

Manual Técnico: Analizador Léxico

Instituto Tecnológico de Tepic

Ingeniería en Sistemas Computacionales

Lenguajes y Autómatas I

Integrantes:

Diego Arturo Cadena Flores 18401089

Darien Rafael Márquez Vázquez 17401326

Luis Sergio Carrillo Sanchez 18401095

Daniel Alejandro Calderon Virgen 18401090

Docente: M.D.O.H. Sonia Alvarado Mares

Contenido

[Introducción. 4](#_Toc72621338)

[Herramientas 4](#_Toc72621339)

[JFlex 4](#_Toc72621340)

[Expresiones Regulares 5](#_Toc72621341)

[Definición de reglas en JFlex 7](#_Toc72621342)

[Tablas de Símbolos 9](#_Toc72621343)

[Generación de Errores 10](#_Toc72621344)

[Conclusión 10](#_Toc72621345)

[Bibliografía: 11](#_Toc72621346)

# Introducción.

Dentro de las fases de un compilador, la primera de ellas es el analizador léxico, que se encarga de analizar caracter por caracter de las cadenas que se ingresaron, y determinar si son o no son válidas las cadenas, así como también con ayuda del analizador sintáctico de ir generando la tabla de símbolos, que se utiliza en todas las fases del compilador.

El siguiente documento pretende ofrecer al lector los conocimientos necesarios para la operación del analizador léxico del lenguaje de programación a fin de poder agilizar su uso y brindar un entendimiento integral del mismo. Como vimos a través de la unidad definir distintos lenguajes, alfabetos y expresiones regulares nos son de gran utilidad para poder crear nuestro analizador léxico, en el contenido de este manual se explicará cómo se aplicaron diferentes herramientas y conocimientos referentes a la construcción de un analizador léxico, siendo estas, por ejemplo, tablas de símbolos, generación y manejo de errores.

# Herramientas

Con el objetivo de desarrollar el analizador léxico se hizo uso de la herramienta JFlex, la cual es un escáner que examina carácter por carácter dentro de un conjunto de cadenas. Pero, ¿cómo funciona JFlex? Como se mencionó anteriormente, JFlex analiza una cadena carácter por carácter para reconocer lexemas, y la forma en que lo hace es mediante expresiones regulares que tienen un formato similar a las que se usan en la teoría de lenguajes formales.

## JFlex

JFlex es una herramienta escrita en Java que es utilizada para generar lexers en un programa. Esta herramienta brinda acceso a varias “técnicas” y sintaxis para poder especificar expresiones y definiciones regulares dentro del programa. Para poder implementar JFlex en el proyecto, se define un archivo que contiene las expresiones y definiciones regulares en base a un alfabeto definido en el diseño del lenguaje. Al completar las definiciones regulares de varios componentes como números, espacios en blanco, etc, se puede definir las acciones que debe de tomar el programa al detectar esos componentes, como ignorar a los comentarios y a los espacios en blanco, como el tabulador y el espacio. En el caso de componentes como identificadores y palabras reservadas, se especifica un valor de retorno, el cual nos sirve como parte del token, ya que la clasificación define el tipo de cadena que es.

## Expresiones Regulares

Para poder identificar si un lexema es válido o no, lo cual es el objetivo de este programa, fue mediante la teoría y métodos de las expresiones regulares. Dichas expresiones fueron hechas con base a las características del lenguaje de programación *LYADRONE*, características previamente definidas en la documentación del diseño del lenguaje.

Lo primero que se hizo fue definir grupos para después construir las expresiones regulares de una manera más sencilla y legible. Dentro de las principales características están que solo se permiten letras con minúscula y sin incluir la ‘*ñ*’, para definir este grupo lo se hace de la siguiente manera:

L = [a-z]

de esta forma cada vez que se utilice *L* en alguna expresión regular, se estará refiriendo a cualquier letra dentro del rango de la ‘a’ a la ‘z’.

Para el tema de los dígitos se hizo algo de forma similar,

D = [0-9]

y de igual manera cada vez que se mande a llamar *D,* se estará refiriendo a cualquier dígito del 0-9.

Con estas dos definiciones ya es suficiente para crear una expresión para los identificadores, los cuales tienen como regla que debe de empezar con un carácter seguido de 0 o 31 letras y/o dígitos y/o guiones bajos ‘\_’, la cual quedó de la siguiente manera:

iden={ L } ( { L } | { D } | \_ ){ 0 , 31 }

Con esa expresión regular cualquier lexema que cumpla con ese patrón y **no** sea palabra reservada, será un identificador.

Los números fueron definidos con una expresión regular {D}+, la cual acepta cadenas de 0 o más, pero no se estarán incluyendo los números decimales ni los números negativos. Entonces lo que se hizo fue generar un grupo de signos con el más ‘+’ y el menos ‘-’:

S = {\-,\+}

Es importante usar el carácter de escape antes de los símbolos {+,-}, de lo contrario no serán detectados.

En base a esto se pueden crear expresiones regulares para números que incluyan números negativos

Números enteros: enteros = { S }?{ num }

Números decimales: numDec = { S }? { num }?\.{ num }

Números exponente: numExp = { S }? { num }?(\.{ num })?e{ enteros }

Con esas 3 expresiones regulares cualquier número que ingrese entrará en alguna de las clasificaciones.

Para ignorar los espacios, los saltos de línea y los tabuladores, también se definió un grupo con dichos caracteres de escape

ESCAPE = [ \t\r\n]

Una de las desventaja de una jFlex es que puede que se escapen algunas cadenas de las expresiones, por ejemplo una de las cadenas que no debe de reconocer el analizador en “aaB12”, esta cadena contiene una “B” que no está dentro del lenguaje, sin embargo, el escáner separa la cadena en 3 partes: “aa”, “B y “12”. Dando como resultado un identificador, un número y un error léxico. Para poder evitar esto, se definió un grupo que incluía las letras de la a-z en mayúsculas y minúsculas, y se declaró una expresión de cadenas inválidas.

LE = [A-Za-z]

invalido = ( { LE } | { D } )\*

De esta manera cualquier cadena que contenga una letra mayúscula la detectara como una cadena invalida.

## Definición de reglas en JFlex

Una vez hechas las expresiones regulares, solo queda definir las reglas de los patrones, para el tema de las palabras reservadas, no se usan expresiones, sino que se declaran así tal cual, al igual que algunos símbolos como los operadores aritméticos, lógicos y relacionales.

Es muy importante definir dichas reglas con un orden de prioridad, por lo cual las palabras reservadas tienen que ir antes que la regla de los identificadores, de otro modo todas las palabras reservadas serán tomadas en cuenta como identificadores.

La forma en que se definieron las reglas fue de siguiente manera

<YYINITIAL> (lexema o expresion regular) {

c.linea=yyline; //se guarda el número de línea

lexema=yytext(); //se guarda el léxema

return componenteLéxico;

}

Ejemplo de una regla para el operador más +

<YYINITIAL> "+" {

c.linea=yyline;

lexema=yytext();

return OPERADOR\_MAS;

}

Ejemplo de regla para la palabra reservada *iniciar*

<YYINITIAL> "iniciar" {

c.linea=yyline;

lexema=yytext();

return RESERVADA\_INICIAR;

}

Ejemplo de regla para identificadores, aquí ya se usó una expresión regular.

<YYINITIAL> {iden} {

c.linea=yyline;

lexema=yytext();

return IDENTIFICADOR;

}

Ejemplo de regla para números decimales

<YYINITIAL>{numDec} {

c.linea=yyline;

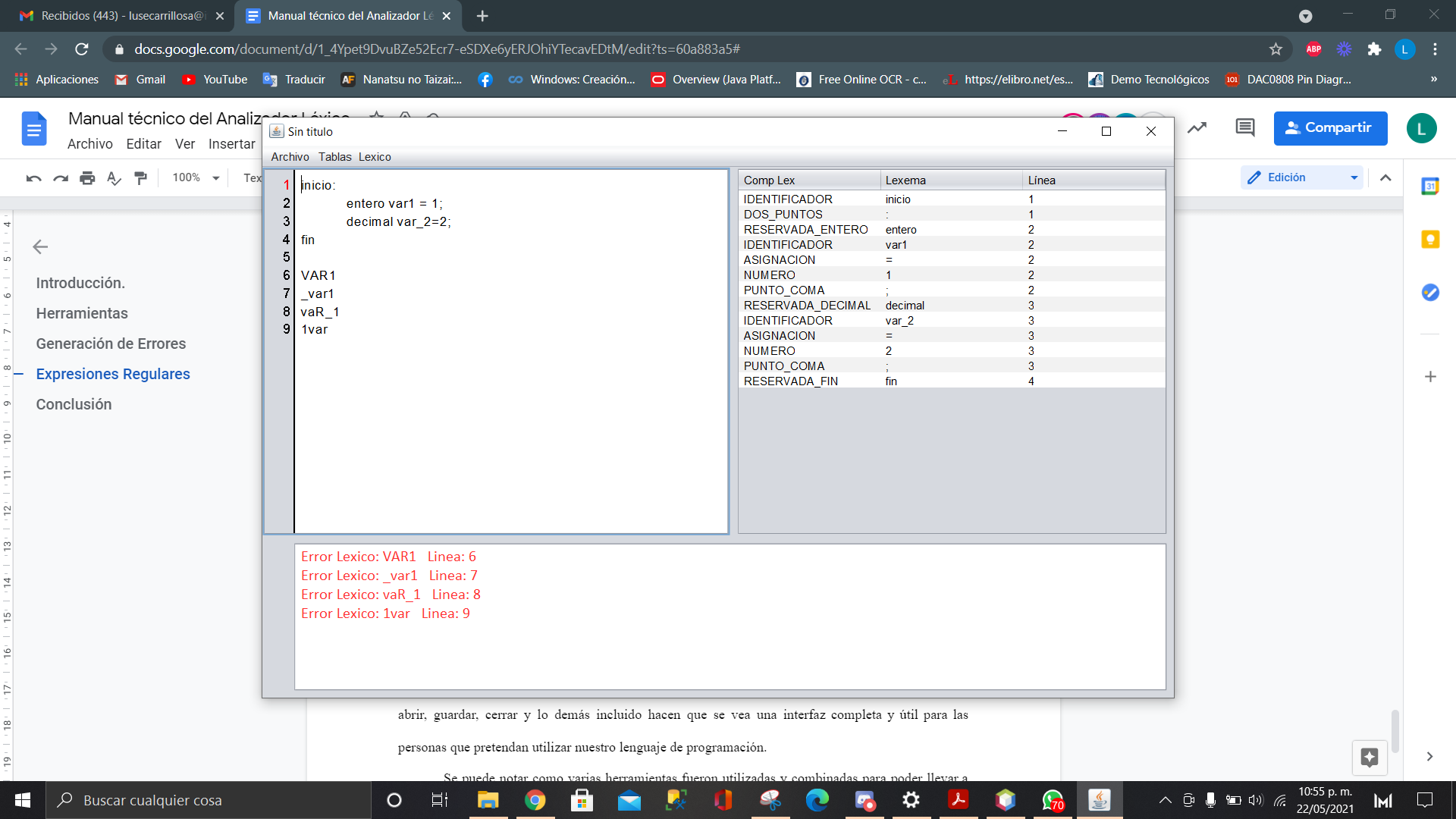
lexema=yytext();

return NUMERO\_DECIMAL;

}

Una vez definidas las reglas, se ejecuta un método de la librería JFlex que genera los autómatas con las reglas que se declararon y las guarda en un archivo Java. Con ese archivo se evalúan las cadenas que en este caso serán leídas desde un área de texto.

Ejemplo final:

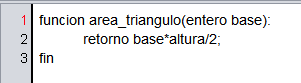


## Tablas de Símbolos

La tabla de símbolos fue generada a partir de los datos generados y retornados por la librería JFlex, donde tomamos la información de cada todas las cadenas analizadas, tomando su componente léxico, el lexema y la línea en la que se encuentra, a diferencia de la tabla mostrada en la ventan principal del editor, cuando se manda a llamar a la tabla de símbolos, esta no agrega a los identificadores dos veces si este es el caso.

Para esto utilizamos una estructura de datos que se encarga de buscar estos identificadores para saber si ya están previamente escritos.

Ejemplo de generación de tabla de símbolos a partir del siguiente código de *LYADRONE*:



Generando la tabla de símbolos:



## Generación de Errores

Para el manejo de errores, tomando en cuenta el diseño del lenguaje, se definieron en grupos las condiciones que producirían un error sintáctico , cualquier cadena que sea identificada como perteneciente a este grupo será catalogada como cadena inválida y retornará un ERROR al manejador principal del programa, este valor retornado se identifica en una estructura switch case y se procede a su tratamiento, identificandolo como error léxico, guardando la cadena correspondiente además de su número de línea en el código del editor, todo esto para ser desplegado en la parte inferior del programa como en el ejemplo a continuación:



## Conclusión

La herramienta JFlex nos facilitó muchísimo el trabajo de analizar las expresiones, creo que la aplicación de lo que hemos visto en cuanto a expresiones regulares se dio más en la parte de definición de los distintos componentes léxicos y lexemas en específico.

El desarrollo del analizador completo implicó traer de nuevo nuestros conocimientos de programación vistos en materias de semestres anteriores ya que aunque el JFlex se encarga de la parte logica y mas pesada del analizador el formato que tomó ya en conjunto con el editor, las opciones de abrir, guardar, cerrar y lo demás incluido hacen que se vea una interfaz completa y útil para las personas que pretendan utilizar nuestro lenguaje de programación.

Se puede notar como varias herramientas fueron utilizadas y combinadas para poder llevar a cabo el análisis léxico de forma satisfactoria

# Bibliografía:

Gálvez Rojas, S., & Mora Mata, M. Á. (2005). *JAVA A TOPE: TRADUCTORES Y COMPILADORES CON LEX/YACC, JFLEX/CUP Y JAVACC.* Málaga: Universidad de Málaga.

*JFlex | Analizador léxico con Java (explicación paso a paso)*. (2019, 17 febrero). [Vídeo]. YouTube. https://www.youtube.com/watch?v=5mIRrn2yEe8&t=254s