Analyzer Generator. Retrieved September 16, 2024 from https://pubs.opengroup.org/onlinepubs/9699919799/utilities/lex.html.

- IBM (n.d.) Lex and Yacc. Retrieved September 16 2024, from https://www.ibm.com/docs/en/zos/2.3.0?topic?=1-tools-lex-yacc

PRÁCTICA DE LABORATORIO

CARRERA	NOMBRE DE LA ASIGNATURA
I.S.C (Ingeniería En Sistemas Computacionales)	LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO
ISC. RICARDO GONZÁLEZ GONZÁLEZ

PRACTICA No.	NOMBRE DE LA PRACTICA	DURACIÓN (HORAS)
1	ANALIZADOR LÉXICO FLEX (al menos tres definiciones léxicas propias del lenguaje programación c.)	3

1 INTRODUCCIÓN

El análisis léxico es una parte importante del compilador porque se encarga de convertir el código fuente en un conjunto de símbolos para ser procesados en etapas posteriores. Flex es un software gratuito que facilita la creación de analizadores léxicos que utilizan expresiones regulares para identificar patrones específicos en el código. En esta práctica de laboratorio, aprenderemos cómo usar Flex para crear un analizador que detecte elementos del lenguaje de programación C. El conocimiento de las herramientas es esencial para el desarrollo de software, lo que implica crear o modificar compiladores. También está diseñado para familiarizar a los estudiantes con los conceptos básicos del análisis léxico y su importancia en el ciclo de vida del compilador.

2 OBJETIVO (COMPETENCIAS)

El objetivo principal de esta práctica es dotar al estudiante de competencias prácticas en el uso del **analizador léxico Flex**, permitiendo que sean capaces de:

- Comprender el papel del análisis léxico en el proceso de compilación.
- Implementar un **analizador léxico** que identifique al menos tres definiciones léxicas propias del lenguaje C, tales como palabras clave, operadores y números.
- Aplicar expresiones regulares para definir patrones léxicos y generar un analizador funcional.
- Desarrollar habilidades para usar herramientas de software libre en la creación de componentes clave de un compilador.

3	MARCO TEÓRICO REFERENCIAL

- ✓ El proceso de compilación tiene varias etapas, y el análisis léxico es una de las primeras y más importantes. En esta fase, el código fuente que escribe el programador se divide en tokens, que son las partes más pequeñas que el compilador puede entender, como palabras clave (ejemplo: if, while), operadores (como +, -, *) y números (como 123 o 45.67).
- ✓ Flex es una herramienta que genera automáticamente analizadores léxicos a partir de reglas que el usuario define usando expresiones regulares. Estas reglas permiten detectar patrones en el código fuente, y Flex luego genera un programa en C que puede leer el código y devolver los tokens encontrados.
- ✓ En el caso del lenguaje C, su sintaxis tiene palabras clave, operadores y estructuras que el compilador debe identificar correctamente. Esta práctica mostrará cómo usar Flex para detectar palabras clave, operadores y números enteros en un código escrito en C.

4	MATERIALES UTILIZADO

- Computadora (i5 13th gen, 16 GB de RAM, 1 TB SSD)

5	REQUISITOS BÁSICOS.

Las herramientas utilizadas para realizar la práctica fueron las siguientes
Software:
- Sistema Operativo: Debian 11 o superior
- Visual Studio Code

6	DESARROLLO DE LA PRÁCTICA (PASO A PASO)

1. Instalación de Flex:

1.1- Instalación del software el analizador léxico Flex:

En la mayoría de las distribuciones de Linux, Flex está disponible en los repositorios oficiales, por lo que la instalación es bastante sencilla utilizando el gestor de paquetes de la distribución.

En Ubuntu/Debian:

1. Abre una terminal.
2. Ejecuta el siguiente comando:

```
sudo apt-get update
sudo apt-get install flex
```

1.2 Verificar la Instalación

Después de instalar Flex, se puede verificar que se haya instalado correctamente ejecutando el siguiente comando en la terminal:

```
flex --version
```

En mi caso se tiene la versión 2.6.4

```
crisscross@DESKTOP-3DM810J:~$ flex --version
flex 2.6.4
```

2. Definición del Problema

En este ejercicio, vamos a construir un analizador léxico que reconozca tres definiciones léxicas comunes en el lenguaje C:

1. **Palabras clave**: if, else, return.
2. **Operadores**: +, -, *, /.
3. **Números enteros**: cualquier secuencia de dígitos.

2.1. Configuración del Entorno

- Crear una carpeta para la gestión de archivos la cual se llamará “flex” con el siguiente comando: `mkdir flex`
- Dentro de la carpeta crear un archivo llamado analizador_c.l en el cual se definirán las reglas léxicas que Flex usará.

```
crisscross@DESKTOP-3DM810J:~/flex$ touch analizador_c.l
```

2.2. Definición de las Reglas Léxicas en Flex

En el archivo analizador_c.l, se definen las reglas para identificar palabras clave, operadores y números enteros. En el siguiente código ya están definidas:

```
%{
#include <stdio.h>
%}

%%

"if"          { printf("Palabra clave: if\n"); }
"else"        { printf("Palabra clave: else\n"); }
"return"      { printf("Palabra clave: return\n"); }

"+"           { printf("Operador: +\n"); }
"-"           { printf("Operador: -\n"); }
"*"           { printf("Operador: *\n"); }
"/"           { printf("Operador: /\n"); }

[0-9]+        { printf("Número entero: %s\n", yytext); }

"("           { printf("Paréntesis izquierdo: (\n"); }
")"           { printf("Paréntesis derecho: )\n"); }
";"           { printf("Punto y coma: ;\n"); }

[ \t\n]       ;  /* Ignorar espacios y tabulaciones */

.             { printf("Símbolo no reconocido: %s\n", yytext); }

%%

int main() {
    yylex();  // Llamada a la función que Flex genera para el análisis léxico
    return 0;
}
```

3.Explicación del código:

3.1 Sección de definiciones y el bloque %{ %}

```
%{
#include <stdio.h>
%}
```

Este bloque **{% ... %}** es un bloque de código en C que Flex copiará directamente en el archivo C generado **(lex.yy.c).** Todo lo que se ponga aquí se incluirá en el archivo generado sin modificaciones.

3.2 Reglas léxicas (%% ... %%)

En esta sección, se definen las expresiones regulares que Flex utilizará para reconocer patrones de texto (tokens) en el código fuente. Cada regla tiene la siguiente estructura:

expresión_regular { acciones }

- *expresión_regular:* Es un patrón que Flex tratará de coincidir en el texto de entrada. Cuando el patrón coincida, Flex ejecutará la acción asociada.
- *acciones:* Son bloques de código en C que se ejecutan cuando la expresión regular coincide con una parte del texto de entrada.

3.3 Palabras clave:

- **Palabras clave**: Se definen las palabras clave if, else, y return como tokens reconocibles. Cuando se encuentra alguna de estas palabras, se imprime un mensaje indicando que es una palabra clave.

```
"if"        { printf("Palabra clave: if\n"); }
"else"      { printf("Palabra clave: else\n"); }
"return"    { printf("Palabra clave: return\n"); }
```

- **Operadores**: Se definen los operadores +, -, *, /. Cuando se detecta alguno de ellos, se imprime el operador correspondiente.

```
"+"         { printf("Operador: +\n"); }
"-"         { printf("Operador: -\n"); }
"*"         { printf("Operador: *\n"); }
"/"         { printf("Operador: /\n"); }
```

- **Números enteros**: Se define una expresión regular [0-9]+ para reconocer cualquier secuencia de uno o más dígitos, que será tratada como un número entero.

```
[0-9]+      { printf("Número entero: %s\n", yytext); }
```

yytext es una variable global proporcionada automáticamente por Flex. Esta variable contiene el texto que coincide con la última expresión regular que fue reconocida.
Por ejemplo, si Flex reconoce un número como 123, el valor de yytext será "123", y al usar printf("Número entero: %s\n", yytext);, se imprime el texto exacto que fue encontrado

Paréntesis y punto y coma: para reconocer los paréntesis (y), ; con el mensaje correspondiente.

```
"("         { printf("Paréntesis izquierdo: (\n"); }
")"         { printf("Paréntesis derecho: )\n"); }
";"         { printf("Punto y coma: ;\n"); }
```

3.4 Ignorar espacios, tabulaciones y saltos de líneas

```
[ \t\n]     ;  /* Ignorar espacios y tabulaciones */
```

- Esta expresión regular reconoce espacios en blanco ( ), tabulaciones (\t) y saltos de línea (\n).
- La acción asociada es simplemente un ;, lo que significa que cuando Flex encuentra alguno de estos caracteres, no hará nada (se ignoran los espacios en blanco).

3.5 Símbolos no reconocidos:

```
.           { printf("Símbolo no reconocido: %s\n", yytext); }
```

- El punto (.) en una expresión regular coincide con cualquier carácter (excepto el salto de línea).
- Esta regla se activa cuando Flex encuentra un carácter que no coincide con ninguna de las reglas anteriores. En este caso, imprimirá "Símbolo no reconocido" seguido del carácter encontrado (almacenado en yytext).

3.6 Sección del código en C para el programa principal:

```
int main() {
    yylex();  // Llamada a la función que Flex genera para el análisis léxico
    return 0;
}
```

- Está la función principal del programa en C (main). En este caso, el único propósito de main es llamar a la función yylex().
- yylex() es la función generada por Flex, que se encarga de leer la entrada de texto, aplicar las reglas léxicas que definiste, y ejecutar las acciones correspondientes.
- Cada vez que yylex() encuentra un token que coincida con una de tus reglas, ejecuta la acción especificada (por ejemplo, imprimir el tipo de token).

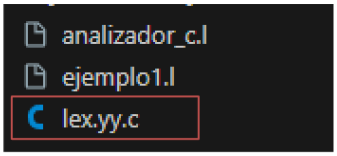
4.- Generación del Analizador Léxico

1. En la terminal de Debian/Ubuntu, posicionado en la carpeta de “flex”que es donde se creó el archivo analizador_c.l.
2. Ejecutar el siguiente comando para que Flex genere el código en C basado en las reglas:

```
flex analizador_c.l
```

```
crisscross@DESKTOP-3DM810J:~/flex$ flex analizador_c.l
```

Esto generará un archivo llamado lex.yy.c, que contiene el código C para el analizador léxico.



5. Compilación y Ejecución del Analizador Léxico

1.- Compilar el archivo generado por Flex con el siguiente comando:

```
gcc lex.yy.c -o analizador_c -lfl
```

2.Una vez compilado, ejecutar el analizador con el comando:

```
./analizador_c
```

3. Se puede probar ingresando líneas de código simples en C como:

```
crisscross@DESKTOP-3DM810J:~/flex$ ./analizador_c
if (x == 1) return 5 + 3;
```

4. El analizador léxico debería generar una salida similar a:

```
Palabra clave: if
Paréntesis izquierdo: (
Símbolo no reconocido: x
Símbolo no reconocido: =
Símbolo no reconocido: =
Número entero: 1
Paréntesis derecho: )
Palabra clave: return
Número entero: 5
Operador: +
Número entero: 3
Punto y coma: ;
```

Palabra clave: if

- El analizador léxico ha encontrado la palabra clave **if** en el código. Esto ocurre porque se tiene una regla en el archivo Flex que reconoce las palabras clave específicas (if, else, return). En este caso, detecta if y lo clasifica correctamente.

Paréntesis izquierdo: (

- El analizador encuentra el símbolo (y lo reconoce como un paréntesis izquierdo. Esto es

porque se definio una regla específica en Flex para manejar los paréntesis ((y)), y aquí se está aplicando dicha regla.

Símbolo no reconocido: x

- El analizador encuentra la variable x, pero en el archivo Flex no se tiene una regla que permita reconocer identificadores (nombres de variables o funciones). Por eso, el analizador lo clasifica como un "símbolo no reconocido".

Símbolo no reconocido: =

- Lo mismo que en el caso anterior: el analizador encuentra otro símbolo =, pero no lo reconoce porque no se tiene una regla específica para operadores de comparación.

Número entero: 1

- El analizador léxico encuentra el número 1 y lo clasifica correctamente como un número entero. Esto ocurre porque se tiene una regla en el archivo Flex que reconoce secuencias de dígitos ([0-9]+) como números enteros.

Paréntesis derecho:)

- Aquí, el analizador detecta el símbolo) y lo clasifica correctamente como un paréntesis derecho, gracias a la regla que se definió para los paréntesis.

Palabra clave: return

- El analizador encuentra la palabra clave return y la clasifica correctamente como una palabra clave porque se tiene una regla para reconocer esta palabra en particular.

Punto y coma: ;

- Finalmente, el analizador encuentra el símbolo; y lo clasifica correctamente como un punto y coma, gracias a la regla específica que se añadió para reconocer este símbolo.

En este ejercicio, hemos creado un analizador léxico básico que reconoce tres componentes clave del lenguaje C: palabras clave, operadores y números enteros. Este tipo de herramientas es fundamental en el proceso de compilación, ya que el análisis léxico es el primer paso para transformar el código fuente en una estructura que puede ser procesada y optimizada por el compilador. Flex ha demostrado ser una herramienta poderosa y eficiente para generar analizadores léxicos a partir de expresiones regulares, facilitando la creación de componentes que pueden ser usados en proyectos más complejos, como compiladores completos o herramientas de análisis de código.

5 BITÁCORA DE INCIDENCIAS

PROBLEMA	FECHA	HORA	SOLUCIÓN	FECHA	HORA

6 OBSERVACIONES

7 ANEXOS

El documento generado:
https://drive.google.com/file/d/1r-_rm5mS2hkTW2-IDryXOENVioR2FuHH/view?usp=drive_link

7 REFERENCIAS BIBLIOGRÁFICAS

Lexical Analysis With Flex, for Flex 2.6.2: Concept Index. (s. f.). https://westes.github.io/flex/manual/Concept-Index.html#Concept-Index

GeeksforGeeks. (2023, 10 abril). *FLEX (Fast Lexical Analyzer Generator)*. GeeksforGeeks. https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/