

INSTITUTO TECNOLÓGICO DE CELAYA



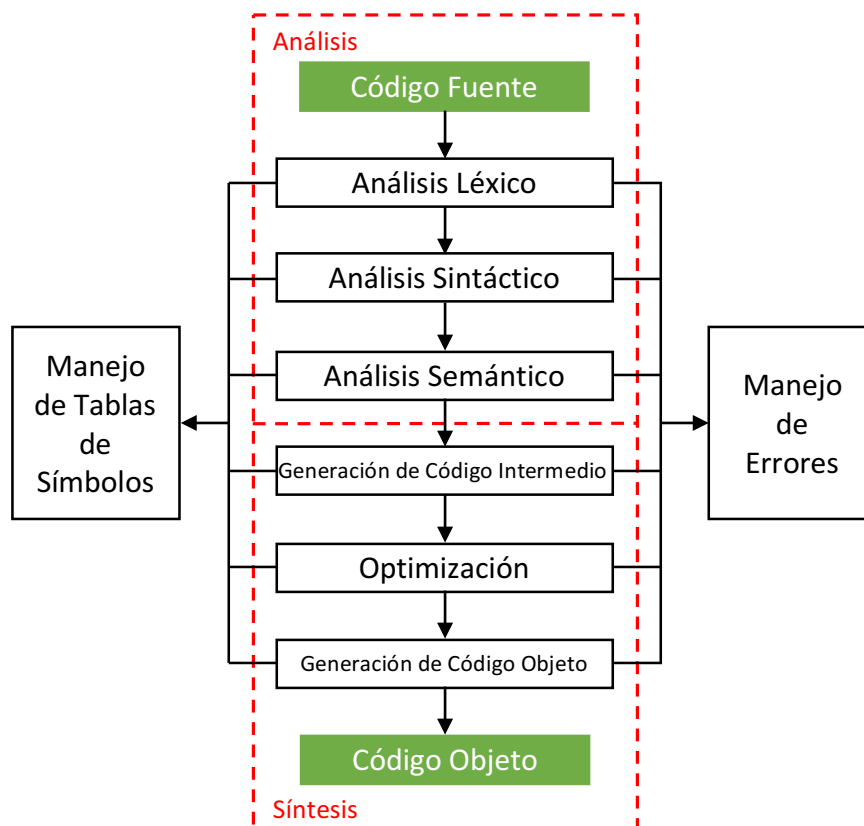
CARRERA	MATERIA
INGENIERÍA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTÓMATAS II
ALUMNO	ACTIVIDAD
JOEL JETZAHUEL TELLO ROJO	P2

NOMBRE DEL PROGRAMA	FECHA
ANALIZADOR SINTÁCTICO	2016-11-18

1 INTRODUCCIÓN

Un compilador es un programa que recibe como datos de entrada el código fuente de un programa escrito por un programador y lo transforma a instrucciones en binario que la computadora puede interpretar es decir un lenguaje objeto.

El proceso de compilación tiene básicamente dos etapas las cuales constan de diferentes fases, en la primera etapa llamada Análisis se llevan a cabo tres fases, un análisis léxico, un análisis Sintáctico y un análisis Semántico. En la segunda y última etapa del proceso llamada Síntesis igualmente consta de tres fases donde la primera es la generación de código intermedio, posteriormente la optimización del mismo y por último la generación de código objeto. Durante la realización de las dos etapas de análisis y síntesis pueden presentarse errores los cuales son tratados en cada una de las fases del proceso de compilación así como el manejo de tablas de símbolos.



Que el alumno sea capaz de comprender como funciona un compilador en su fase de análisis léxico, en cuanto a como es que se generan los tokens y como son tratados para la generación de la tabla de símbolos también se espera que el alumno desarrolle un programa que pueda generar un archivo que contenga un microprograma con el lenguaje que definió en la actividad 2, además el programa debe ser capaz de leer dicho archivo y separar en tokens el código fuente y categorizarlos en una tabla de símbolos.

Análisis Léxico

El análisis léxico o Scanner, es la primera fase del compilador que lee el código fuente, carácter por carácter y a partir de un patrón construye unas entidades primarias llamadas tokens que también son llamados componentes léxicos que serán utilizados en la siguiente fase.

Un token es una secuencia de caracteres que representa una unidad de información en el programa fuente, los tokens mas comunes son las palabras reservadas, identificadores, signos de puntuación, etc.

Un Lexema representan cadenas de caracteres en el programa fuente que se pueden tratar juntos como una unidad léxica. Un lexema es una secuencia de caracteres en el programa fuente con la que concuerda el patrón para un token.

Un patrón es una regla que describe el conjunto de lexemas que pueden representar a un determinado token en los programas fuente. En otras palabras, es la descripción del componente léxico mediante una regla o expresión regular.

Token	Lexema	Patrón
Identificador	indice, a1, map, n	Letra seguida de letras o dígitos.
int	12, 9, 129	Digito seguido de n dígitos.
if	if	Letra i seguida de letra f
Op_asig	=	Carácter =

La tabla de símbolos es una importante estructura de datos creada y mantenida por los compiladores con el fin de almacenar información acerca de la ocurrencia de diversas entidades, tales como nombres de variables, nombres de funciones, objetos, clases, interfaces, etc. tabla de símbolos se utiliza en el análisis y la síntesis de un compilador. Las operaciones que se puedan realizar sobre la tabla de símbolos es Insertar, Actualizar, Buscar.

La **tabla de símbolos** puede implementar en diferentes formas en función a la cantidad de datos que va a manejar.

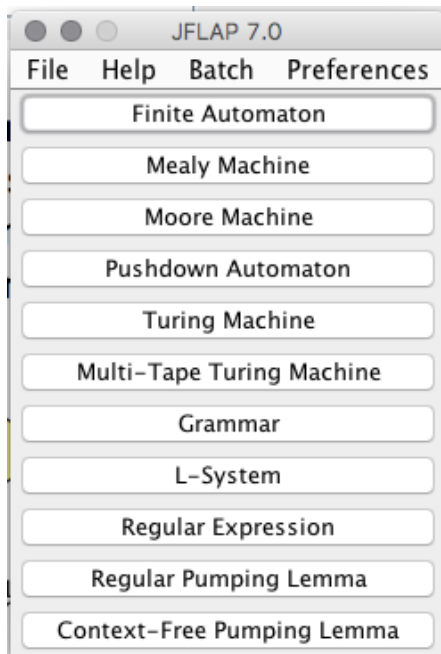
1- Lineal (ordenadas o desordenadas) lista 2.- Árbol de búsqueda binaria 3.-Tabla Hash

Una tabla de símbolos puede servir para los fines siguientes.

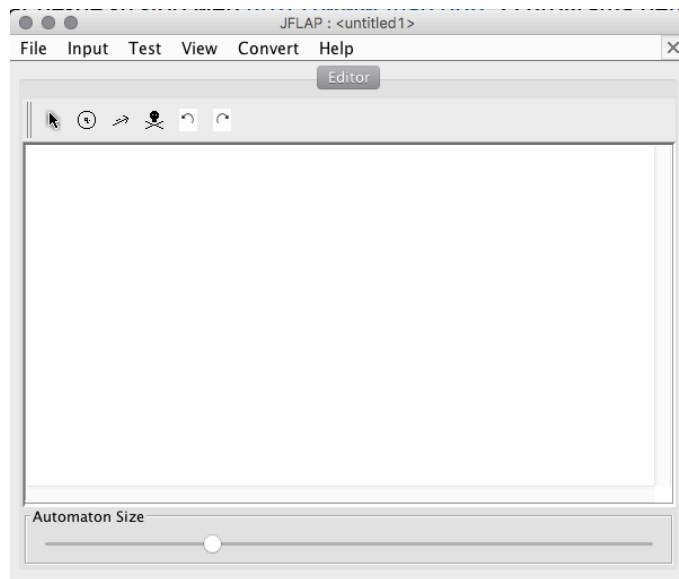
- Para almacenar los nombres de todas las entidades de forma estructurada en un solo lugar.
- Para verificar si se ha declarado una variable.
- Comprobación del tipo de implemento y las expresiones en el código fuente son semánticamente correcto.

Los **autómatas** son modelos matemáticos que reciben información, la transforman y producen otra información que se transmite al entorno, El autómata recibe los símbolos de entrada, uno detrás de otro, es decir secuencialmente. EL símbolo de salida que en un instante determinado produce un autómata, no sólo depende de el último símbolo recibido ala entrada, sino de toda la secuencia o cadena, que ha recibido hasta ese instante.

Para realizar el diseño de los autómatas se utilizo un programa de java que se llama **JFLAP** la versión 7.0 el cual puedes descargar desde su sitio web <http://www.jflap.org/>. El programa tiene la siguiente interfaz.

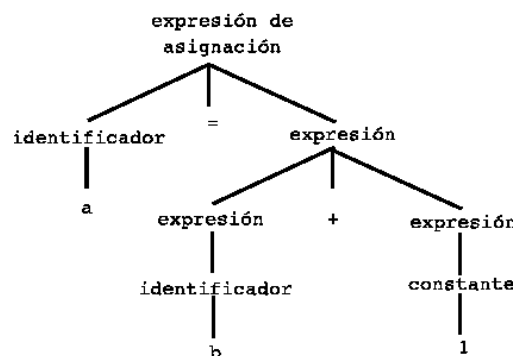


Para crear un autómata selecciona la primera opción "Finite Automaton" La cual nos llevara a la siguiente ventana donde de manera grafica podemos crear autómatas.



Análisis Sintáctico

La segunda fase del compilador es el **análisis sintáctico** o *parsing*. El parser (analizador sintáctico) utiliza los primeros componentes de los tokens producidos por el analizador de léxico para crear una representación intermedia en forma de árbol que describa la estructura gramatical del flujo de tokens. Una representación típica es el *árbol sintáctico*, en el cual cada nodo interior representa una operación y los hijos del nodo representan los argumentos de la operación.



Las fases siguientes del compilador utilizan la estructura gramatical para ayudar a analizar el programa fuente y generar el programa destino.

El analizador sintáctico también hace:

- Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador semántico).
- Chequeo de tipos (del analizador semántico).
- Generar código intermedio.
- Generar errores cuando se producen.

Es decir, realiza casi todas las operaciones de la compilación.

Existen 2 tipos generales de analizadores sintácticos para gramáticas:

a) Análisis sintáctico descendente(LL). Construye árboles de análisis sintáctico desde arriba (raíz) hacia abajo (hojas). El análisis se realiza de lo general a lo particular.

b) Análisis sintáctico ascendente(LR, LALR). Construyen árboles de análisis sintáctico comenzando en las hojas y suben hacia la raíz. El análisis se realiza de lo particular a lo general.

Ya sea ascendente o descendente se examina la entrada al analizador léxico de izquierda a derecha, un símbolo a la vez. La salida del analizador sintáctico es una representación del árbol de análisis sintáctico para la cadena de componentes léxicos producida por el analizador léxico.

La tabla de símbolos se crea durante la fase de análisis léxico a través de los componentes léxicos, pero en el proceso de análisis sintáctico sufren algunas modificaciones.

- Generalmente se agregan valores de tipo y significado para el análisis sintáctico

Manejo de errores sintácticos

El manejo de errores de sintaxis es el más complicado desde el punto de vista de la creación de compiladores. Nos interesa que cuando el compilador encuentre un error, se recupere y siga buscando errores. Por lo tanto el manejador de errores de un analizador sintáctico debe tener como objetivos:

- Indicar los errores de forma clara y precisa. Aclarar el tipo de error y su localización.
- Recuperarse del error, para poder seguir examinando la entrada.
- No ralentizar significativamente la compilación.

Estrategias de recuperación de errores

Ignorar el problema (Panic mode): Consiste en ignorar el resto de la entrada hasta llegar a una condición de seguridad. Una condición tal se produce cuando nos encontramos un token especial (un ';' o un 'END'). A partir de este punto se sigue analizando normalmente.

Recuperación a nivel de frase: Intenta recuperar el error una vez descubierto.

Reglas de producción adicionales para el control de errores: La gramática se puede aumentar con las reglas que reconocen los errores más comunes

Corrección Global : Dada una secuencia completa de tokens a ser reconocida, si hay algún error por el que no se puede reconocer, consiste en encontrar la secuencia completa más parecida que sí se pueda reconocer. Es

decir, el analizador sintáctico le pide toda la secuencia de tokens al léxico, y lo que hace es devolver lo más parecido a la cadena de entrada pero sin errores, así como el árbol que lo reconoce.

4 REQUISITOS BÁSICOS

- Un ordenador que cuente con un entorno de desarrollo que soporte java.
- Conocimientos en POO, Estructuras de Datos y manejo de ficheros.
- Compresión sobre como es que se lleva acabo el análisis léxico en el proceso de compilación.

5 DESARROLLO

Repositorio en github: <https://github.com/Jetzahel/AIIPF>

Para el desarrollo del compilador se diseño el siguiente lenguaje.

Alfabeto

a = [a, b, c, ..., z, A, B, C ..., Z, 0, 1, 2, ..., 9, { }, (,), +, =, *, <, >, ;]

Palabra Reservada	Equivalencia en Java	Tipo	Descripción
num	int	Entero	Se utiliza para guardar números enteros.
cad	string	Cadena de Texto	Se utiliza para guardar cadenas de texto.
si	if	Estructura de control	Si el valor de la condición evaluada es verdadero ejecuta el código que esta entre llaves
cont	for	Estructura de control	Si el valor de la condición evaluada es verdadero ejecuta el código dentro de las llaves hasta que la condición deje de ser verdadera incrementa un contador con cada iteración.
imp	print	Java.io	Imprime un mensaje.

Operadores

Símbolo	Descripción
+	Suma dos variables.
*	Multiplica dos variables y tiene prioridad sobre la suma
=	Asigna un valor a una variable.
<	Compara si la expresión anterior es menor que la expresión sucesora.
>	Compara si la expresión anterior es mayor que la expresión sucesora.

Delimitadores

{ }	Delimita un bloque de código.
()	Agrupar una expresión para darle prioridad. En caso de que sea una estructura de control agrupa la expresión a evaluar.
;	Termina una línea de código o separa argumentos dentro de un cont.

Sintaxis de un si

```
si(condición){  
    //bloque de código  
}
```

Sintaxis de un cont

```
Cont(variable;condición;incremento de variable){  
    //bloque de código  
}
```

Características de un identificador.

Están formados por letras y debe finalizar con un número.

No puede ser una palabra reservada. Las palabras reservadas son todas las que aparecen en la primera tabla.

Ejemplo de identificadores válidos:

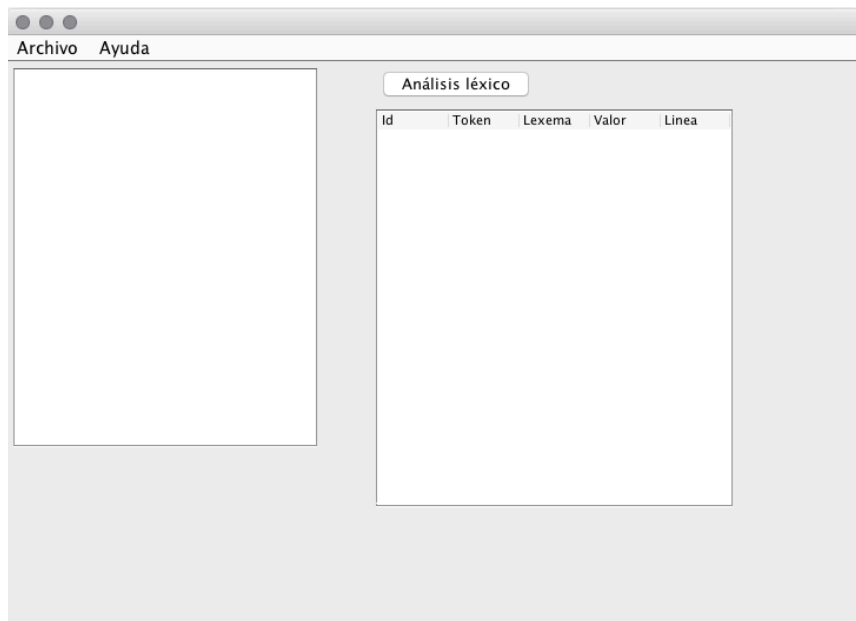
Edad1 nombre1 precio2 Edad2
cantidad0 precioventapublico1

Ejemplo de identificadores no válidos:

4num z# "Edad"

Se diferencia mayúsculas y minúsculas, por lo tanto, nombre1 y Nombre1 son identificadores con el diferente valor.

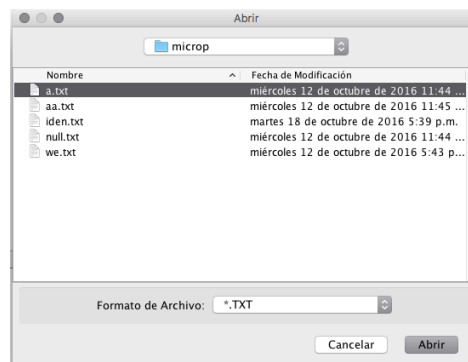
El programa inicia con la siguiente ventana.



El cual cuenta con un menú el primer elemento es Archivo del cual se despliega un submenú donde podemos generar un nuevo archivo, o abrir uno existente así como guardar uno que escribamos en el área de texto.



- La opción “Nuevo” limpiara el área de trabajo borrando todo lo que contenga.
- La opción “Abrir” abrirá una venta donde puedes seleccionar el archivo que desees abrir.



- La opción “Guardar” y “Guardar como” realizan la misma acción que es guardar un nuevo documento ya que para fines de ver los cambios en cada uno de los microprogramas se genera una nueva versión.

El programa cuenta con un botón el cual ejecuta el análisis léxico.

Análisis léxico

Para la asignación de id's a los tokens se utilizo la siguiente tabla

idToken	Token
10	si
11	cont
12	num
13	cad
14	imp
20	{
21	}
22	(
23)
30	identificador
31	numero
32	cadena
40	<
41	>
42	=
43	+
44	-
45	*
46	/
50	;

Al hacer clic en el botón el programa lee lo que hay en el área de trabajo y los clasifica en tokens asignándoles un id y los agrega en la tabla de símbolos agregando también el número de línea donde fueron escritos;

```
num    var0;  
si (var0 > 0){  
cad palabra0 = 'lopez';  
}
```

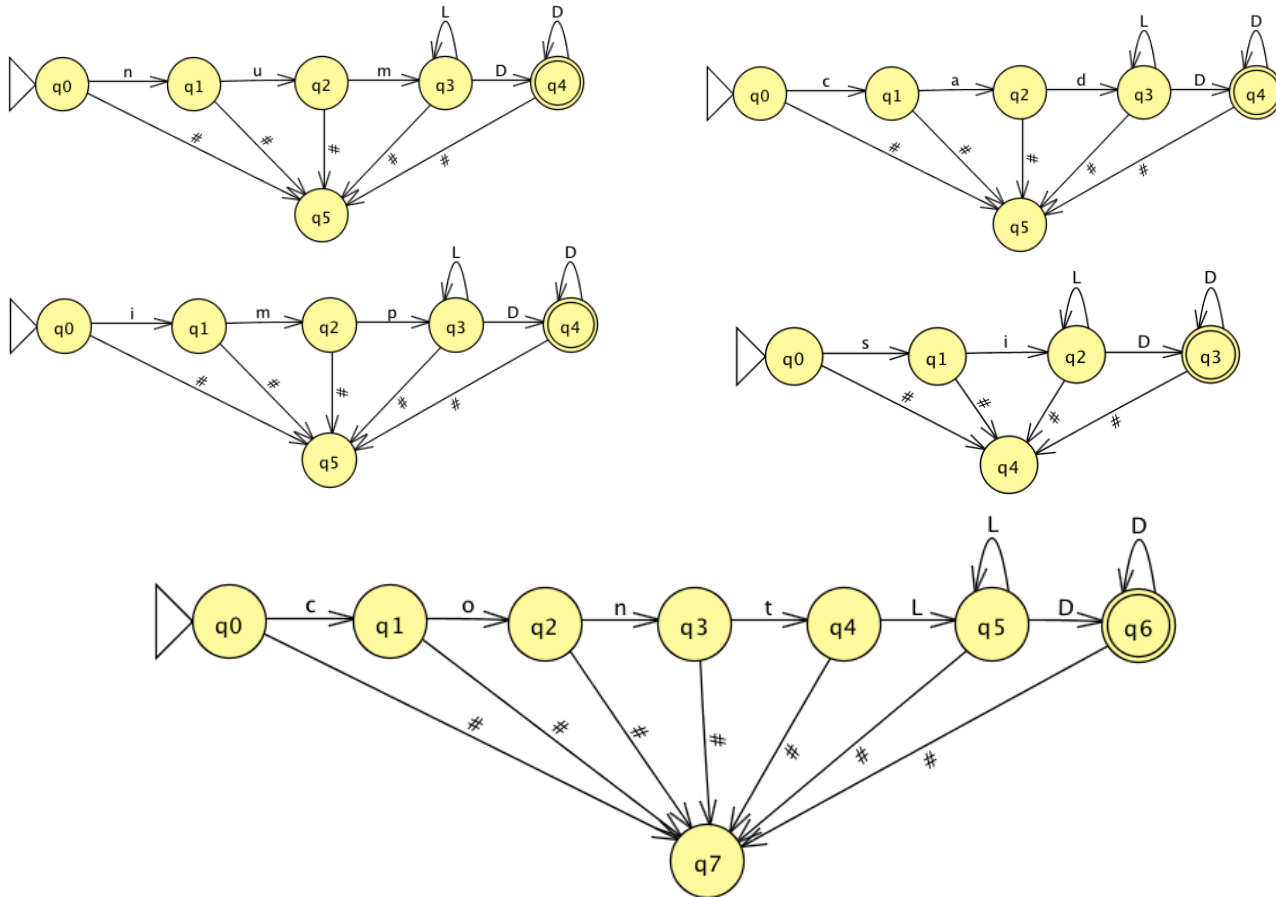
Análisis léxico

id	Token	Lexema	Linea
12	palabra_r	num	1
20	identificador	var0	1
52	delimitador	:	1
10	palabra_r	si	2
20	identificador	var0	2
41	mayor_q	>	2
21	numero	0	2
20	llave_abierta	{	2
13	palabra_r	cad	3
20	identificador	palabra0	3
42	Asignacion	=	3
14	cadena	'lopez'	3
52	delimitador	;	3
21	llave_cerra...	}	4

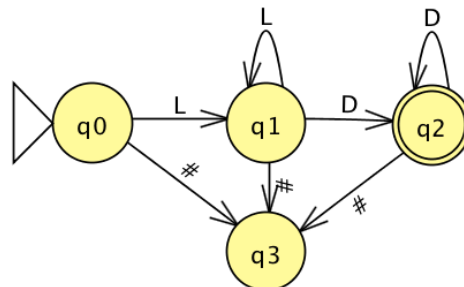
El algoritmo que sigue el análisis léxico

- 1.- el programa lee lo que contiene el área de trabajo.
- 2.- el programa separa en palabras lo que lee por espacios.
- 3.- el programa lee esas palabras para categorizarlas en tokens.
- 4.- el programa envía errores si es que existen.

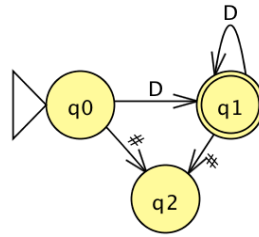
Estos autómatas validan que los identificadores que comienzan con una palabra reservada sean reconocidos como tal. La cadena debe de empezar por una o más letras mayúsculas o minúsculas (L) (Cualquier letra del abecedario), seguido de uno o más dígitos enteros (D) (Cualquier numero).



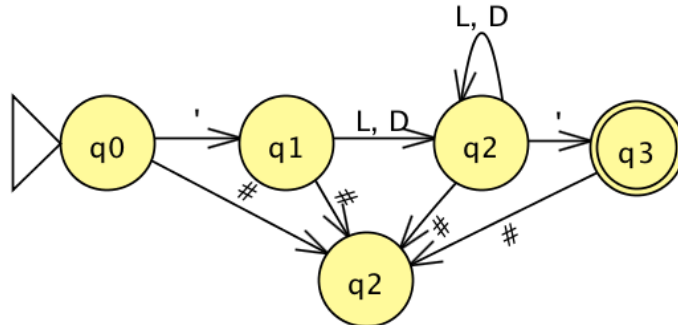
Este autómata valida que los identificadores comiencen con una o más letras mayúsculas o minúsculas (L) (Cualquier letra del abecedario), seguido de uno o más dígitos enteros (D) (Cualquier numero).



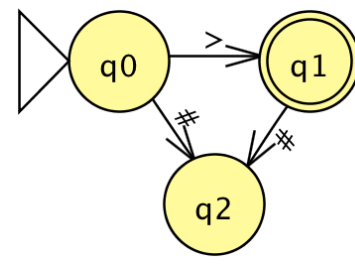
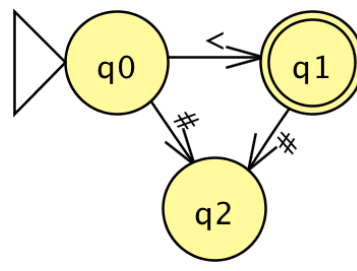
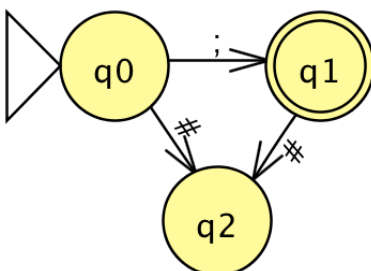
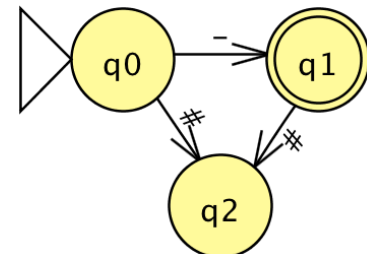
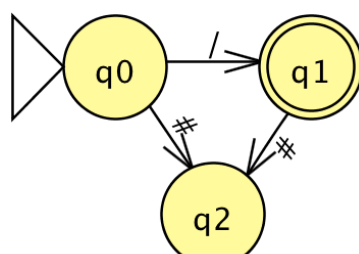
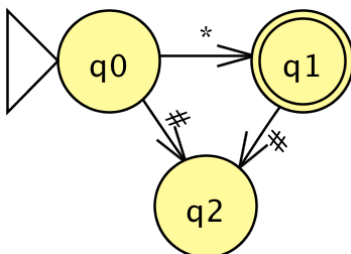
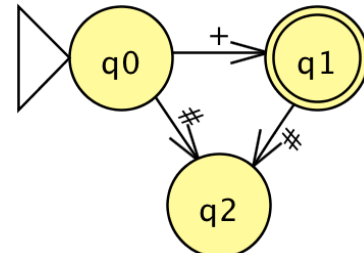
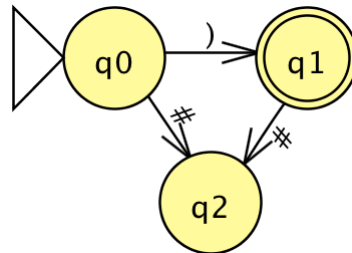
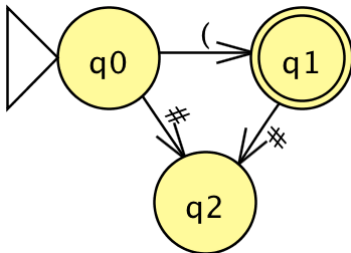
Este autómata valida que los números sean enteros. La cadena debe contener uno o más dígitos enteros (D) (Cualquier numero).

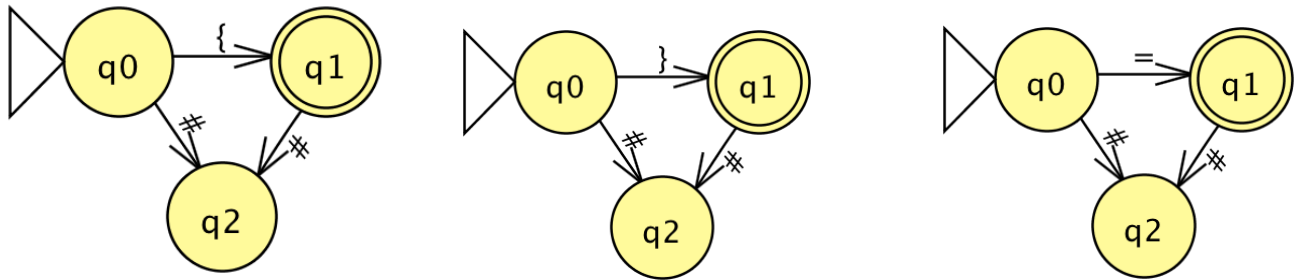


Este autómata valida las cadenas. La cadena debe comenzar con 'después puede contener cualquier cantidad de letras y números (L, D) y debe finalizar con '.

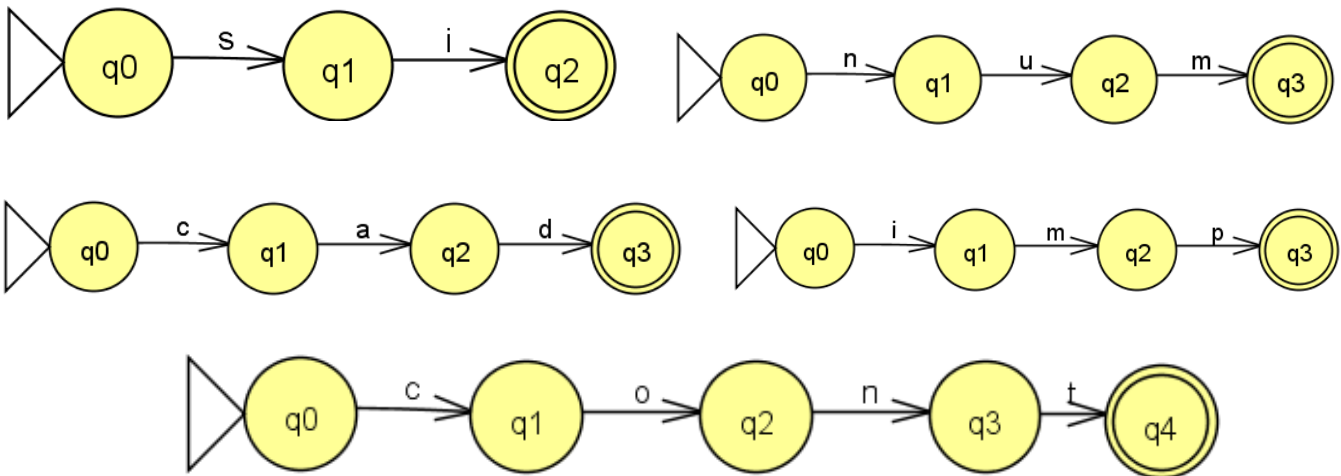


Los siguientes autómatas tienen la función de reconocer los paréntesis, los delimitadores, las llaves, los operadores.





Los siguientes autómatas reconocen las palabras reservadas establecidas en el lenguaje.



El siguiente código es un ejemplo de cómo fue que implemente los autómatas para realizar la clasificación de tokens.

```

for(int indice=0;indice<t.length();indice++){//recorre toda la cadena t
    char letra = t.charAt(indice);//t es la variable que contiene la cadena de texto a evaluar
    //letra tiene un caracter de la cadena t

    switch (Estado){
        case 0:
            if(letra == 's')
                Estado = 1;
            cadenacom += letra;
            break;
        case 1:
            if(letra == 'i')
                Estado = 2;
            cadenacom += letra;
            break;
        case 2:

            if(Character.isLetter(letra)){
                Estado = 2;

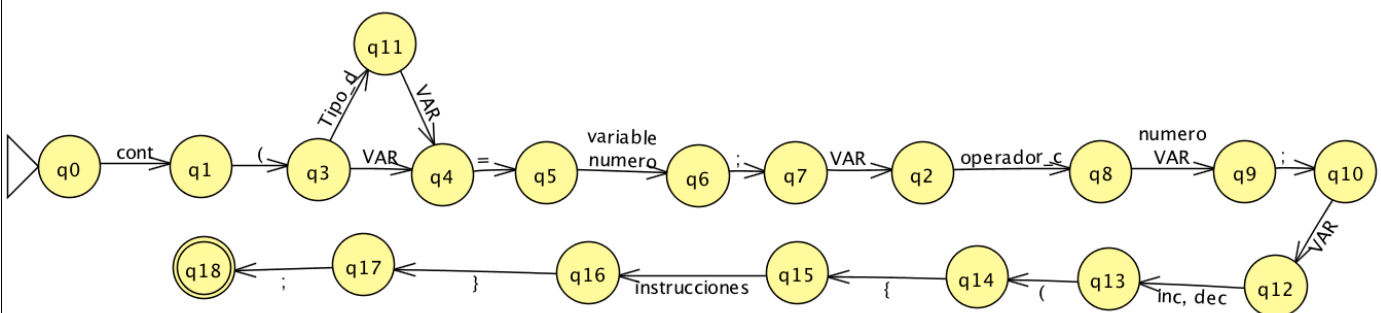
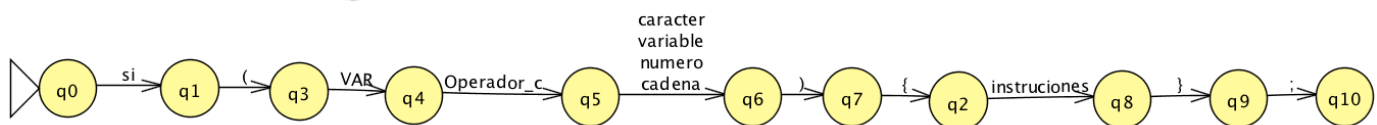
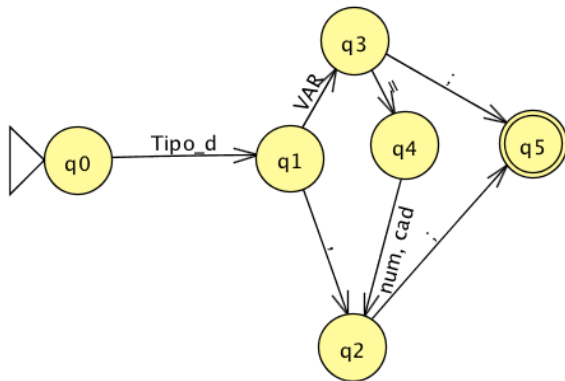
```

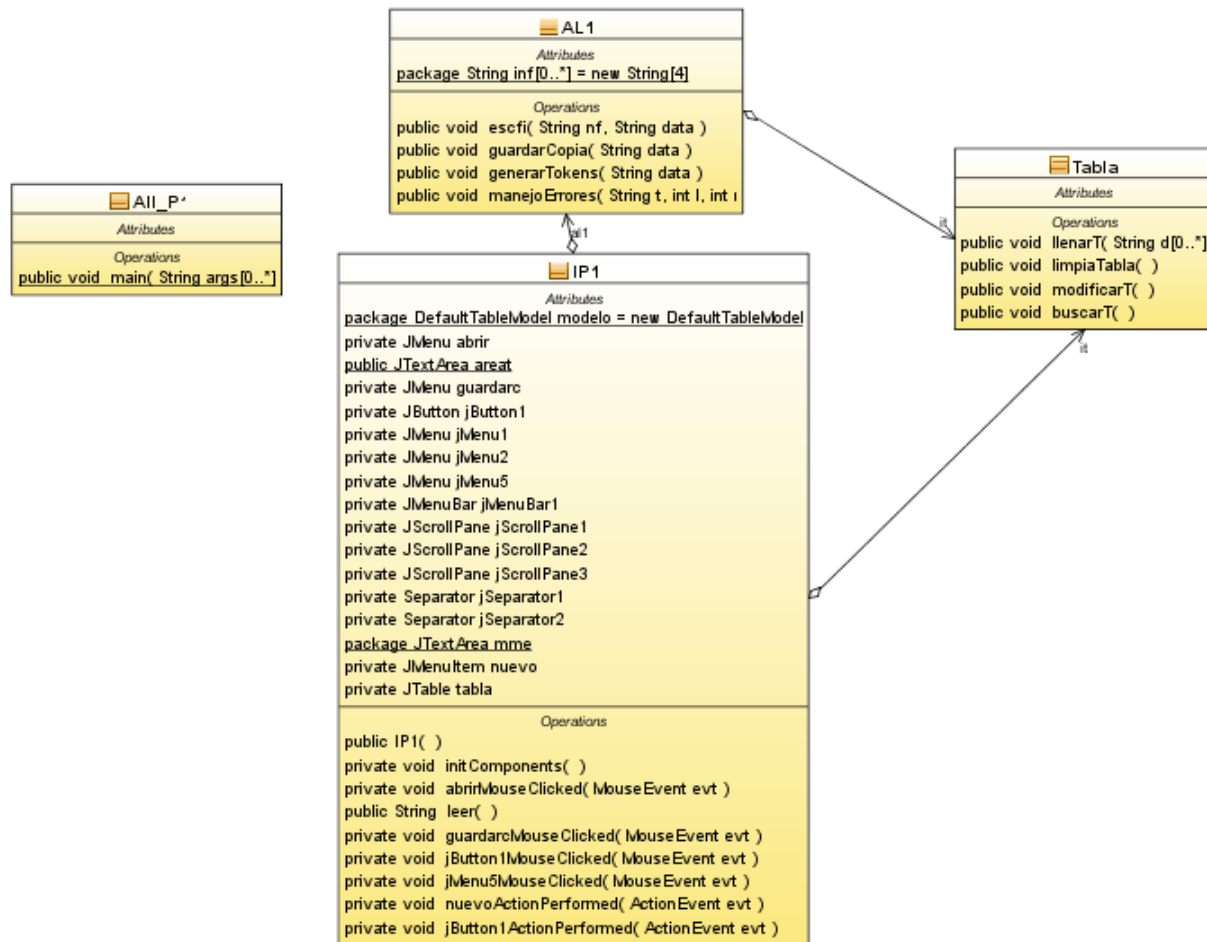
```

cadenacom += letra;}
if(Character.isDigit(letra)){
    Estado = 3;
    cadenacom += letra;
    linea--;
}
break;
case 3://el estado 3 es el estado de aceptación y es donde se le asigna un id y se agrega la tabla.
    System.out.println("na");
    inf[0]="20";
    inf[1]= "identificador";
    inf[2]=cadenacom;
    inf[3]=String.valueOf(linea);
    it.llenarT(inf);
break;
} //fin switch
} //fin for

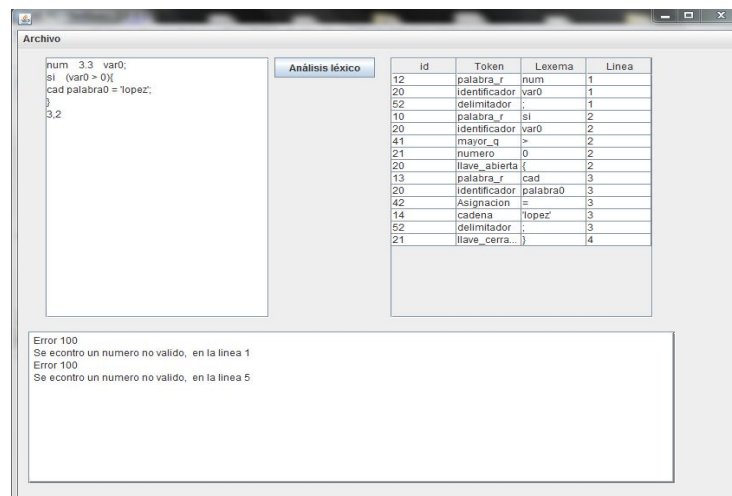
```

Para el análisis sintáctico se implementaran los siguientes autómatas.





Caso 1: El usuario ingresa un número que no está formado de manera correcta o con caracteres inválidos. Por ejemplo: 10;42, 10,42, 3.14.16



Caso 2: El usuario ingresa algún carácter que no está definido dentro del lenguaje(@,?,\$,&). El compilador no reconoce estos signos.

Archivo

```
num ? var0;
si (var0 > 0){ $
cad palabra0 = 'lopez'; &
}
@
```

Análisis léxico

id	Token	Lexema	Linea
12	palabra_r	num	1
20	identificador	var0	1
52	delimitador	;	1
10	palabra_r	si	2
20	identificador	var0	2
41	mayor_q	>	2
21	numero	0	2
20	llave_abierta	{	2
13	palabra_r	cad	3
20	identificador	palabra0	3
42	Asignacion	=	3
14	cadena	'lopez'	3
52	delimitador	;	3
21	llave_cerra...	}	4

Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 1
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 2
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 3
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 5

Caso 3: El usuario escribe una palabra reservada con errores de ortografía, cambiando el orden de las letras u omite alguna, el programa reconocerá esos errores como caracteres que no pertenecen al lenguaje.

Archivo

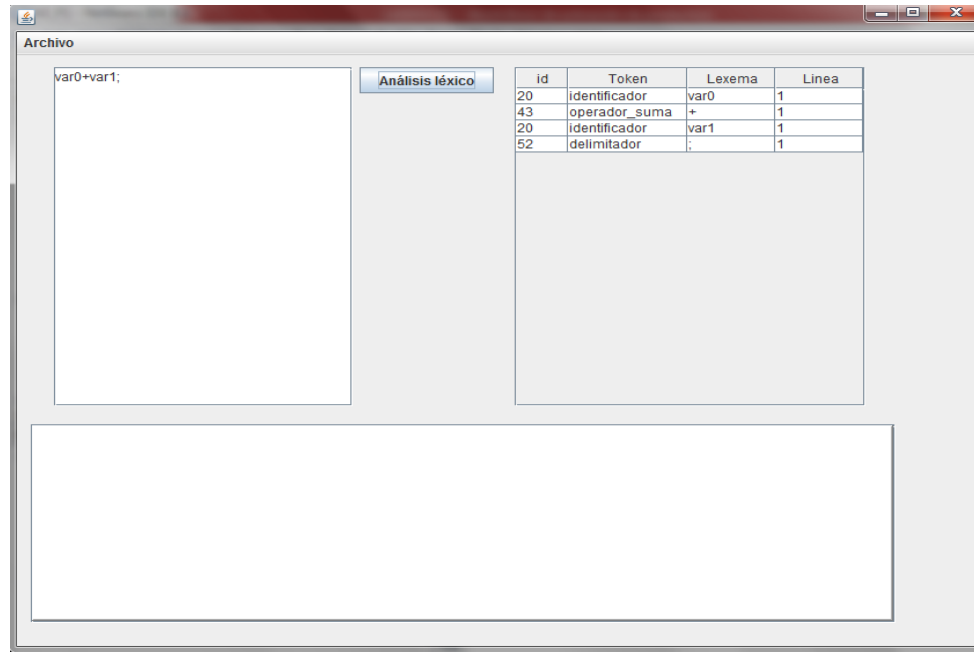
```
num var0;
si (var0 > 0){
cad palabra0 = 'lopez';
}
s
nmu
```

Análisis léxico

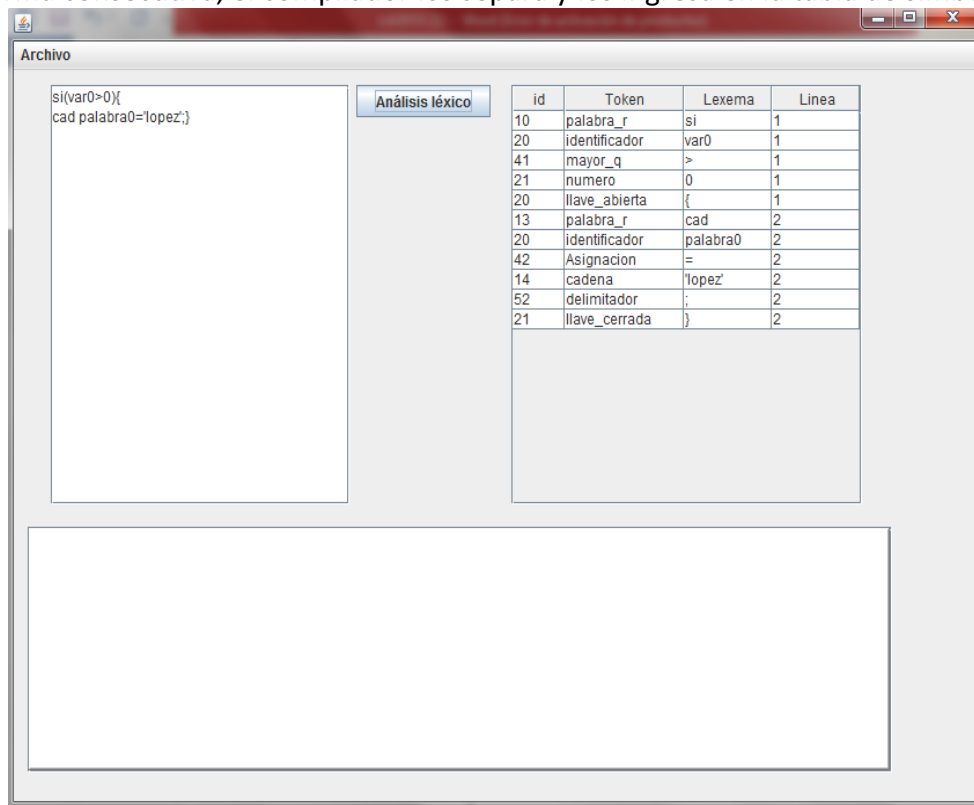
id	Token	Lexema	Linea
12	palabra_r	num	1
20	identificador	var0	1
52	delimitador	;	1
10	palabra_r	si	2
20	identificador	var0	2
41	mayor_q	>	2
21	numero	0	2
20	llave_abierta	{	2
13	palabra_r	cad	3
20	identificador	palabra0	3
42	Asignacion	=	3
14	cadena	'lopez'	3
52	delimitador	;	3
21	llave_cerra...	}	4

Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 5
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 6
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 6
Error 101
Se encontro un caracter que no pertenece al lenguaje, en la linea 6

Caso 4: El usuario escribe identificadores con signos (sin espacios) reconocidos por el lenguaje. El compilador separa el identificador del operador, los guarda y los categoriza por separado en la tabla de símbolos.



Caso 5: El usuario escribe más de un carácter que forme parte del lenguaje. Cuando el usuario escriba varios caracteres de forma consecutiva, el compilador los separa y los ingresa en la tabla de símbolos.



Casos de uso de Análisis Sintáctico

Caso 1

El usuario intenta declarar una variable de tipo entero. Si está bien escrita (<palabra reservada num> <Identificador> <signo asignación> <número> <delimitador>) entonces el compilador toma la declaración como correcta y procede a la siguiente línea, si no, entonces el compilador le hace saber al usuario por medio de la consola de errores que hace falta ya sea el signo de asignación, el delimitador o el identificador.

The screenshot shows a compiler interface with the following components:

- Code Editor:** Contains the following code:

```
1 num
2 num var1
3 num var2=
4 num var3=9
```
- Buttons:** "Análisis léxico" (green) and "Análisis Sintáctico" (red).
- Token Table:** A table with columns idToken, Lexema, Linea, and Valor.
- Error Log:** A list of error messages.

idToken	Lexema	Linea	Valor
12	num	1	
12	num	2	
30	var1	2	
12	num	3	
30	var2	3	
42	=	3	
12	num	4	
30	var3	4	9
42	=	4	
31	9	4	

Error Log:

- Error 200: Se esperaba un identificador, en la linea 1
- Error 201: Se esperaba un '=' o un ';', en la linea 2
- Error 202: Se esperaba un numero, en la linea 3
- Error 203: Se esperaba un ';', en la linea 4

Caso 2

El usuario intenta incluir una operación aritmética en su código fuente. Si la operación está bien escrita y contiene la estructura <identificador> <operador aritmético> <identificador | número> <delimitador> entonces el compilador lo toma en cuenta como correcta y prosigue a la siguiente línea, si está mal, entonces el compilador le hace saber al usuario por medio de la consola de errores que el código requiere un operador aritmético, un número o identificador o delimitador.

Caso 3

El usuario intenta declarar una variable de tipo cadena de texto. Si está bien escrita (<palabra reservada cad> <Identificador> <signo asignación> <texto entre comillas simples> <delimitador>) entonces el compilador toma la declaración como correcta y procede a la siguiente línea, si no , entonces el compilador le hace saber al usuario por medio de la consola de errores que hace falta ya sea el signo de asignación, el delimitador o el identificador.

The screenshot shows a compiler interface with the following components:

- Code Editor:** Contains four lines of code:

```
1 cad
2 cad var1
3 cad var2=
4 cad var3='hola'
```
- Buttons:** Two buttons are present: "Análisis léxico" (Lexical Analysis) and "Análisis Sintáctico" (Syntactic Analysis).
- Token Table:** A table showing the results of lexical analysis.

idToken	Lexema	Linea	Valor
13	cad	1	
13	cad	2	
30	var1	2	
13	cad	3	
30	var2	3	
42	=	3	
13	cad	4	
30	var3	4	
42	=	4	
32	'hola'	4	'hola'
- Error Console:** Displays a list of errors:

```
Error 200
Se esperaba un identificador, en la línea 1
Error 201
Se esperaba un '=' o un ';', en la línea 2
Error 204
Se esperaba una cadena, en la línea 3
Error 203
Se esperaba un ';', en la línea 4
```

Caso 4

El usuario omite algún paréntesis o alguna llave en el código fuente. El compilador le hace saber al usuario ya sea la falta de paréntesis o de llaves de apertura o de cierre.

Caso 5

El usuario intenta compilar un código que contiene dos signos de asignación (=) seguidos en una línea que no contiene una expresión para una condición o un ciclo. El compilador le hace saber al usuario que existe un error en dicha línea y le avisa que falta un operador aritmético, un delimitador, etc.

Caso 6

El usuario intenta compilar un código fuente en el cual se tiene una condición si con una estructura incorrecta. El compilador le hace saber al usuario por medio de la consola de errores el elemento faltante en la sintaxis de la instrucción SI. La cual debe ser:

<SI> <Paréntesis Apertura> <identificador> <operador lógico>

8 BITÁCORA			
FECHA	HORA	DESCRIPCIÓN	SOLUCIÓN
25/10/2016	11:00	El primer problema que enfrente fue como manejar la implementación de autómatas en el P1 a la hora de generar tokens.	Pensé en separar cada uno de los diferentes tipos de tokens en pequeños autómatas y manejarlos como métodos dentro del programa así que diseñe primero todos los pequeños autómatas para cada uno de los diferentes tipos de tokens.
26/10/2016	03:00	Cuando el autómata llega al estado de aceptación no guarda los tokens en la tabla.	Estaba haciendo mal la referencia de la tabla por eso no los imprimía en la tabla
27/10/2016	14:00	En el autómata para reconocer cadenas no reconocía las cadenas se perdía en algún estado	Tenía mal la transición del estado 2 al estado 3 no lo mandaba a ningún estado por lo tanto siempre se quedaba en ese estado, así que solo fue cuestión de hacer la transición de un estado a otro.
27/10/2016	9:00	Cuando hacia el recorrido de la cadena y ser evaluado con un autómata no recorría toda la cadena.	Tenía mal una condición dentro de un for, así que use herramienta de java.length para que recorriera la cadena completa.
7/11/2016		En la revisión del programa se hizo la observación de que los autómatas les hacia falta un estado el cual reconociera algún carácter que no perteneciera al token.	Se implemento un nuevo estado con la función de continuar con el siguiente autómata y si era el ultimo autómata mandar un error.
14/11/2016		Tenia que agregar el campo valor a la tabla de símbolos y darle valor a cada variable si es que esta estaba declarada, pero cuando tenia mas una variable declarada cambiaba el valor de las anteriores a cero.	Cuando el programa iba a asignar un valor a una variable primero verifica que campo valor este vacío si lo esta guarda el valor si no pues no cambia nada.
15/11/2016		En los autómatas que verifican el análisis sintáctico no entraba al estado final.	El programa no entraba el estado final porque la condición del For no se cumplía incremente la variable para que entrara una vez mas al For y así entrara nuevamente al switch .
15/11/2016		Cuando incremente la variable evaluada en la condición del for al momento de leer un valor de la pila marcaba error.	El error era por que la pila ya estaba vacía así que implemente un if para que si la pila no estaba vacía tomara el tope de pila, pero si estaba vacía no entraría a leer esa línea de código.

16/11/2016		Para implementar la función de que los botones se activaran o se desactivaran si existía código fuente, pero no se activaban hasta que tenía más de un carácter.	Estaba usando un Evento llamado "KeyPressed" después lo cambie por un "KeyTyped" pero seguía pasando lo mismo así que use el "KeyReleased".
------------	--	--	---

9	REFERENCIAS
<ul style="list-style-type: none"> • https://www.tutorialspoint.com/es/compiler_design/compiler_design_symbol_table.htm • http://puntocomnoesunlenguaje.blogspot.mx/p/teoria_7.html • http://codigosparadesarrolladores.blogspot.mx/2013/12/Codigo-JAVA-Agregar-datos-de-un-formulario-a-una-tabla-y-eliminar-una-fila-y-o-registro-de-una-tabla.html • http://compiladorsistemas.blogspot.mx/2010/11/lexemas-patrones-y-tokens.html • http://puntocomnoesunlenguaje.blogspot.mx/2013/07/ejemplos-expresiones-regulares-java-split.html • https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/La_clase_StringTokenizer • https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html • http://compiladoresasignatura.blogspot.mx/2011/05/unidad-vii-manejo-de-errores.html • http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf • http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2_u4.pdf • 	

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: Solicitud de Actividades

Celaya, Gto., 2016-11-13

LENGUAJES Y AUTÓMATAS II

PROFESOR DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

ACTIVIDAD P2 (VALOR 40 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTE ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN EL DOCUMENTO DE LA **GUÍA TUTORIAL**, ASÍ COMO EN LAS OBSERVACIONES HECHAS EN CLASE POR EL TITULAR DE ESTA MATERIA.:

- EN EQUIPO, IMPLEMENTE LA **PRIMERA ETAPA** DE UN ANALIZADOR **SINTÁCTICO**.

ESTA ETAPA CONSISTE EN GENERAR UN PROTOTIPO DE UN ANALIZADOR SINTÁCTICO, QUE REVISE A PARTIR DE LOS LEXEMAS O *TOKEN's*, LA CORRESPONDENCIA CON LA SINTAXIS DE LOS AUTÓMATAS DISEÑADOS PREVIAMENTE.

- IMPLEMENTE LAS ESTRUCTURAS DE DATOS, Y LOS AUTÓMATAS NECESARIOS QUE CORRESPONDAN A LA REVISIÓN SINTÁCTICA DE LOS ELEMENTOS DEL LENGUAJE PROTOTIPO DEFINIDO DESDE EL INICIO DEL PROYECTO.

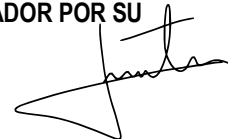
USE EL PROGRAMA DENOMINADO **JFLAP** (<http://www.jflap.org/>) PARA APOYARSE EN EL DESARROLLO DE ESTE PUNTO.

MUY IMPORTANTE :

- A. AL MOMENTO DE REALIZAR LA IMPLEMENTACIÓN DE SU ANALIZADOR SINTÁCTICO, ELABORE UNA BITÁCORA ORDENADA CRONOLÓGICAMENTE, EN LA CUAL COMO EQUIPO, REGISTREN LOS PROBLEMAS Y LA FORMA EN QUE LOS RESOLVIERON EN LA PROGRAMACIÓN.

ESTA BITÁCORA ES MUY IMPORTANTE LA PRESENTEN POR FECHA Y CORRECTAMENTE REDACTADA TANTO LA PROBLEMÁTICA, COMO LA SOLUCIÓN. ADEMÁS, DEBE INCLUIR DICHA BITÁCORA EN EL ARCHIVO PDF QUE HABRÁ DE ENTREGAR EL FINAL DE ESTA ETAPA.

- B. AL DOCUMENTO PDF, DEBERÁ INTEGRAR UN DIAGRAMA DE BLOQUE MUY BIEN DETALLADO QUE MUESTRE LA ARQUITECTURA QUE DIO A SU APLICACIÓN.
- C. LLEVE UN CONTROL DE VERSIONES DE CADA APLICACIÓN QUE ELABORE, DONDE CAMBIOS O MODIFICACIONES SIGNIFICATIVAS SEAN CONSIDERADAS COMO UNA NUEVA VERSIÓN O SUB-VERSIÓN.
- D. PROPONGA LOS CASOS DE USO PARA EL ANÁLISIS SINTÁCTICO, DOCUMENTE ÉSTOS Y GENERE PEQUEÑOS PROGRAMAS CON ERRORES INTENCIONALES QUE SERÁN REVISADOS Y SEÑALADOR POR SU PROTOTIPO.



- E. COMIENCE LA CREACIÓN DE UNA INTERFACE GRÁFICA EN LA CUAL, UN USUARIO FINAL PUEDA ADMINISTRAR ARCHIVOS CON CÓDIGO FUENTE DE SU LENGUAJE. ADEMÁS ESTA INTERFACE DEBERÁ IMPLEMENTAR UNA BARRA DE MENÚ Y BRINDAR UN EDITOR DE TEXTO, QUE MOSTRARÁ LAS LÍNEAS CÓDIGO FUENTE PERFECTAMENTE NUMERADAS.
- F. LA INTERFACE DE USUARIO TAMBIÉN DEBERÁ CONTENER UNA VISTA DE LA TABLA DE SÍMBOLOS ASÍ COMO UN PANEL PARA MOSTRAR LOS MENSAJES, ADVERTENCIAS Y ERRORES.
- G. POR ÚLTIMO AMPLÍE SU CATÁLOGO DE ERRORES (*ERRORES LÉXICOS*) , PARA AGREGAR AQUELLOS QUE SE PRESENTEN EN EL ANÁLISIS SINTÁCTICO. ESTE CATÁLOGO DEBERÁ MOSTRARSE EN UNA OPCIÓN DE AYUDA, EN LA MISMA BARRA DE MENÚ DE LA INTERFACE.
- H. IMPLEMENTE UN SEMÁFORO A SU INTERFACE GRÁFICA EN LA CUAL DEBERÁ CONTAR CON TRES ELEMENTOS. EJEMPLOS DE LA OPERACIÓN DEL SEMÁFORO :

NO SE HA CARGADO ARCHIVO CON PROGRAMA O BIEN NO SE HA ESCRITO NADA EN EL EDITOR. EL GRIS INDICA DESHABILITADO, NO EN EJECUTABLE POR EL MOMENTO.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

SE CARGÓ ARCHIVO CON PROGRAMA O BIEN SE ESCRIBIÓ AL MENOS UN CARÁCTER EN EL EDITOR.

INICIO DEL ANÁLISIS. EL AMARILLO INDICA LISTO PARA INICIAR.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS LÉXICO INCORRECTO. CON ERRORES.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS LÉXICO CORRECTO. SIN ERRORES Y ANÁLISIS SINTÁCTICO A LA ESPERA DE INICIAR.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS SINTÁCTICO INCORRECTO. CON ERRORES.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS LÉXICO Y SINTÁCTICO CORRECTOS. SIN ERRORES Y ANÁLISIS SEMÁNTICO A LA ESPERA DE INICIAR.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS LÉXICO, SINTÁCTICO CORRECTOS. SIN ERRORES. ANÁLISIS SEMÁNTICO INCORRECTO, CON ERRORES.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO

ANÁLISIS LÉXICO, SINTÁCTICO Y SEMÁNTICO CORRECTOS. SIN ERRORES. TERMINA EL PROCESO DE ANÁLISIS EN SUS TRES ETAPAS PRIMARIAS.

ANÁLISIS LÉXICO ANÁLISIS SINTÁCTICO ANÁLISIS SEMÁNTICO



- I. POR ÚLTIMO, LA INTERFACE DEBERÁ TRABAJAR EN **DOS MODALIDADES. PASO A PASO** A TRAVÉS DEL SEMÁFORO ANTERIOR, Y **EN UN SOLO PASO**. DONDE EJECUTARÁ EL PROCESO DE ANÁLISIS DE INICIO A FIN.

IMPORTANTE : EL SEMÁFORO SOLO ESTARÁ COMPLETAMENTE TERMINADO HASTA CONCLUIR LA IMPLEMENTACIÓN DE LA ACTIVIDAD P3.

LA REDACCIÓN DEL REPORTE DE ESTA ACTIVIDAD LO PUEDE DESCARGAR DE LA SIGUIENTE LIGA :

http://sisinfo.itc.mx/ITC-APIRGG/Actividades/2016-08-20_EJEMPLO_FORMATO_PRACTICA.PDF

NOTAS

- EL EJEMPLO INDICADO EN LA LIGA, SOLO DEMUESTRA LAS SECCIONES MÍNIMAS QUE DEBE CONTENER DICHA PRÁCTICA O IMPLEMENTACIÓN DE LA FASE.
- UNA SECCIÓN QUE DEBE CONSIDERAR ANEXA A ESTE FORMATO, ES LA DE **BITÁCORA DE INCIDENCIAS**, DONDE DEBERÁ DESCRIBIR LOS POR MENORES DE LOS PROBLEMAS AFRONTADOS, ASÍ COMO SU SOLUCIÓN.

CADA INCIDENCIA POR LO MENOS, DEBE LLEVAR **FECHA, HORA, DESCRIPCIÓN Y SOLUCIÓN** DE LA MISMA.
- SUSTENTE CADA UNA DE LAS ACCIONES IMPLEMENTADAS EN PROGRAMACIÓN CON LA TEORÍA CORRESPONDIENTE.
- PUEDE USAR CUALQUIER LENGUAJE DE ALTO NIVEL DE LOS SIGUIENTES : **JAVA O C#**
- **DEBE MOSTRAR POR LO MENOS EN 2 OCASIONES AVANCES DE SU PRÁCTICA Y ES FORZOSO, Y SIN EXCEPCIÓN, QUE TODOS LOS MIEMBROS DE UN EQUIPO ASISTAN A DICHA REVISIÓN. DICHAS REVISIONES SE LLEVARÁN A CABO LA SEMANA DEL 31 DE OCTUBRE AL 4 DE NOVIEMBRE DEL 2016.**
- CUALQUIER DUDA PUEDE USTED CONSULTAR VÍA CLASE, ASESORÍA PRESENCIAL EN EL CUBÍCULO, O BIEN POR CORREO ELECTRÓNICO.



OBSERVACIONES:

LA REVISIÓN SERÁ EN DIVERSAS VERTIENTES. PUEDE SER AL MOMENTO DE SOLICITAR LA CARPETA DE EVIDENCIAS, O BIEN PUEDE SER AL SOLICITAR LA EXPOSICIÓN DE LA TAREA EN LA CLASE. TAMBIÉN PUEDE SER POR SOLICITUD EXPRESA DEL INTERESADO A PARTICIPAR EN CLASE EXPONIENDO BREVEMENTE SU ACTIVIDAD.

AQUELLAS ACTIVIDADES EN FORMATO DIGITAL SE DEBERÁN TENER SIEMPRE, Y EN TODO MOMENTO A LA MANO EN UNA MEMORIA USB.

ÉSTAS ACTIVIDADES PODRÁN SER SOLICITADAS EN LA CLASE, O BIEN PARA SU ENVÍO A UNA CUENTA DE CORREO.

ESTAS ACTIVIDADES DEBEN ESTAR LISTAS E INTEGRADAS A LA CARPETA DE EVIDENCIAS PARA LA FECHA INDICADA AL FINAL DE ÉSTE DOCUMENTO.

UNA VEZ REVISADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA SIGUIENTE NOMENCLATURA.

ADEMÁS DE ESTA FORMA IRA INTEGRANDO SU CARPETA ELECTRÓNICA DE EVIDENCIAS, QUE SERÁ ENTREGADA AL FINAL DEL SEMESTRE.

POR ÚLTIMO, SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON ESTA NOMENCLATURA NO SERÁN REVISADAS NI TOMADAS EN CUENTA. POR FAVOR GESTIONE SU TIEMPO APROPIADAMENTE.

LA NOMENCLATURA ES :

AAAA-MM-DD_MATERIA_DOCUMENTO_NOCTROL_APELLIDOS_NOMBRE_SEM.PDF

(NOTA : *** TODO EN MAYÚSCULA ***)

DONDE :

AAAA : AÑO
MM : MES
DD : DÍA
MATERIA : TIC, LAII, LI
DOCUMENTO : A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC.
(CAMBIANDO EL NÚMERO CONSECUTIVO POR EL QUE CORRESPONDA)
NOCTROL : SU NOMBRE
APELLIDOS : SUS APELLIDOS
NOMBRE : SU NOMBRE
SEM : SU SEMESTRE ENE-JUN O AGO-DIC

EJEMPLO :

2016-11-13_LAII_P2_99999999_PEREZ_PEREZ_JUAN_AGO-DIC16.PDF

FECHA DE ENTREGA: VÍA EMAIL, EL VIERNES 18 DE NOVIEMBRE DEL 2016, HASTA LAS 20:00 HORAS. DESPUÉS DE ESTA HORA, LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ PARA SU EVALUACIÓN.

POR FAVOR ANEXE A SU ARCHIVO .PDF DE EVIDENCIAS, ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO. POR ÚLTIMO NO OLVIDE ASISTIR A CLASE EL DÍA DE ESTA ENTREGA.

