UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA



MATERIA: COMPILADORES  
GRUPO : #1  
PROYECTO : Analizador Léxico

-Baez Cadena Diestefano Michel  
-Curiel Reyes Joshua

-Cadena Campos Luis  
-Tejada Orozco Diego Francisco

ÍNDICE

**Contenido**

[OBJETIVO 3](#_Toc56635170)

[DESCRIPCIÓN DEL PROBLEMA 3](#_Toc56635171)

[DISEÑO DE LA SOLUCIÓN 3](#_Toc56635172)

[IMPLEMENTACIÓN 4](#_Toc56635173)

[EXPRESIONES REGULARES 6](#_Toc56635174)

[CONCLUSIONES 10](#_Toc56635175)

# OBJETIVO

Los objetivos del proyecto son los siguientes:

* Que los alumnos aprendan a implementar sus conocimientos de LF & A, para darle una aplicación.
* Entender una de las fases en el proceso de compilación
* Aplicar el conocimiento adquirido en JFLEX de las prácticas 1 a 3
* Crear las expresiones regulares, que puedan reconocer las cadenas.

# DESCRIPCIÓN DEL PROBLEMA

La problemática que da origen al análisis e implementación de conocimientos teóricos y prácticos previos es la creación de un analizador lexico en JFLEX, que nos permita reconocer cada uno de los tokens que pueden formar parte del alfabeto para el lenguaje generado por la gramática. Es altamente recomendable que el analizador léxico solo retorne un entero por cada token encontrado. En caso de los operadores y de las palabras reservadas se recomienda devolver un entero por cada uno de los componentes de este grupo.

Para darle funcionalidad a lo anterior, se colocarán primero las expresiones regulares, mismas que nos permitirán posteriormente obtener un AFD. La creación y buena implementación del Analizador Léxico es un precedente base para darle solución a las siguientes fases de compilación.

# DISEÑO DE LA SOLUCIÓN

Para poder diseñar una solución óptima y adecuada en la creación del Analizador Léxico se deben generar los elementos necesarios para la gramática. Para empezar, se debe de hacer un análisis previo del problema (presentado anteriormente), para de ahí poder comprender las necesidades lógicas del mismo. De este modo sabemos entonces que el primer paso para llegar a una solución óptima es generar los elementos de las expresiones regulares. Una vez que estas hayan sido probadas, se deberá encontrar un modelo AFD para poder convertirlo y así hacer un uso correcto y lógico de las expresiones en el programa.

Una parte importante para poder llegar a la solución fue la de analizar las expresiones mediante elementos punteados:

# IMPLEMENTACIÓN

Implementamos este código haciendo las expresiones regulares, pero también haciendo un análisis mediante elementos punteados. Las expresiones regulares que se sometieron al análisis fueron las siguientes:

Expresiones Regulares.

Digito → [0-9]

Letra → [a-zA-Z]

Letra\_Compuesta → ({Letra} | \\_ )

Enteros → {Digito}+ ([ \_ ]{Digito}+)?

Cadenas → “ [^”]\* “

Identificador → {Letra} ({Letra\_Compuesta} | {Digitol})\*

Cerradura ({

Enteros → •{Digito}+ ([ \_ ]{Digito}+)?

Cadenas → • “ [^”]\* “

Identificador → •{Letra}({Letra\_Compuesta} | {Digito})\*

}) = q0

goto (q0, {Digito}) = {

Enteros → {Digito}+ (•[ \_ ]{Digito}+)?

Enteros → •{Digito}+ ([ \_ ]{Digito}+)?

Enteros → {Digito}+ ([ \_ ]{Digito}+)?•

} = q1

goto (q0, “) = {

Cadenas → “ •[^”]\* “

Cadenas → “ [^”]\*• “

} = q2

goto (q0, {letra}) = {

Identificador → {Letra} (•{Letra\_Compuesta} | {Digito})\*

Identificador → {Letra} ({Letra\_Compuesta} | •{Digito})\*

Identificador → {Letra} ({Letra\_Compuesta} | {Digito})\*•

} = q3

goto (q1, [ \_ ]) = {

Enteros → {Digito}+ ([ \_ ]•{Digito}+)?

} = q4

goto (q1, {Digito}) = q1

goto (q2, [^”]) = q2

goto (q2, “) = {

Cadenas → “ [^”]\* “•

} = q5

goto (q3, {letra\_Compuesta}) = q3

goto (q4, {Digito}) =

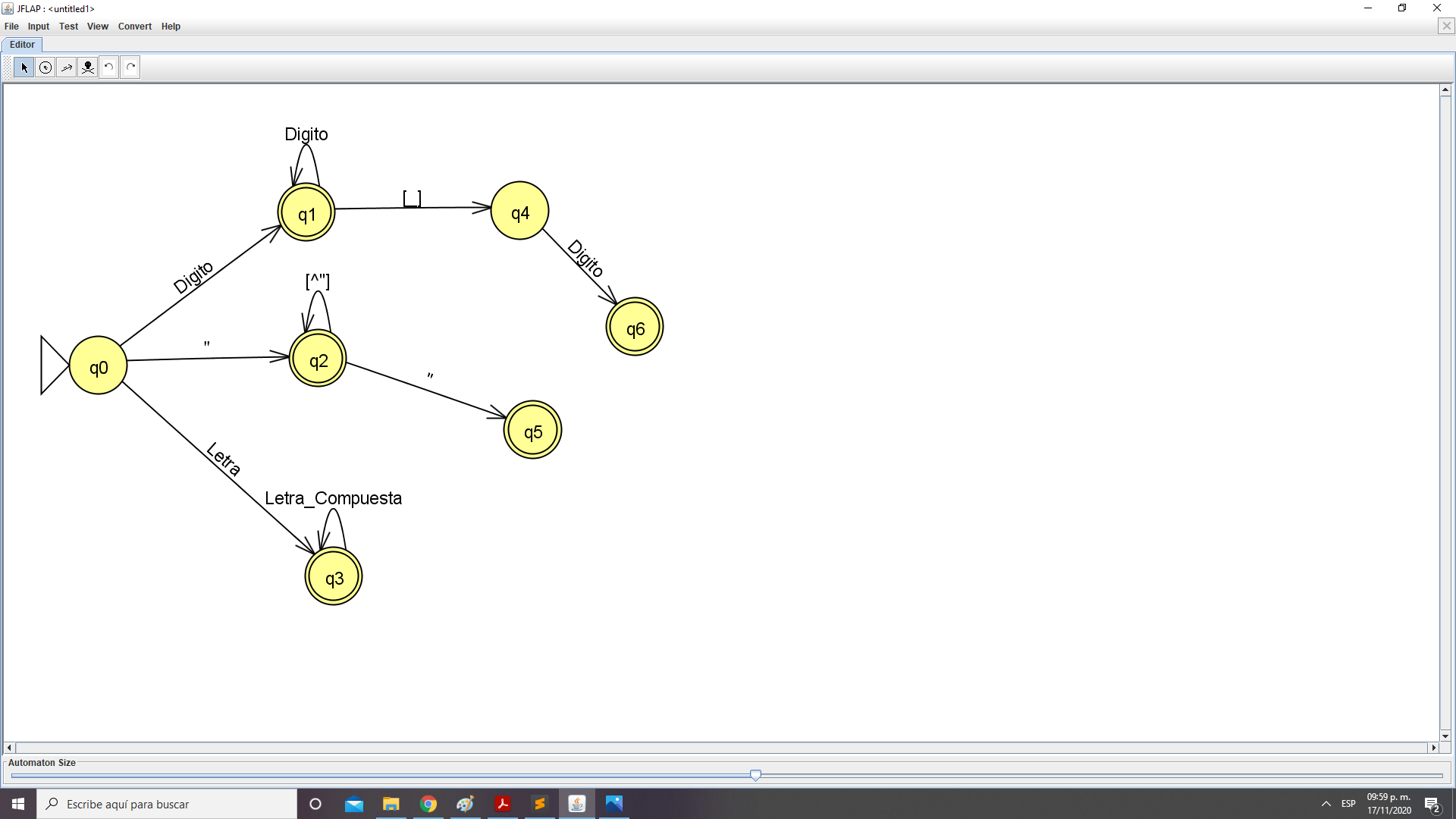
Enteros → {Digito}+ ([ \_ ]{Digito}+)?•

} = q6

Después de eso, obtenemos la respectiva tabla.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Edo | {Digito} | " | {letra} | [ \_ ] | [ ^" ] | {letra\_Compuesta]} |
| *→* | q0 | q1 | q2 | q3 | ø | ø | ø |
| \* | q1 | q1 | ø | ø | q4 | ø | ø |
| \* | q2 | ø | q5 | ø | ø | q2 | ø |
| \* | q3 | ø | ø | ø | ø | ø | q3 |
|  | q4 | q6 | ø | ø | ø | ø | ø |
| \* | q5 | ø | ø | ø | ø | ø | ø |
| \* | q6 | ø | ø | ø | ø | ø | ø |

Y el autómata obtenido por esta tabla, es el siguiente:



# EXPRESIONES REGULARES

import java.util.\*;

import java.io.File;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.IOException;

%%

%class Lexer

%int

//Definimos las expresiones regulares

//Identificadores

Letra =[a-zA-Z]

Digito=[0-9]

Letra\_Compuesta=({Letra}|\\_)

Identificador = {Letra}({Letra\_Compuesta}|{Digito})\*

//Enteros

Enteros ={Digito}+([\_]{Digito}+)?

//Exponentes

exponente = ([eE][+-]?{Enteros}+)

//Decimales

Dig\_decimal = ((({Entero})\*\.({Digito})+|({Entero})+\.({Digito})\*){exponente}?)|({Entero})+({exponente})

//Imaginarios

Imaginario = ({Entero}|{Dig\_Decimal})+'i'

//Cadenas

Cadenas=("[^']\*")

//Constantes booleanas

Booleanas = (True|False)

//op\_aritmetico =("\*"|"/"|"%"|"+"|"-")

//op\_unario

op\_unario=("--"|"++")

//Operadores de Asignacion

op\_asignacion =\=

op\_sum\_izq\_asig =\+=

op\_res\_izq\_asig =\-=

op\_mul\_izq\_asig =\\*=

op\_divi\_izq\_asig =\/=

op\_pot\_izq\_asig =\^=

op\_mod\_izq\_asig =\%=

//Operador Relacional

op\_igual\_que = "=="

op\_diferente\_que ="!="

op\_menor\_que = "<"

op\_menor\_igual\_que = "<="

op\_mayor\_que = ">"

op\_mayor\_igual\_que = ">="

//Operador Logico

op\_and\_logico ="&&"

op\_or\_logico="||"

//Operador Shift

op\_desplazo\_derecha =">>"

op\_desplazo\_izquierda ="<<"

//Operadores paréntesis

parentesis\_abre = \(

parentesis\_cierra = \)

op=({op\_aritmetico}|{op\_unario}|{op\_asignacion}|{op\_sum\_izq\_asig}|{op\_res\_izq\_asig}|{op\_mul\_izq\_asig}|{op\_divi\_izq\_asig}|{op\_pot\_izq\_asig}|{op\_mod\_izq\_asig}|{op\_igual\_que}|{op\_diferente\_que}|{op\_menor\_que}|{op\_menor\_igual\_que}|{op\_mayor\_que}|{op\_mayor\_igual\_que}|{op\_and\_logico}|{op\_or\_logico}|{op\_desplazo\_derecha}|{op\_desplazo\_izquierda}|{parentesis\_abre}|{parentesis\_cierra})

//Palabras reservadas

p\_reservada=(print|delete|main|public|static|void|abstract|assert|boolean|break|byte|case|catch|char|class|conts|continue|default|do|double|else|export|extends|for|if|new|package|private|protected|public|return|short|static|super|switch|this|throws|void|volatile|var|goto)

//Terminamos expresiones regulares

public static final int Identificador =0, Enteros=1,exponente=2,Dig\_decimal=3;

public static final int Imaginario =4, Cadenas=5,Booleanos=6,op\_aritmetico=7;

public static final int op\_unario =8, op\_asignacion=9,op\_sum\_izq\_asig=10;

public static final int op\_res\_izq\_asig=11, op\_mul\_asig\_izq =12, op\_divi\_izq\_asig=13;

public static final int op\_pot\_izq\_asig=14, op\_mod\_izq\_asig=15, op\_igual\_que =16;

public static final int op\_diferente\_que =17 ,op\_menor\_que =18;

public static final int op\_menor\_igual\_que=19, op\_mayor\_que=20;

public static final int op\_mayor\_igual\_que=21 , op\_and\_logico=22;

public static final int op\_or\_logico=23, op\_desplazo\_derecha =24;

public static final int op\_desplazo\_izquierda =25, parentesis\_abre=26;

public static final int parentesis\_cierra =27, op=28, p\_reservada=29;

public static void main (String[] args){

Lexer lexer = new lexer();

try{

br = new BufferedReader( new FileReader (new File(args[0])));

lexer.yylex();

} catch (IOException e){

System.out.println("El archivo no se pudo abrir");

}

}

%}

%eofval{

return 0;

%eofval}

%{

public string lexeme;

%}

{esps}{/\*ignorar\*/}

{Enteros}{System.out.println(Enteros);}

{exponentes}{System.out.println(exponentes);}

{Dig\_decimal}{System.out.println(Dig\_decimal);}

{Imaginario}{System.out.println(Imaginario);}

{Cadenas}{System.out.println(Cadenas);}

{Booleanos}{Syste.out.println(Booleanos);}

{op\_aritmetico}{Syste.out.println(op\_aritmetico);}

{op\_unario}{Syste.out.println(op\_unario);}

{op\_asignacion}{Syste.out.println(op\_asignacion);}

{op\_sum\_izq\_asig}{Syste.out.println(op\_sum\_izq\_asig);}

{op\_res\_izq\_asig}{Syste.out.println(op\_res\_izq\_asig);}

{op\_mul\_izq\_asig}{Syste.out.println(op\_mul\_izq\_asig);}

{op\_divi\_izq\_asig}{Syste.out.println(op\_divi\_izq\_asig);}

{op\_pot\_izq\_asig}{Syste.out.println(op\_pot\_izq\_asig);}

{op\_mod\_izq\_asig}{Syste.out.println(op\_mod\_izq\_asig);}

{op\_igual\_que}{Syste.out.println(op\_igual\_que);}

{op\_diferente\_que}{Syste.out.println(op\_diferente\_que);}

{op\_menor\_que}{Syste.out.println(op\_menor\_que);}

{op\_menor\_igual\_que}{Syste.out.println(op\_menor\_igual\_que);}

{op\_mayor\_que}{Syste.out.println(op\_mayor\_que);}

{op\_mayor\_igual\_que}{Syste.out.println(op\_mayor\_igual\_que);}

{op\_and\_logico}{Syste.out.println(op\_and\_logico);}

{op\_o\_logico}{Syste.out.println(op\_o\_logico);}

{op\_desplazo\_derecha}{Syste.out.println(op\_desplazo\_derecha);}

{op\_desplazo\_izquierda}{Syste.out.println(op\_desplazo\_izquierda);}

{parentesis\_abre}{Syste.out.println(parentesis\_abre);}

{parentesis\_cierra}{Syste.out.println(parentesis\_cierra);}

{p\_reservada}{Syste.out.println(p\_reservada);}

# CONCLUSIONES

Báez Cadena Diestefano Michel: Con esta práctica reforzamos los temas vistos sobre expresiones regulares, elementos punteados, y JFlex. Es muy importante hacer esta parte del análisis léxico porque es la primera fase de compilación, porque si desde aquí fallamos, todo lo que vendrá después estará mal, y el compilador fallará.

Curiel Reyes Joshua:

La elaboración del Analizador Léxico implica un análisis detallado de la gramática la cual será definida para que al momento de que el código fuente sea ingresado, se produzca una salida compuesta de Tokens o símbolos los cuales son necesarios en una etapa posterior que es el Análisis Sintáctico. El desarrollo de esta parte del proyecto en particular, presenta diversos retos entre ellos, el cómo definir las expresiones regulares de manera adecuada. Posterior a esto, el análisis de elementos punteados, para la obtención de elementos básicos y así poder realizar la construcción del AFD con los estados resultantes. Por otra parte otro reto que se presentó fue el uso e implementación de JFlex en Java para la creación del Analizador, identificando las secciones principales de un fichero JFlex el cual se compone de 3 secciones.

Cadena Campos Luis: Este proyecto hizo que pusiera en práctica lo hemos estado aprendiendo tanto en teoría como en las prácticas que nos ha dejado el profesor, sin embargo hacer las expresiones regulares fue un poco complicado, ya que en muchas ocasiones me pregunté si era correcto lo que estaba haciendo, sumando el hecho de que el crear las expresiones regulares no fue lo único complicado, recordar la programación en Java también dificulto un poco este proyecto, sin embargo aun con estos contratiempos hemos creado un proyecto bastante aceptable.

Tejada Orozco Diego Francisco: Con la realización de esta práctica, pudimos comprender mejor la realización de varios de los temas vistos en clase, desde la creación de AFDs a partir de expresiones regulares hasta la programación de un analizador léxico. Es importante mencionar, que todos los pasos fueron retadores y en cada uno de ellos surgieron dudas, por lo cual puedo concluir que el ejercicio cumplió con su objetivo de ampliar nuestro conocimiento y capacidad de análisis y abstracción del tema.