



## **LENGUAJES Y AUTOMATAS II**

### **ANALIZADOR LEXICO**

**DOCENTE:**

**ISC.RICARDO GONZÁLEZ GONZÁLEZ**

**ACTIVIDAD 2**

**PRESENTAN:**

**CASTRO GONZALEZ RICARDO ISAAC  
CERVANTES VALADEZ ARCADIO  
HERNANDEZ ALTAMIRA LUIS FELIPE  
PANTOJA VALLE LAURA JAZMIN**

**FECHA: 12-MARZO-2018**

**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO  
en Celaya

## DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **Solicitud de Actividades**

Celaya, Gto., 2018-02-23

### LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

#### ACTIVIDAD 2 (VALOR 70 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTES ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA GUÍA TUTORIAL, Y LA RÚBRICA DE EVALUACIÓN.

1. TOMANDO EN CUENTA TODAS LAS INDICACIONES Y EXPLICACIONES EN CLASE Y UNA VEZ DEFINIDA LA GRAMÁTICA Y EL LENGUAJE PROTOTIPO A IMPLEMENTAR, COMO EQUIPO INVESTIGUEN, DISEÑEN E IMPLEMENTEN LA PRIMERA FASE O ETAPA DE UN ANALIZADOR LÉXICO.
2. LA ACTIVIDAD COMO EQUIPO DEBERÁ COMENZAR CON LA INVESTIGACIÓN Y FUNDAMENTACIÓN DE LOS SIGUIENTES TEMAS.
  - A. INVESTIGAR. ¿ QUÉ ES UN ANÁLISIS LÉXICO APLICADO A LA VALORACIÓN DE UN LENGUAJE ?
  - B. INVERTIGAR. ¿ EN QUÉ CONSISTE UN ANÁLISIS LÉXICO Y QUÉ LO CARACTERIZA?
  - C. IDENTIFICAR. ¿ QUÉ CASOS DE ESTUDIOS SON LOS IMPORTANTES A CONSIDERAR EN EL ANÁLISIS LÉXICO ?
  - D. INVESTIGAR Y PROPOSICIÓN. ¿ QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANÁLISIS LÉXICO ?
  - E. INVESTIGAR. ¿ CÓMO IMPLEMENTAR UN ANÁLISIS LÉXICO ?

NOTA : ESTOS TEMAS DEBEN SER EL RESULTADO DE UNA INVESTIGACIÓN Y ANÁLISIS COMO EQUIPO, Y NO UNA TRANSCRIPCIÓN DE TEMAS AISLADOS, PUES EN TODO MOMENTO LAS FUENTES DE CONSULTA DEBEN SER LA BASE DEL DESARROLLO DE ESTE PUNTO Y SUS APARTADOS.



60 Aniversario  
TecNM  
en Celaya  
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57. Comutador:(461)6117575. Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)





3. COMO EQUIPO Y DERIVADO DEL ANÁLISIS ANTERIOR SE DEBEN PROponER LOS ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO DE ANALIZADOR LÉXICO. ES DECIR, MANEJO Y OPERACIONES CON ARCHIVOS, TABLA DE SÍMBOLOS, DISEÑO DE AUTÓMATAS, PILA DE ERRORES, ETC.

CONCRETAMENTE EN ESTE PUNTO DEBERÁN COMO EQUIPO, REDACTAR Y DETALLAR TODOS LOS ELEMENTOS QUE SE USARÁN PARA LA IMPLEMENTACIÓN DEL ANALIZADOR LÉXICO.

4. PARA EL INCISO C DEL PUNTO 2 ANTERIOR, SE DEBERÁN CREAR PROGRAMAS DE MÍNIMO 25 LÍNEAS (SIN CONSIDERAR LOS COMENTARIOS) CADA UNO, EN LOS CUALES SE CODIFIQUE EN EL LENGUAJE PROTOTIPO, INSTRUCCIONES CON LÓGICA QUE EJEMPLIFIQUEN LOS ERRORES QUE EN CADA CASO DE ESTUDIO SE PROPONGAN.

TALES PROGRAMAS DEBEN ESTAR PERFECTAMENTE DOCUMENTADOS Y CORRELACIONADOS CON LOS CASOS DE ESTUDIO CORRESPONDIENTES.

5. CARACTERÍSTICAS QUE LA ACTIVIDAD 2 DEBE POSEER PARA CONSIDERARSE COMPLETA.

- A. EL FUNDAMENTO DE LA GRAMÁTICA A UTILIZAR, ASÍ COMO SU DEFINICIÓN FORMAL Y EL ALFABETO A UTILIZAR.
- B. LA CARECTERIZACIÓN DEL LENGUAJE PROTOTIPO, ES DECIR SU DESCRIPCIÓN MEDIANTE NOTACIÓN BNF. SE DEBEN INCLUIR ADEMÁS LOS SÍMBOLOS ESPECIALES COMO DELIMITADORES IMPLÍCITOS Y EXPLÍCITOS, OPERADORES LÓGICOS, RELACIONALES Y ARITMÉTICOS.
- C. CATEGORIZACIÓN E IDENTIFICACIÓN POR ID, DE CADA UNO LOS ELEMENTOS QUE EL LENGUAJE PROTOTIPO PROPONE.
- D. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR LÉXICO, ASI COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.
- E. PREPARACIÓN DE LOS PROGRAMAS SUFICIENTES, ESCRITOS EN EL LENGUAJE PROTOTIPO, QUE APOYEN LA COMPROBACIÓN DE CADA CASO DE ESTUDIO. TALES PROGRAMAS DEBERÁN ESTAR COMPLETAMENTE DOCUMENTADOS.
- F. GENERACIÓN DE UN CATÁLOGO DE ERRORES, CORRELACIONADO A LOS CASOS DE USO PROPUESTOS.
- G. DISEÑO DE UNA SOLUCIÓN MODELADA EN OBJETOS, APOYADA EN DIAGRAMAS UML QUE DESCRIBAN LA ARQUITECTURA PROPUESTA PARA EL PROTOTIPO DEL ANALIZADOR LÉXICO.
- H. PROPUESTA Y MODELADO DE LAS PROPIEDADES Y COMPORTAMIENTOS DE UNA TABLA DE SÍMBOLOS, ASÍ COMO LA JUSTIFICACIÓN DE CADA UNA DE LAS COLUMNAS QUE LA INTEGREN.



60  
Aniversario  
TecNM  
en Celaya  
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57. Comutador: (461) 6117575. Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**I. PROPUESTA Y MODELADO DE LAS PROPIEDADES Y COMPORTAMIENTOS DE UNA PILA DE ERRORES.**

**J. MODELADO DE LOS PROCESOS DE APERTURA Y LECTURA DEL ARCHIVO DE CÓDIGO FUENTE, ASÍ COMO DEL PROCESO DE TOKENIZACIÓN (QUE NO DEBERÁ IMPLEMENTARSE A PARTIR DE FUNCIONALIDADES DE BIBLIOTECAS DE TERCEROS).**

**K. DISEÑO, PRUEBAS, MODELADO E IMPLEMENTACIÓN DE LOS AUTOMÁTAS SUFICIENTES Y NECESARIOS PARA LA CATEGORIZACIÓN DE CADA UNO DE LOS TOKENS GENERADOS.**

**L. MODELADO E IMPLEMENTACIÓN DE UN PROCESO DE :**

**LECTURA DE CÓDIGO FUENTE => TOKENIZACIÓN => CATEGORIZACIÓN DE LOS TOKENS => CONTRUCCIÓN Y LLENADO DE LA TABLA DE SIMBOLOS => MANEJO Y DESPLIEGUE DE ERRORES => DESPLIEGUE DE LA TABLA DE SÍMBOLOS RESULTANTE.**

**M. GENERACIÓN DE UN CALENDARIO DE ACTIVIDADES PLANIFICADAS VS. ACTIVIDADES REALIZADAS.**

**N. GENERACIÓN DE UNA BITÁCORA DE INCIDENCIAS.**

**O. TODAS LAS EVIDENCIAS GENERADAS Y REUNIDAS DEBERÁN INTEGRARSE AL UN ARCHIVO PDF, NOMBRADO COMO SE INDICA MÁS ADELANTE.**

**ESTAS EVIDENCIAS PODRÁN ELABORARSE CON HERRAMIENTAS ELECTRÓNICAS, COMO PROCESADOR DE TEXTO, DE IMÁGENES, HOJAS DE CÁLCULO, ETC.**

**P. EL NÚCLEO DE CADA ALGORITMO CODIFICADO TAMBIÉN DEBERÁ FORMAR PARTE DEL ARCHIVO DE EVIDENCIAS.**

**NOTAS GENERAL DE LA ACTIVIDAD:**

- SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD, LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL PUNTO 3 INCISO i (Trabajo en equipo).**
- SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL PUNTO 6, (Evidencias tipo a).**



**60**  
Aniversario  
TecNM  
en Celaya  
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57. Comutador:(461)6117575. Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO  
en Celaya

**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 (FÍSICAMENTE) A LA FECHA DE ENTREGA INDICADA AL FINAL DE ÉSTE DOCUMENTO.
- ✓ CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO.
- ✓ UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA SIGUIENTE NOMENCLATURA.
- ✓ SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- ✓ CON ESTA ACTIVIDAD, USTED DEBERÁ IR INTEGRANDO SUS CARPETAS FÍSICA Y ELECTRÓNICA DE EVIDENCIAS, Y AL FINAL DEL SEMESTRE EN UN DISCO COMPACTO HARÁ ENTREGA DE SU CARPETA ELECTRÓNICA DE EVIDENCIAS.
- ✓ PARA TENER DERECHO A LA REVISIÓN Y EVALUACIÓN DE SUS ACTIVIDADES, DEBE REGISTRAR SU ASISTENCIA A CLASE, EL DÍA SEÑALADO PARA LA ENTREGA DE LA MISMA.
- ✓ FALTAR A CLASE EL DÍA DE LA ENTREGA, ANULA LA REVISIÓN DE SUS EVIDENCIAS.
- ✓ POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA.
- ✓ AÚN PARA TRABAJOS EN EQUIPO APlican TODAS LAS MISMAS OBSERVACIONES ANTERIORES.



60 Aniversario  
TecNM  
en Celaya  
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57, Comunitador (461) 6117575, Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**ANAB**  
ACCREDITED  
MANAGEMENT SYSTEMS  
CERTIFICATION BODY

**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO  
en Celaya

**LA NOMENCLATURA SOLICITADA ES :**

**AAAA-MM-DD\_MATERIA\_DOCUMENTO\_EQUIPO\_NOCTROL\_APELLIDOS\_NOMBRE\_SEM.PDF**

**( NOTA : \*\*\* TODO EN MAYÚSCULA \*\*\* )**

**DONDE :**

AAAA	: AÑO
MM	: MES
DD	: DÍA
MATERIA	: SO, TSO, LAII, LI
DOCUMENTO	: A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUТИVO POR EL QUE CORRESPONDA)
EQUIPO	: NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR.
NOCTROL	: SU NÚMERO DE CONTROL
APELLIDOS	: SUS APELLIDOS
NOMBRE	: SU NOMBRE
SEM	: EL PERÍODO SEMESTRAL EN CURSO: ENE-JUN / AGO-DIC

**EJEMPLO :**

**2018-03-12\_LAII\_A2\_LEXICO\_EQUIPO\_99\_9999999\_PEREZ\_PEREZ\_JUAN\_ENE-JUN18.PDF**

**FECHA DE ENTREGA:**

**VÍA CORREO ELECTRÓNICO, EL LUNES 12 DE MARZO DEL 2018, CON HORA LÍMITE DE ENTREGA  
HASTA LAS 14:00 HORAS (2 DE LA TARDE).**

**DESPUÉS DE ESTA HORA, LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO  
CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.**

**MUY IMPORTANTE:**

**POR FAVOR ANEXE A SU ARCHIVO .PDF DE EVIDENCIAS, ESTA SOLICITUD DE ACTIVIDADES CON  
TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.**

**CALENDARIO 2017-2018 - EVALUACIÓN 2**

FEBRERO						
L	M	W	J	V	S	D
				3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

MARZO						
L	M	W	J	V	S	D
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

EVALUACIÓN FORMATIVA DE 1<sup>RA</sup> OPORTUNIDAD (PARCIALES)



**60**  
Aniversario  
TecNM  
en Celaya  
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010  
Celaya, Gto. AP 57. Comutador:(461)6117575. Correo electrónico: [lince@itcelaya.edu.mx](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



## **CONTENIDO.**

<b>¿QUE ES UN ANÁLISIS LÉXICO APLICADO A LA VALORACIÓN DE UN LENGUAJE?</b>	7
<b>¿EN QUÉ CONSISTE UN ANÁLISIS LÉXICO Y QUE LO CARACTERIZA?</b>	8
Token, Patrón, Lexema	9
Atributos para los tokens	10
<b>¿QUE CASOS DE ESTUDIOS SON LOS MÁS IMPORTANTES A CONSIDERAR EN EL ANÁLISIS LÉXICO</b>	11
Catálogo de errores	12
<b>¿QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANALISIS LEXICO?</b>	13
Casos de uso	14
Respuestas ante el error	15
<b>ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO</b>	16
Pila, Pila de errores	17
Funciones de la pila	17
Listas enlazadas	18
Tabla de símbolos	19
Diagrama de clases	20
Interfaz a utilizar	21
Definición de gramática y producciones.	22
BNF (Formato Backus-Naur)	23
Sintaxis	24
Ventajas y Desventajas	25
Autómata generado en JFlap	26
Código Fuente (Evidencia de la programación)	27
Creación de Programas	43
Pruebas generadas sobre los códigos anteriormente descritos	48
<b>CALENDARIO DE ACTIVIDADES REALIZADAS</b>	54
<b>REPORTE DE INCIDENCIAS</b>	55

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTES ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA [GUÍA TUTORIAL](#), Y LA [RÚBRICA DE EVALUACIÓN](#).

1. TOMANDO EN CUENTA TODAS LAS INDICACIONES Y EXPLICACIONES EN CLASE Y UNA VEZ DEFINIDA LA GRAMÁTICA Y EL LENGUAJE PROTOTIPO A IMPLEMENTAR, COMO EQUIPO INVESTIGUEN, DISEÑEN E IMPLEMENTEN **LA PRIMERA FASE O ETAPA DE UN ANALIZADOR LÉXICO.**
2. LA ACTIVIDAD COMO EQUIPO DEBERÁ COMENZAR CON LA INVESTIGACIÓN Y FUNDAMENTACIÓN DE LOS SIGUIENTES TEMAS.
  - A) **INVESTIGAR.** ¿QUE ES UN ANÁLISIS LÉXICO APLICADO A LA VALORACIÓN DE UN LENGUAJE?

El analizador léxico se encarga de realizar la lectura de un flujo de caracteres que componen el programa fuente y agruparlos en secuencia significativa.

Algunas de las acciones que puede realizar un analizador léxico son las siguientes: comprobar alguna restricción adicional (**ejemplo: el valor de alguna literal entera no esté dentro de un rango**), preparar los atributos del componente y emitir u omitir dicho componente.

La especificación de un analizador léxico debe incluir por cada categoría del lenguaje el conjunto de atributos y las acciones asociadas.

Categoría	Expresión regular	Acciones	Atributos
Entero	[0-9]	<ul style="list-style-type: none"><li>• Calcular el valor</li><li>• Comprobar el rango</li></ul>	Valor
Real	[0-9]+\.[0-9]	<ul style="list-style-type: none"><li>• Calcular valor</li><li>• Comprobar el rango</li></ul>	Valor
Identificador	[aA-zZ][aA-zZ-0-9]	<ul style="list-style-type: none"><li>• Copiar lexema</li></ul>	Lexema
Asignación	=	Emitir	

Tabla.2.a.1

## Esquema de procesamiento.

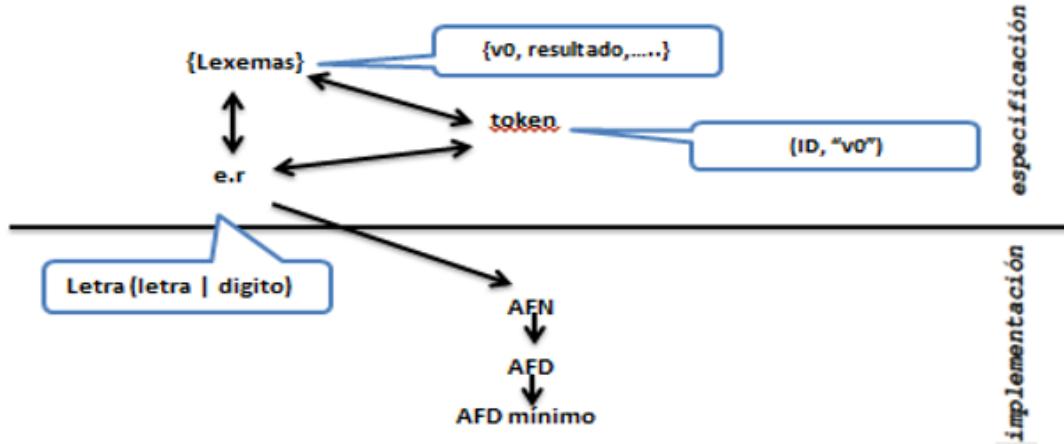


Imagen. 2. a.1

Algunos de los errores que son detectados en el nivel léxico son aquellos que solo pueden encontrarse al ejecutar las acciones asociadas a dichas categorías. Un ejemplo de esto, sería encontrar un número fuera de rango. Es posible encontrar una expresión regular para esto, por ejemplo los enteros de 32 bits. Esta expresión resulta sumamente complicada y se puede realizar la comprobación mediante una simple comparación.

Por ejemplo, un programa fuente contiene la siguiente instrucción asignada:

**Ubicación = inicio + velocidad \* 60**

Los lexemas en este ejemplo se pueden agrupar en los siguientes lexemas y mapearlos en los siguientes tokens que se pasan posteriormente al analizador sintáctico:

1. **Ubicación:** Es un lexema que se asigna a un token **{id. 1}**, id es el símbolo abstracto que representa la palabra que lo identifica y 1 apunta a la entrada en la tabla de símbolos.
2. **Símbolo de asignación (=):** De igual manera, es un lexema que se asigna al token **(=)**. Este token no necesita un valor-atributo, por estrategia de notación se optó por usar el mismo lexema como nombre para el símbolo abstracto.
3. **Inicio:** Es un lexema que se asigna a un token **{id, 2}**, de igual forma el 2 apunta a la entrada en la tabla de símbolos.
4. **Operación (+):** Es un lexema que se asigna a un token.
5. **Velocidad:** Es un lexema que se asigna a un token **{id, 3}**, de igual forma el 3 apunta a la entrada en la tabla de símbolos.
6. **Operación (\*):** Es un lexema que se asigna a un token.

7. 60: Es un lexema que se asigna al token.

Una de las tareas del analizador léxico es omitir los espacios en blanco que separan a los lexemas descritos. [1]

{Id, 1} {=} {Id, 2} {+} {Id, 3} {\*} {60}

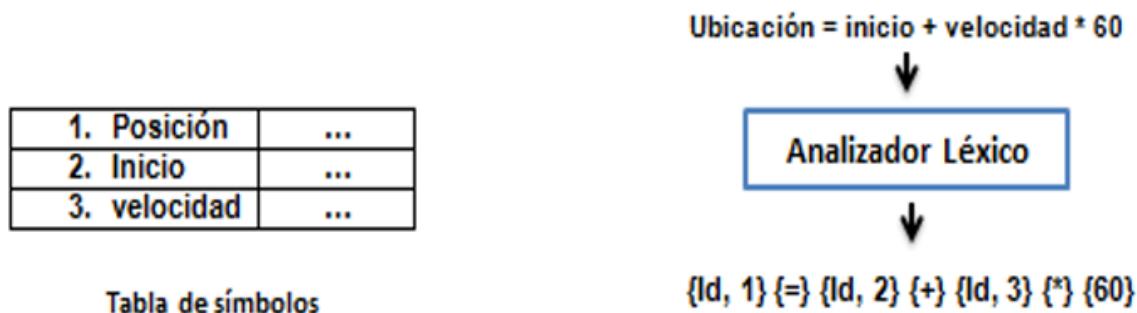


Imagen. 2. a.2

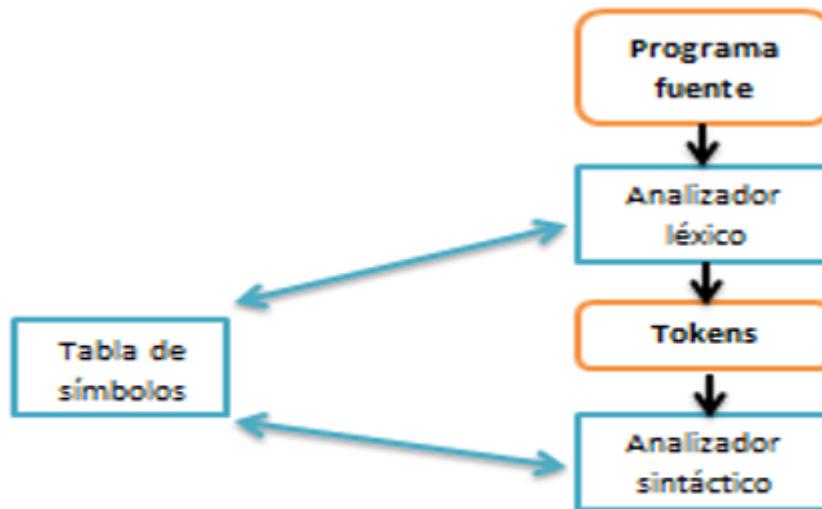


Imagen. 2. a.3

- [1] [Webdiis.unizar.es](http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2.analisislexico.pdf)  
<http://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2.analisislexico.pdf>

## B) INVESTIGAR. ¿EN QUÉ CONSISTE UN ANÁLISIS LÉXICO Y QUE LO CARACTERIZA?

Es la primera fase del compilador, la función principal es leer la secuencia generada por los caracteres del programa fuente, carácter a carácter, de esta manera se genera una salida de componentes léxicos que serán utilizados en el analizador sintáctico.

Un programa fuente es una serie de símbolos (**letras, caracteres especiales: +,\*,!)**. Con estos símbolos se pueden representar las construcciones del lenguaje como son las **variables, palabras reservadas**, etc. Es necesario que el compilador o el traductor identifiquen los diferentes significados de las construcciones generadas.

Posteriormente el analizador sintáctico da la orden al analizador léxico para que este agrupe los caracteres para dar un significado propio llamados tokens.

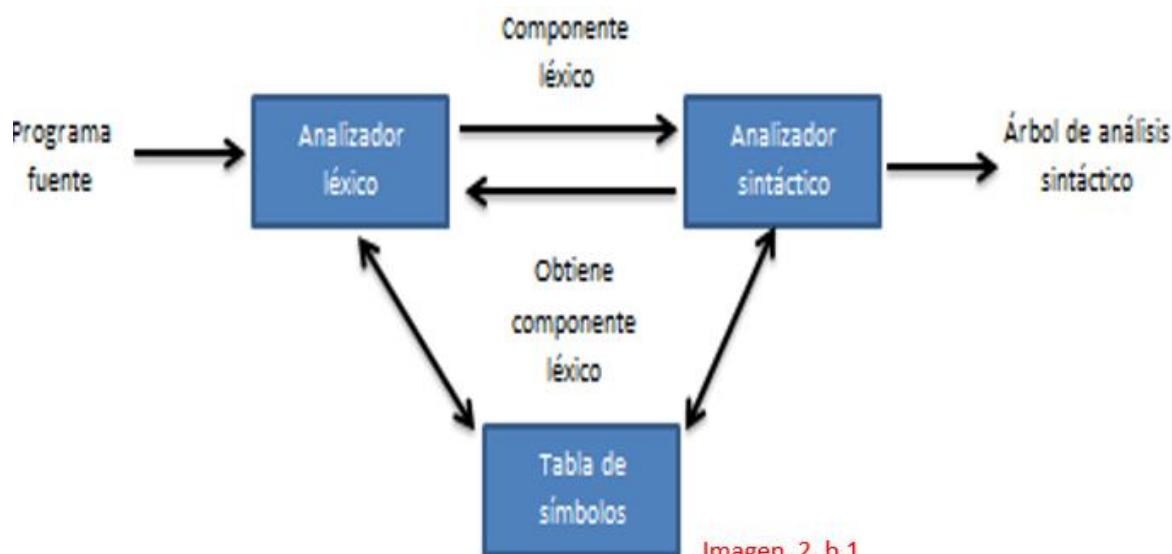


Imagen. 2. b.1

Algunas otras responsabilidades del analizador léxico son:

- Eliminar los comentarios o espacios en blanco.
- Lectura de caracteres.
- Mostrar mensajes de error.

Otra de las funciones secundarias es relacionar los mensajes de error del compilador con el programa fuente. Por ejemplo, el analizador léxico puede tener localizado el número de caracteres de la nueva línea que analiza, de esta manera puede relacionar el número de línea con un mensaje de error. En la fase de análisis léxico, los términos de componente léxico conocidos como tokens, patrón y lexema, son empleados con un fin específico.

**Componente léxico o token:** Es una secuencia lógica de caracteres relativos a una categoría como las siguientes:

- Identificadores.
- Palabras reservadas.
- Operadores.

El token representa un patrón de caracteres que el analizador léxico debe reconocer.

Este consiste en un nombre del token correspondiente y un valor de atributo opcional.

Para cada lexema, el analizador léxico genera una salida un token de la forma:

**{nombre-token, valor-atributo}**

**Componente (nombre-token):** es un símbolo abstracto que es utilizado durante la fase del análisis sintáctico y el segundo componente (valor-atributo) apunta a una entrada en la tabla de símbolos que será utilizada para este token.

Los nombres de los tokens son símbolos de entrada que se procesan en el analizador sintáctico.

**Patrón:** Son las reglas que generan las secuencias de caracteres que se pueden representar en un determinado componente léxico.

Para los identificadores, el patrón es una estructura más compleja que se relaciona mediante muchas cadenas.

- **Nota:** “[¿Qué es una expresión regular?](#)

**Frecuentemente se les llama patrones, ya que son expresiones que describen a un conjunto de cadenas.”**

**[2]**

**Lexema:** Son las cadenas de caracteres que concuerdan con el patrón que describe un componente léxico, a su vez el analizador léxico identifica como una instancia de un token.

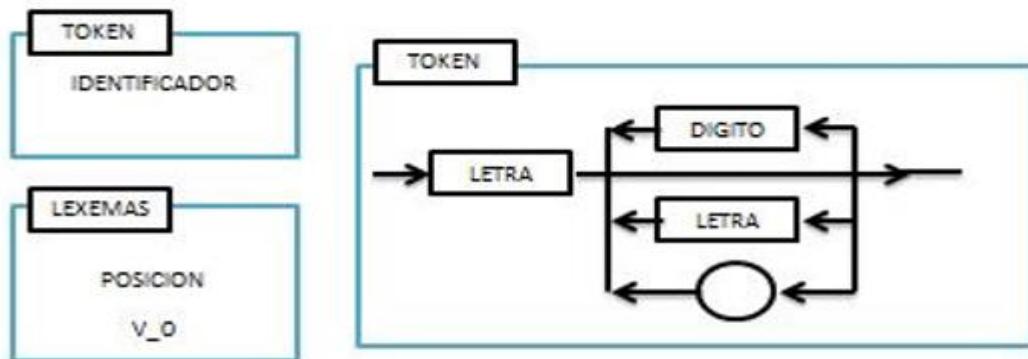


Imagen.2.b.2

Un analizador recoge información de los componentes léxicos. Los tokens influyen en las decisiones del análisis sintáctico y los atributos en la traducción de los tokens. [\[3\]](#)

Ejemplo de tokens comunes, patrones y lexemas de ejemplo:

TOKEN	DESCRIPCION INFORMAL	LEXEMA DE EJEMPLO
If	Caracteres i, f	If
else	Caracteres e, l, s, e	else
Numero	Cualquier constante numérica	5.3474 o 500
Id	Letras sugeridas por letras y/o dígitos	D1, valor

Tabla.2.b.1

1. Se genera un token para cada palabra clave. El patrón para una palabra clave.
2. Los tokens se generan para los operadores, los podemos encontrar de manera individual o en casos como el token de comparación.
3. Uno o más token que representarán a las constantes.
4. Un token que representa a los identificadores.
5. Un token para cada signo de puntuación que sea generado.

#### Atributos para los tokens

En algunas ocasiones, más de un lexema puede llegar a **coincidir con un patrón**, la función del analizador léxico es proporcionar a las subsiguientes fases del compilador la suficiente información sobre el lexema que se especifica, es muy importante para el generador de código saber el lexema que se encontró en el programa fuente. En muchos de los casos el analizador léxico devuelve al analizador sintáctico no solo el nombre que corresponde al token, sino un valor de **atributo que describa al lexema**.

En general, el análisis léxico es un análisis que se realiza a nivel de caracteres, su objetivo es reconocer los componentes léxicos o tokens.

- [3] [Paginasprodigy.com](http://www.paginasprodigy.com/edserna/cursos/compilador/notas/Notas.pdf)  
<http://www.paginasprodigy.com/edserna/cursos/compilador/notas/Notas.pdf>

**C) IDENTIFICAR. ¿QUE CASOS DE ESTUDIOS SON LOS MÁS IMPORTANTES A CONSIDERAR EN EL ANÁLISIS LÉXICO?**

- Uso de caracteres que no estén **dentro del alfabeto**.
- **Eliminar** los comentarios que contenga el código fuente y espacios (caracteres de espacio, nueva línea, tabulador y otros caracteres que se utilicen para separar los tokens).
- **Lectura** de cada carácter que se encuentre en el código fuente.
- **Agrupar** los caracteres en tokens.
- Llenado de la tabla de símbolos, con respecto a cada token. **[4][5]**

**Generación de catálogo de errores, correlacionando a los casos de uso propuestos.**

El siguiente catálogo de errores este hecho en base a los casos de uso descritos en el punto 2. Inciso C

Tipo de error	Código	Descripción
léxico	100	La cadena contiene uno o varios caracteres inválidos dentro del lenguaje
léxico	101	Se omitió un carácter en la cadena
léxico	103	La cadena no se reconoce dentro del lenguaje

- [4]** Compiladoresasignatura.blogspot.mx. (2018). *UNIDAD 7: MANEJO DE ERRORES.* [online] Available at: <http://compiladoresasignatura.blogspot.mx/2011/05/unidad-vii-manejo-de-errores.html>
- [5]** Informatica.uv.es. (2018). *Citar un sitio web - Cite This For Me.* [online] Available at: <http://informatica.uv.es/docencia/iiguia/asignatu/2000/PL/2008/tema2.pdf>

## D) INVESTIGAR Y PROPONER. ¿QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANALISIS LEXICO?

Los **errores léxicos** se detectan cuando el analizador léxico intenta reconocer componentes léxicos (tokens) y esta cadena no encaja con ningún patrón por ejemplo situaciones en las que se usa un carácter inválido que no pertenece al vocabulario del lenguaje o bien al escribir mal un identificador, palabra reservada o algún operador pues los errores léxicos se deben a descuidos del programador.

Entre los errores más comunes que se encuentran en la fase de análisis léxico se han clasificado por los siguientes casos de uso.

### Casos de uso

**Caso 1:** Nombres ilegales de identificadores: un nombre contiene caracteres inválidos.

**Caso 2:** Números incorrectos: Un número contiene caracteres inválidos o no está formado correctamente, por ejemplo 5,1 en vez de 5.1

**Caso 3:** Errores de ortografía en palabras reservadas: caracteres omitidos, adicionales o cambiados de sitio, por ejemplo, la palabra while en vez de hwile.

**Caso 4:** Fin de archivo: se detecta un fin de archivo a la mitad de un componente léxico.

Gran parte de la detección de errores o problemas en el proceso de compilación de un lenguaje se centra en las fases de análisis léxico y sintáctico. La razón es que muchos de los errores son de naturaleza sintáctica o se manifiestan cuando la cadena de componentes léxicos que proviene del analizador léxico desobedece las reglas gramaticales que definen la especificación.

La tarea que realiza el analizador léxico es un caso muy especial de coincidencia de patrones por lo cual se necesitan los métodos necesarios para poner llevar a cabo la especificación y el reconocimiento de los mismos, estos métodos son principalmente expresiones regulares y autómatas finitos, al hablar de un analizador léxico también es importante recalcar que al ser parte del traductor se maneja una entrada de código fuente por lo cual a menudo se involucra un importante gasto de tiempo por lo que el analizador léxico debe de funcionar lo más eficientemente posible. El analizador léxico debe **devolver el componente léxico** de un identificador y dejar a otra fase se ocupe de los errores.

El traductor debe de **recuperarse de cada error** lo suficientemente rápido para poder detectar errores subsiguientes, así como reportar clara y exactamente la presencia de estos. Si un analizador léxico no pudiera continuar porque ninguno de los patrones no coincidiera con un prefijo de entrada, de ser así tal vez la recuperación más sencilla sea la recuperación

conocida como “**Modo de pánico**” en el cual se borran los caracteres sucesivos de la entrada hasta que el analizador léxico pueda encontrar un componente léxico bien formado.

Una buena estrategia para la recuperación de errores léxicos es:

Si en el momento de detectar el error ya hemos pasado por algún estado final, ejecutamos la acción correspondiente al último estado final visitado con el lexema formado hasta que salimos de él, el resto de caracteres leídos se devuelven al flujo de entrada y se vuelve al estado inicial.

Si no hemos pasado por ningún estado final, advertimos que el carácter encontrado no se esperaba, lo eliminamos y proseguimos con el análisis.

**Las respuestas ante el error pueden ser:**

**Inaceptables:** Provocadas por fallos del traductor, entrada en lazos infinitos, producir resultados erróneos, y detectar sólo el primer error y detenerse.

**Aceptables:** Evitar la avalancha de errores (**mala recuperación**) y, aunque más complejo, informar y reparar el error de forma automática. La conducta de un Analizador de Léxico es el de un **Autómata finito** o “scanner”.

**Detección del error:** El analizador de Léxico detecta un error cuando no existe transición desde el estado que se encuentra con el símbolo de la entrada. El símbolo en la entrada no es el esperado. **[4]**

- [4]** [Informatica.uv.es  
http://informatica.uv.es/docencia/iigua/asignatu/2000/PL/2008/tema2.pdf](http://informatica.uv.es/docencia/iigua/asignatu/2000/PL/2008/tema2.pdf)

### 3. COMO EQUIPO Y DERIVADO DEL ANÁLISIS ANTERIOR SE DEBEN PROponER LOS ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO DE ANALIZADOR LÉXICO. ES DECIR, MANEJO Y OPERACIONES CON ARCHIVOS, TABLA DE SÍMBOLOS, DISEÑO DE AUTÓMATAS, PILA DE ERRORES, ETC.

**Pila:** Es una estructura de datos, en el cual todos los datos son del mismo tipo.

**Pila de Error:** Es una pila de tipo de dato Error.

Una de las propiedades de las pilas y que la diferencia del resto de las estructuras de datos es que los elementos (o datos) son ingresados por un extremo de la estructura, para poder eliminar u obtener un dato de esta podemos usar **LIFO (Last In First Out)**, es decir, el primer elemento ingresado es el último en salir.

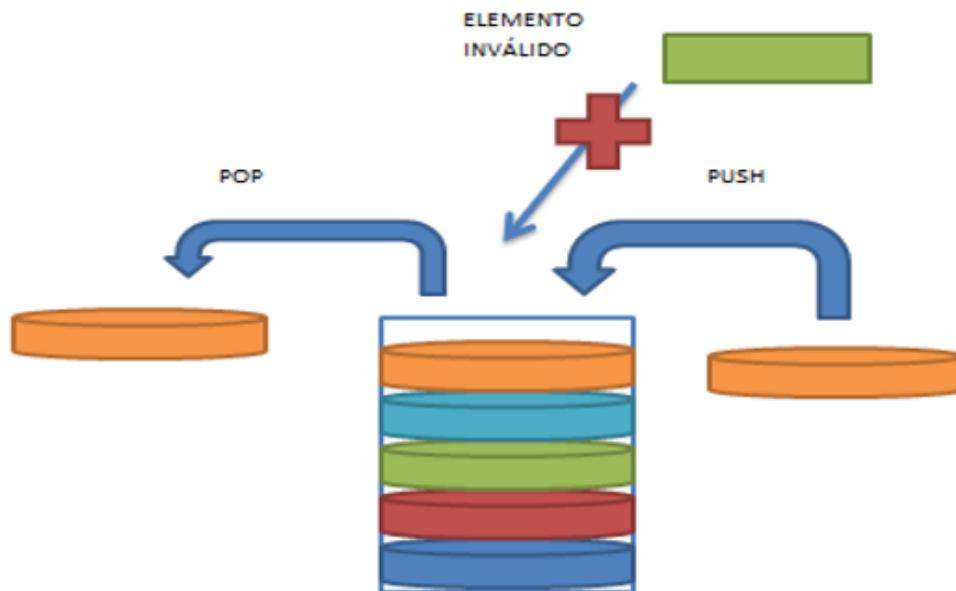


Imagen. I.1

Las funciones propias de las pilas son las siguientes:

- **Push:** Ingresa un elemento a la pila.
- **Pop:** Elimina un elemento de la pila.
- **Top:** Devuelve el elemento que se encuentra en la cima de la pila pero sin eliminarlo.
- **isEmpty:** Permite verificar si la pila está vacía, si este es el caso, la operación regresa el valor de verdadero y en caso contrario regresa falso.
- **Size:** Regresa el número de elementos que se encuentran en la pila.

En una pila no está permitido el acceso aleatorio, ya que la principal propiedad de la pila es el ingreso, obtención y retirada de datos por un solo extremo de la pila.

Una pila puede ser implementada de dos maneras:

- A través de un arreglo
- Por el uso de nodos enlazados por punteros.

Lo ideal a implementar en las pilas, es a través de unas listas enlazadas por punteros. La idea de esto es generar nodos de manera dinámica a medida que se vayan necesitando, y evitar gastos de memoria que quizás no será utilizada.

Cuando se implementa una pila por medio de listas enlazadas, cada elemento de la pila se almacena en uno de los nodos de la lista.

### Representación de la pila con listas enlazadas.

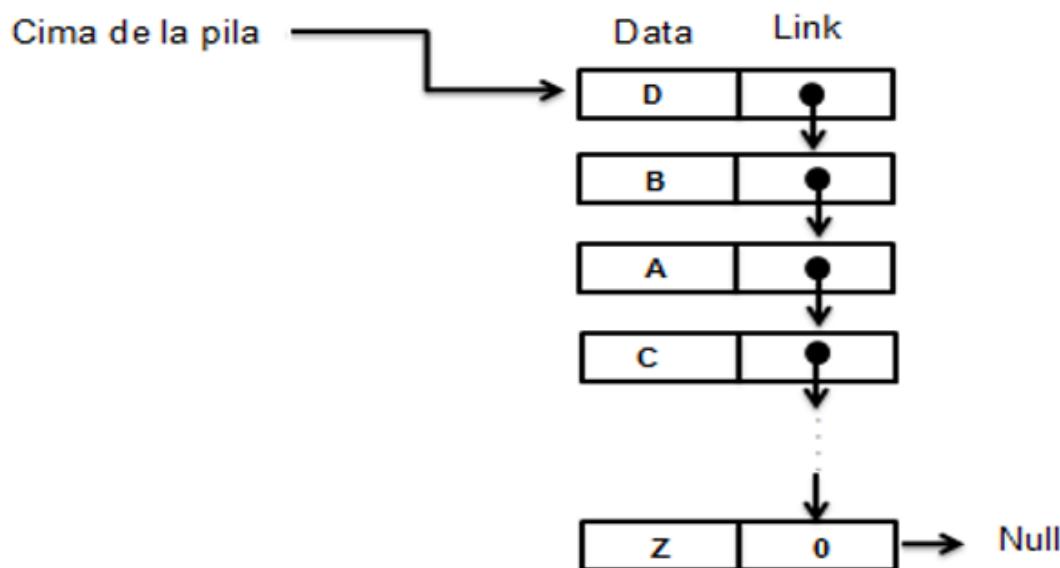


Imagen. I.2

La manera de cómo se enlazan los nodos de la pila, es colocando una liga al nodo que representa la cima de la pila, a su vez este nodo tendrá una referencia al elemento que se encuentra por debajo de este y así sucesivamente hasta llegar al elemento que se encuentra en la parte inferior de la pila, este último elemento tendrá el valor de null en el campo que referencia al elemento de abajo.

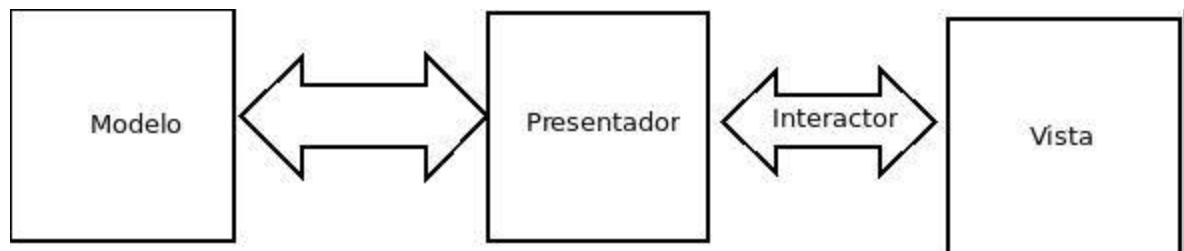
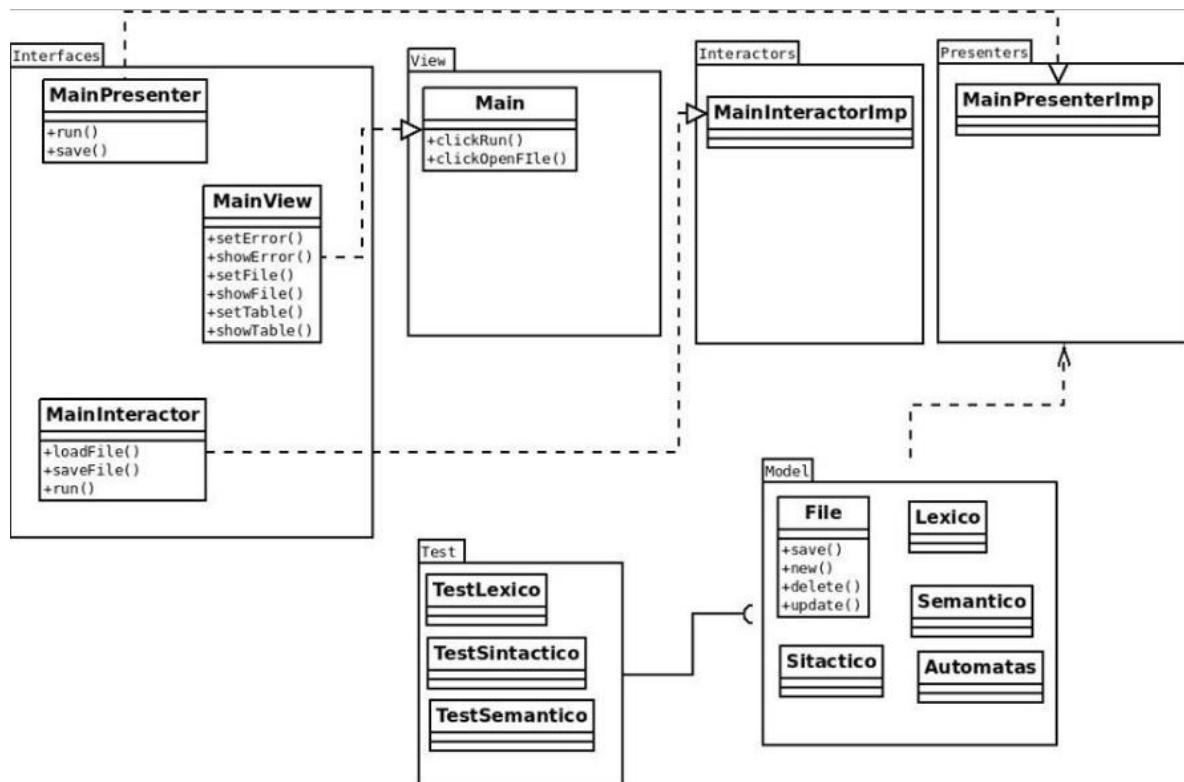
## Tabla de símbolos.

<b>AGRUPACIÓN O TIPO.</b>	<b>IDENTIFICADOR. (ID).</b>	<b>DESCRIPCION.</b>
<b>ASIGNACIÓN</b>		
=	100	Este signo es utilizado para dar valores a identificadores.
<b>PALABRAS RESERVADAS</b>		
init	101	Palabra reservada que marca el inicio del código.
String	102	Palabra reservada que es utilizada en los tipos de datos de las variables del programa.
Int	103	Tipo de dato entero.
read	104	Indica que se dará lectura de caracteres desde el teclado.
print	105	Indica que mostrara en pantalla algún mensaje.
while	106	Palabra reservada para realizar repeticiones dentro del código.
if	107	Palabra reservada para realizar condiciones.
else	108	Palabra reservada que hace énfasis en un bloque de instrucciones en caso de no cumplirse la condición.
<b>DELIMITADORES</b>		
;	110	Signo utilizado para indicar el fin de una instrucción.
<b>AGRUPACIONES</b>		
{	115	Inicio de un bloque de instrucciones.

}	116	Final de un bloque de instrucciones
(	117	Agrupación de elementos o también utilizado en las condiciones.
)	118	
<b>OPERADORES DE IGUALDAD</b>		
<	120	Menor que.
>	121	Mayor que.
==	122	Igual.
i	123	Cambio de valor de verdad.
<=	124	Menor igual que.
>=	125	Mayor igual que.
i=	126	Diferentes.
	127	"OR"
<b>OPERADORES ARITMETICOS</b>		
+	130	Signo utilizado para la suma de dos números.
-	131	Signo para indicar resta entre dos números.
*	132	Signo para indicar la multiplicación de dos números.
/	133	Signo para indicar la división entre dos números.
<b>IDENTIFICADORES</b>	140	Representación de forma escrita de diferentes componentes del lenguaje.

## Diagrama de clases

El siguiente diagrama de clases, muestra todas las clases que utilizaremos para el desarrollo del análisis léxico, tanto el nombre de cada clase, como los elementos de cada clase que se utilizan en el desarrollo.



## Interfaz a utilizar



Definición de gramática del lenguaje prototipo.

$$G = \{V_T, V_N, P, S\}$$

$$V_T = \{g, s, t, r, i, n, w, h, f, d, p, s\}$$

$$V_N = \{S_0, S_1, S_2, S_3, S_4 \dots S_{23}\}$$

$$P = \{$$

<b>S<sub>0</sub></b>	->	<b>S S<sub>1</sub></b>
<b>S<sub>1</sub></b>	->	<b>T S<sub>3</sub></b>
<b>S<sub>3</sub></b>	->	<b>R S<sub>4</sub></b>
<b>S<sub>4</sub></b>	->	<b>I S<sub>5</sub></b>
<b>S<sub>5</sub></b>	->	<b>N S<sub>6</sub></b>
<b>S<sub>7</sub></b>	->	<b>G</b>

<b>S<sub>0</sub></b>	->	<b>I S<sub>7</sub></b>
<b>S<sub>7</sub></b>	->	<b>N S<sub>8</sub></b>
<b>S<sub>8</sub></b>	->	<b>T</b>

<b>S<sub>0</sub></b>	->	<b>I S<sub>9</sub></b>
<b>S<sub>9</sub></b>	->	<b>F</b>

<b>S<sub>0</sub></b>	->	<b>E S<sub>10</sub></b>
<b>S<sub>10</sub></b>	->	<b>L S<sub>11</sub></b>
<b>S<sub>11</sub></b>	->	<b>S S<sub>12</sub></b>
<b>S<sub>12</sub></b>	->	<b>E</b>

<b>S<sub>0</sub></b>	->	<b>W S<sub>13</sub></b>
<b>S<sub>13</sub></b>	->	<b>H S<sub>14</sub></b>
<b>S<sub>14</sub></b>	->	<b>I S<sub>15</sub></b>
<b>S<sub>15</sub></b>	->	<b>L S<sub>12</sub></b>

<b>S<sub>0</sub></b>	->	<b>R S<sub>16</sub></b>
<b>S<sub>16</sub></b>	->	<b>E S<sub>17</sub></b>
<b>S<sub>17</sub></b>	->	<b>A S<sub>18</sub></b>
<b>S<sub>18</sub></b>	->	<b>D</b>

<b>S<sub>0</sub></b>	->	<b>P S<sub>19</sub></b>
<b>S<sub>19</sub></b>	->	<b>R S<sub>20</sub></b>
<b>S<sub>20</sub></b>	->	<b>I S<sub>21</sub></b>
<b>S<sub>21</sub></b>	->	<b>N S<sub>22</sub></b>
<b>S<sub>22</sub></b>	->	<b>T S<sub>8</sub></b>

<b>S<sub>0</sub></b>	->	<b>I S<sub>23</sub></b>
<b>S<sub>23</sub></b>	->	<b>N S<sub>24</sub></b>
<b>S<sub>24</sub></b>	->	<b>I S<sub>8</sub></b>

## **BNF (Formato Backus-Naur)**

Es un sistema notacional que sirve para especificar tipos de datos o categorías sintácticas, de igual manera sirve para especificar la sintaxis de los lenguajes de programación mediante las ya conocidas reglas de producción.

### **Un poco de historia...**

La idea de transcribir la estructura del lenguaje con reglas de reescritura se remonta al trabajo del indio Panini (460 a.C) que la utilizó en su descripción de la estructura de palabras del idioma sánscrito. Incluso se ha sugerido renombrar BNF a Forma Panini-Backus.

John Backus. Diseñador de lenguajes de programación de IBM, adoptó las reglas de Chomsky para describir la sintaxis de un nuevo lenguaje de programación IAL, conocido en la actualidad como ALGOL 58 (1959), este se presentó por primera vez en el Congreso de Computación Mundial.

Peter Naur, realizó un reporte sobre ALGOL 60 (1963) e identificó la notación de Backus como la Forma Normal de Backus y a su vez la simplificó para usar un conjunto de símbolos menor, pero a sugerencia de Donald Knuth, su apellido fue agregado en reconocimiento a su contribución.

### **Backus-Naur**

Para la construcción de la sintaxis del lenguaje usaremos la notación Backus-Naur (BNF por sus siglas en inglés Backus Naur Form) la cual es un metalenguaje usado para expresar gramáticas libres de contexto, es decir una manera formal de describir lenguajes formales.

El BNF se utiliza como notación para las gramáticas de los lenguajes de programación, de los sistemas de comando y protocolos de comunicación, así como una notación para representar partes de las gramáticas de la lengua natural. Un BNF se escribe de la siguiente manera

<símbolo> ::= <expresión con símbolos>

Donde <símbolo> es un no terminal, la expresión consiste en secuencias de símbolos o secuencias separadas por la barra vertical '|', indicando una opción, el conjunto es una posible sustitución para el símbolo de la izquierda. Los símbolos que nunca aparecen en una lado izquierdo son terminales. [5]

Otra alternativa a este estilo de metalenguaje es la siguiente

```
símbolo = expresión
expresión = 'nombre'
```

Donde cada elemento dentro de comillas es un símbolo terminal, y al igual que el estilo de bnf anteriormente presentado se hace uso de la barra vertical "|", para indicar opción, para hacer uso de concatenación usamos la coma "," y para el final de la sentencia usamos punto y coma ". ". Nosotros usaremos la segunda opción, dado que es más rápido de escribir.

## Sintaxis

```
programa = 'init', '(', ')', '{' sentencia '}';
sentencia = if | while | read | while | print
| asignacion | declaracion | expresion;
read = 'read', '(', ( var | numero ), ')', ';' ;
print = 'print', '(', ( var | cadena ), ')', ';' ;
while = 'while', '(', expresion, ')', sentencia ;
if = 'if', '(', expresion, ')', '{' sentencia '}' , [else', '{' sentencia '}'] ;
asignacion = id, '=', expresion, ';' ;
declaracion = tipo, id, '=', expresion, ';' ;
expresion = exprMat, {(>,exprMat) | (<,exprMat) | (=,exprMat) | (!,exprMat) |
(<=,exprMat) | (>=,exprMat) | (==,exprMat) | (!=,exprMat) | (&&,exprMat) |
(||,exprMat)}
exprMat = termino, {(+,termino) | (-,termino)} ;
termino = factor, {(*,factor) | (/,factor)} ;
factor = '(', expresion, ')'
| '-', factor
| '!', factor
| numero
| cadena
| booleano
| id ;
id = letra, { letra | digito } ;
tipo = 'Int'|'Boolean'|'String' ;
booleano = 'true'|'false' ;
cadena = "", ? todos los caracteres ?, "" ;
letra = "A" | "B" | "C" | "D" | "E" | "F" | "G"
| "H" | "I" | "J" | "K" | "L" | "M" | "N"
| "O" | "P" | "Q" | "R" | "S" | "T" | "U"
| "V" | "W" | "X" | "Y" | "Z" | "a" | "b"
| "c" | "d" | "e" | "f" | "g" | "h" | "i"
| "j" | "k" | "l" | "m" | "n" | "o" | "p"
| "q" | "r" | "s" | "t" | "u" | "v" | "w"
| "x" | "y" | "z" | letra;

numero = digito, { digito } ;
digito = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
```

## Ventajas

1. Se puede considerar un lenguaje sencillo, para realizar operaciones muy simples
2. Es muy similar a C por lo cual es facil de aprender
3. Es fuertemente tipado
4. Es util para uso de scripting shell

## Desventajas

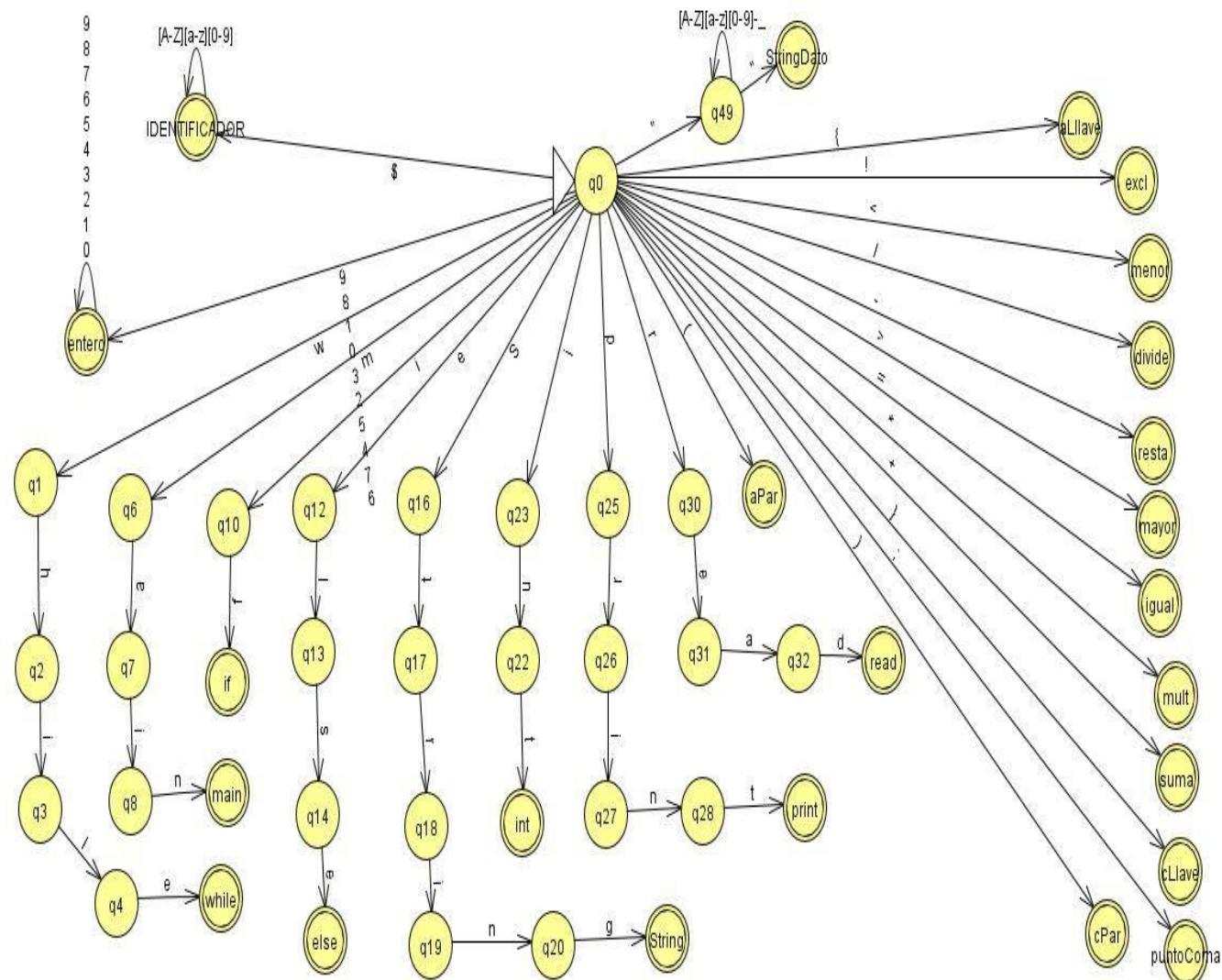
1. Muy pocos tipos de datos

2. No es orientado a objetos
3. Carece de funciones adicionales, por lo cual todo se debe crear desde cero
4. Se asimila mucho a C sin embargo tiene muchas carencia
5. Es de programación secuencial (Una instrucción detrás de otra)

**[6] Roberto Martínez**  
**Formato Backus-Naur (Lab)**  
**Ldp-roberto.blogspot.mx**  
<http://Ldp-roberto.blogspot.mx/2010/11/formato-backus-naur.html>

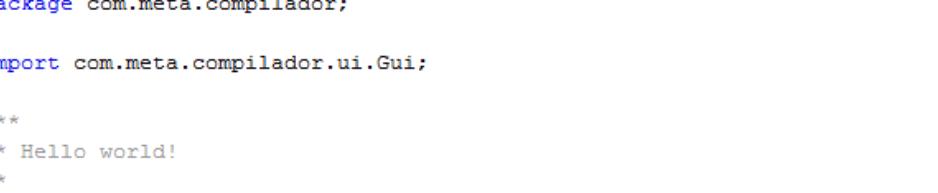
### Autómata generado en JFlap

Este autómata reconoce nuestras palabras reservadas en el lenguaje prototipo:



# Código Fuente (Evidencia de la programación)

## App.java



```
1 package com.meta.compilador;
2
3 import com.meta.compilador.ui.Gui;
4
5 /**
6  * Hello world!
7  *
8  */
9 public class App
10 {
11     public static void main( String[] args )
12     {
13         new Gui(true);
14     }
15 }
16
```

## Estados.java

```
1 package com.meta.compilador.lexico;
2
3 public enum Estados {
4     Q0("q0",Boolean.FALSE),
5
6     Q1("q1",Boolean.FALSE),
7     Q2("q2",Boolean.FALSE),
8     Q3("q3",Boolean.FALSE),
9     Q4("q4",Boolean.FALSE),
10    WHILE("while",Boolean.TRUE),
11
12    Q6("q6",Boolean.FALSE),
13    Q7("q7",Boolean.FALSE),
14    Q8("q8",Boolean.FALSE),
15    MAIN("main",Boolean.TRUE),
16
17    Q10("q10",Boolean.FALSE),
18    IF("if",Boolean.TRUE),
19
20    Q12("q12",Boolean.FALSE),
21    Q13("q13",Boolean.FALSE),
22    Q14("q14",Boolean.FALSE),
23    ELSE("else",Boolean.TRUE),
24
25    Q16("q16",Boolean.FALSE),
26    Q17("q17",Boolean.FALSE),
27    Q18("q18",Boolean.FALSE),
28    Q19("q19",Boolean.FALSE),
29    Q20("q20",Boolean.FALSE),
30    STRING("String",Boolean.TRUE),|
```

```
31
32     Q22("q22",Boolean.FALSE),
33     Q23("q23",Boolean.FALSE),
34     INT("Int",Boolean.TRUE),
35
36     Q25("q25",Boolean.FALSE),
37     Q26("q26",Boolean.FALSE),
38     Q27("q27",Boolean.FALSE),
39     Q28("q28",Boolean.FALSE),
40     PRINT("print",Boolean.TRUE),
41
42
43     Q30("q30",Boolean.FALSE),
44     Q31("q31",Boolean.FALSE),
45     Q32("q32",Boolean.FALSE),
46     READ("read",Boolean.TRUE),
47
48     ABRE_PARENTESIS("abre parentisis",Boolean.TRUE),
49     CIERRE_PARENTESIS("cierre parentesis",Boolean.TRUE),
50     PUNTO_COMA("punto y coma",Boolean.TRUE),
51     CIERRE_LLAVE("cierre Llave",Boolean.TRUE),
52     SUMA("suma",Boolean.TRUE),
53     MULTIPLICACION("multiplicacion",Boolean.TRUE),
54     IGUAL("igual",Boolean.TRUE),
55     MAYOR("mayor",Boolean.TRUE),
56     RESTA("resta",Boolean.TRUE),
57     DIVIDE("divide",Boolean.TRUE),
58     MENOR("menor",Boolean.TRUE),
59     EXCL("excl",Boolean.TRUE),
60     ALLAVE("aLLave",Boolean.TRUE),
61     ENTERO("entero",Boolean.TRUE),
62     IDENTIFICADOR("identificador",Boolean.TRUE),
```

The screenshot shows a Java code editor with the file 'Estados.java' open. The code defines a class 'Estados' with private fields 'token' and 'fin', and public methods 'getToken()' and 'isFin()'. The code is color-coded for syntax: tokens in green, strings in orange, and boolean values in blue.

```
63     Q49("q49",Boolean.FALSE),  
64     STRING_DATO("String dato",Boolean.TRUE)  
65 ;  
66  
67  
68  
69     private final String token;  
70     private final boolean fin;  
71     Estados (String token,boolean fin){  
72         this.token = token;  
73         this.fin = fin;  
74     }  
75     public String getToken() {  
76         return token;  
77     }  
78     public boolean isFin() {  
79         return fin;  
80     }  
81  
82 }  
83 }
```

## Lexico.java

The screenshot shows a Java code editor with the file 'Lexico.java' open. The code defines a class 'Lexico' with various imports and a static block that initializes a transition map.

```
1 package com.meta.compilador.lexico;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.io.RandomAccessFile;
8 import java.util.ArrayList;
9 import java.util.HashMap;
10 import java.util.LinkedHashMap;
11 import java.util.Map;
12
13 import com.meta.compilador.utils.Error;
14 import com.meta.compilador.utils.PilaErrores;
15 import com.meta.compilador.utils.TablaSimbolos;
16 import com.meta.compilador.utils.Token;
17
18 public class Lexico {
19     static private LinkedHashMap<Estados, Map<Character, Estados>> automata = new LinkedHashMap<>();
20     RandomAccessFile raf;
21
22     ArrayList<String> simbolos = new ArrayList<>();
23     static {
24
25         Map<Character, Estados> transicion = new HashMap<>();
26         // Transiciones de q0
27         transicion.put('w', Estados.Q1);
28         transicion.put('m', Estados.Q6);
29         transicion.put('i', Estados.Q10);
30         transicion.put('e', Estados.Q12);
31         transicion.put('S', Estados.Q16);
32         transicion.put('I', Estados.Q22);
```

```
automata.put(Estados.Q0, transicion);
transicion = null;
// Transiciones de Q1
transicion = new HashMap<>();
transicion.put('h', Estados.Q2);
automata.put(Estados.Q1, transicion);
transicion = null;
// Transiciones de Q2
transicion = new HashMap<>();
transicion.put('i', Estados.Q3);
automata.put(Estados.Q2, transicion);
transicion = null;
// Transiciones de Q3
transicion = new HashMap<>();
transicion.put('l', Estados.Q4);
automata.put(Estados.Q3, transicion);
transicion = null;
// Transiciones de Q4
transicion = new HashMap<>();
transicion.put('e', Estados.WHILE);
automata.put(Estados.Q4, transicion);
transicion = null;

// Transicion Q6
transicion = new HashMap<>();
transicion.put('a', Estados.Q7);
automata.put(Estados.Q6, transicion);
transicion = null;
// Transicion Q7
transicion = new HashMap<>();
transicion.put('i', Estados.Q8);
automata.put(Estados.Q7, transicion);
```

The screenshot shows a Java code editor with the file `Lexico.java` open. The code implements a finite state machine (`Automata`) for lexical analysis. It defines states (`MAIN`, `IF`, `Q10`, `Q12`, `Q13`, `Q14`, `ELSE`, `Q16`, `Q17`) and transitions based on characters ('n', 'f', 'l', 's', 'e', 't'). The code uses nested `HashMap`s to map characters to states.

```
automata.put(Estados.Q7, transicion);
transicion = null;
// Transicion Q8
transicion = new HashMap<>();
transicion.put('n', Estados.MAIN);
automata.put(Estados.Q8, transicion);
transicion = null;

// Transicion Q10
transicion = new HashMap<>();
transicion.put('f', Estados.IF);
automata.put(Estados.Q10, transicion);
transicion = null;

// Transicion Q12
transicion = new HashMap<>();
transicion.put('l', Estados.Q12);
automata.put(Estados.Q12, transicion);
transicion = null;
// Transicion Q13
transicion = new HashMap<>();
transicion.put('s', Estados.Q14);
automata.put(Estados.Q13, transicion);
transicion = null;
// Transicion Q14
transicion = new HashMap<>();
transicion.put('e', Estados.ELSE);
automata.put(Estados.Q14, transicion);
transicion = null;

// Transicion Q16
transicion = new HashMap<>();
transicion.put('t', Estados.Q17);
```

```
126     transicion.put('t', Estados.Q17);
127     automata.put(Estados.Q16, transicion);
128     transicion = null;
129
130     transicion = new HashMap<>();
131     transicion.put('r', Estados.Q18);
132     automata.put(Estados.Q17, transicion);
133     transicion = null;
134
135     transicion = new HashMap<>();
136     transicion.put('l', Estados.Q19);
137     automata.put(Estados.Q18, transicion);
138     transicion = null;
139
140     transicion = new HashMap<>();
141     transicion.put('n', Estados.Q20);
142     automata.put(Estados.Q19, transicion);
143     transicion = null;
144
145     transicion = new HashMap<>();
146     transicion.put('g', Estados.STRING);
147     automata.put(Estados.Q20, transicion);
148     transicion = null;
149
150
151     transicion = new HashMap<>();
152     transicion.put('n', Estados.Q23);
153     automata.put(Estados.Q22, transicion);
154     transicion = null;
155
156
157     transicion = new HashMap<>();
158     transicion.put('n', Estados.Q23);
159     automata.put(Estados.Q22, transicion);
160     transicion = null;
```

The screenshot shows a Java code editor with the file 'Lexico.java' open. The code is part of a lexical analyzer implementation, specifically handling transitions for characters 't', 'r', 'i', 'n', 't', 'e', and 'e'. The code uses HashMaps to map characters to states and automata objects. The editor interface includes tabs for 'Source' and 'History', and various toolbars and status bars typical of an IDE.

```
157 // Transicion Q23
158 transicion = new HashMap<>();
159 transicion.put('t', Estados.INT);
160 automata.put(Estados.Q23, transicion);
161 transicion = null;
162
163 // Transicion Q25
164 transicion = new HashMap<>();
165 transicion.put('r', Estados.Q26);
166 automata.put(Estados.Q25, transicion);
167 transicion = null;
168 // Transicion Q26
169 transicion = new HashMap<>();
170 transicion.put('i', Estados.Q27);
171 automata.put(Estados.Q26, transicion);
172 transicion = null;
173 // Transicion Q27
174 transicion = new HashMap<>();
175 transicion.put('n', Estados.Q28);
176 automata.put(Estados.Q27, transicion);
177 transicion = null;
178 // Transicion Q28
179 transicion = new HashMap<>();
180 transicion.put('t', Estados.PRINT);
181 automata.put(Estados.Q28, transicion);
182 transicion = null;
183
184 // Transicion Q30
185 transicion = new HashMap<>();
186 transicion.put('e', Estados.Q31);
187 automata.put(Estados.Q30, transicion);
188 transicion = null;
```

The screenshot shows a Java code editor window with the file 'Lexico.java' open. The code is written in Spanish and defines a state transition automaton. It includes comments in Spanish (# Transicion Q30, # Transicion Q31, etc.) and uses variables like 'transicion' (transition), 'automata' (automaton), and 'Estados' (States). The code handles transitions for characters 'e', 'a', 'd', digits from 1 to 9, and the state 'ENTERO'. The code editor has a toolbar at the top with various icons for file operations like Open, Save, Print, and Undo/Redo.

```
184 // Transicion Q30
185 transicion = new HashMap<>();
186 transicion.put('e', Estados.Q31);
187 automata.put(Estados.Q30, transicion);
188 transicion = null;
189 // Transicion Q31
190 transicion = new HashMap<>();
191 transicion.put('a', Estados.Q32);
192 automata.put(Estados.Q31, transicion);
193 transicion = null;
194 // Transicion Q32
195 transicion = new HashMap<>();
196 transicion.put('d', Estados.READ);
197 automata.put(Estados.Q32, transicion);
198 transicion = null;
199
200 // Transicion de Entero
201 transicion = new HashMap<>();
202 transicion.put('1', Estados.ENTERO);
203 transicion.put('2', Estados.ENTERO);
204 transicion.put('3', Estados.ENTERO);
205 transicion.put('4', Estados.ENTERO);
206 transicion.put('5', Estados.ENTERO);
207 transicion.put('6', Estados.ENTERO);
208 transicion.put('7', Estados.ENTERO);
209 transicion.put('8', Estados.ENTERO);
210 transicion.put('9', Estados.ENTERO);
211 transicion.put('0', Estados.ENTERO);
212 automata.put(Estados.ENTERO, transicion);
213 transicion = null;
214 //Transiciones de Identificadores
215
```

Lexico.java

```

215
216     transicion = new HashMap<>();
217     transicion.put('1', Estados.IDENTIFICADOR);
218     transicion.put('2', Estados.IDENTIFICADOR);
219     transicion.put('3', Estados.IDENTIFICADOR);
220     transicion.put('4', Estados.IDENTIFICADOR);
221     transicion.put('5', Estados.IDENTIFICADOR);
222     transicion.put('6', Estados.IDENTIFICADOR);
223     transicion.put('7', Estados.IDENTIFICADOR);
224     transicion.put('8', Estados.IDENTIFICADOR);
225     transicion.put('9', Estados.IDENTIFICADOR);
226     transicion.put('0', Estados.IDENTIFICADOR);
227     transicion.put('Q', Estados.IDENTIFICADOR);
228     transicion.put('W', Estados.IDENTIFICADOR);
229     transicion.put('E', Estados.IDENTIFICADOR);
230     transicion.put('R', Estados.IDENTIFICADOR);
231     transicion.put('T', Estados.IDENTIFICADOR);
232     transicion.put('Y', Estados.IDENTIFICADOR);
233     transicion.put('U', Estados.IDENTIFICADOR);
234     transicion.put('I', Estados.IDENTIFICADOR);
235     transicion.put('O', Estados.IDENTIFICADOR);
236     transicion.put('P', Estados.IDENTIFICADOR);
237     transicion.put('A', Estados.IDENTIFICADOR);
238     transicion.put('S', Estados.IDENTIFICADOR);
239     transicion.put('D', Estados.IDENTIFICADOR);
240     transicion.put('F', Estados.IDENTIFICADOR);
241     transicion.put('G', Estados.IDENTIFICADOR);
242     transicion.put('H', Estados.IDENTIFICADOR);
243     transicion.put('J', Estados.IDENTIFICADOR);
244     transicion.put('K', Estados.IDENTIFICADOR);
245     transicion.put('L', Estados.IDENTIFICADOR);
246     transicion.put('Z', Estados.IDENTIFICADOR);
247     transicion.put('X', Estados.IDENTIFICADOR);

Lexico.java
```

Source History

```

247     transicion.put('X', Estados.IDENTIFICADOR);
248     transicion.put('C', Estados.IDENTIFICADOR);
249     transicion.put('V', Estados.IDENTIFICADOR);
250     transicion.put('B', Estados.IDENTIFICADOR);
251     transicion.put('N', Estados.IDENTIFICADOR);
252     transicion.put('M', Estados.IDENTIFICADOR);
253     transicion.put('q', Estados.IDENTIFICADOR);
254     transicion.put('w', Estados.IDENTIFICADOR);
255     transicion.put('e', Estados.IDENTIFICADOR);
256     transicion.put('r', Estados.IDENTIFICADOR);
257     transicion.put('t', Estados.IDENTIFICADOR);
258     transicion.put('y', Estados.IDENTIFICADOR);
259     transicion.put('u', Estados.IDENTIFICADOR);
260     transicion.put('i', Estados.IDENTIFICADOR);
261     transicion.put('o', Estados.IDENTIFICADOR);
262     transicion.put('p', Estados.IDENTIFICADOR);
263     transicion.put('a', Estados.IDENTIFICADOR);
264     transicion.put('s', Estados.IDENTIFICADOR);
265     transicion.put('d', Estados.IDENTIFICADOR);
266     transicion.put('f', Estados.IDENTIFICADOR);
267     transicion.put('g', Estados.IDENTIFICADOR);
268     transicion.put('h', Estados.IDENTIFICADOR);
269     transicion.put('j', Estados.IDENTIFICADOR);
270     transicion.put('k', Estados.IDENTIFICADOR);
271     transicion.put('l', Estados.IDENTIFICADOR);
272     transicion.put('z', Estados.IDENTIFICADOR);
273     transicion.put('x', Estados.IDENTIFICADOR);
274     transicion.put('c', Estados.IDENTIFICADOR);
275     transicion.put('v', Estados.IDENTIFICADOR);
276     transicion.put('b', Estados.IDENTIFICADOR);
277     transicion.put('n', Estados.IDENTIFICADOR);
278     transicion.put('m', Estados.IDENTIFICADOR);

```

Source History

Lexico.java

```

279     automata.put(Estados.IDENTIFICADOR, transicion);
280     transicion = null;
281
282     //Transiciones Cadenas
283     transicion = new HashMap<>();
284     transicion.put('1', Estados.Q49);
285     transicion.put('2', Estados.Q49);
286     transicion.put('3', Estados.Q49);
287     transicion.put('4', Estados.Q49);
288     transicion.put('5', Estados.Q49);
289     transicion.put('6', Estados.Q49);
290     transicion.put('7', Estados.Q49);
291     transicion.put('8', Estados.Q49);
292     transicion.put('9', Estados.Q49);
293     transicion.put('0', Estados.Q49);
294     transicion.put('Q', Estados.Q49);
295     transicion.put('W', Estados.Q49);
296     transicion.put('E', Estados.Q49);
297     transicion.put('R', Estados.Q49);
298     transicion.put('T', Estados.Q49);
299     transicion.put('Y', Estados.Q49);
300     transicion.put('U', Estados.Q49);
301     transicion.put('I', Estados.Q49);
302     transicion.put('O', Estados.Q49);
303     transicion.put('P', Estados.Q49);
304     transicion.put('A', Estados.Q49);
305     transicion.put('S', Estados.Q49);
306     transicion.put('D', Estados.Q49);
307     transicion.put('F', Estados.Q49);
308     transicion.put('G', Estados.Q49);
309     transicion.put('H', Estados.Q49);
310     transicion.put('J', Estados.Q49);
311
312     transicion.put('K', Estados.Q49);
313     transicion.put('L', Estados.Q49);
314     transicion.put('Z', Estados.Q49);
315     transicion.put('X', Estados.Q49);
316     transicion.put('C', Estados.Q49);
317     transicion.put('V', Estados.Q49);
318     transicion.put('B', Estados.Q49);
319     transicion.put('N', Estados.Q49);
320     transicion.put('M', Estados.Q49);
321     transicion.put('q', Estados.Q49);
322     transicion.put('w', Estados.Q49);
323     transicion.put('e', Estados.Q49);
324     transicion.put('r', Estados.Q49);
325     transicion.put('t', Estados.Q49);
326     transicion.put('y', Estados.Q49);
327     transicion.put('u', Estados.Q49);
328     transicion.put('i', Estados.Q49);
329     transicion.put('o', Estados.Q49);
330     transicion.put('p', Estados.Q49);
331     transicion.put('a', Estados.Q49);
332     transicion.put('s', Estados.Q49);
333     transicion.put('d', Estados.Q49);
334     transicion.put('f', Estados.Q49);
335     transicion.put('g', Estados.Q49);
336     transicion.put('h', Estados.Q49);
337     transicion.put('j', Estados.Q49);
338     transicion.put('k', Estados.Q49);
339     transicion.put('l', Estados.Q49);
340     transicion.put('z', Estados.Q49);
341     transicion.put('x', Estados.Q49);
342     transicion.put('c', Estados.Q49);
343     transicion.put('v', Estados.Q49);

```



```

Lexico.java
Source History
405     sb.append(caracter);
406     id++;
407     TablaSimbolos.tabla.insertar(new Token(id, Estados.IDENTIFICADOR getToken(), sb.toString(), Estados.IDENTIFICADOR));
408     sb = null;
409     sb = new StringBuilder();
410     continue;
411 }
412 //Checa estados para palabras reservadas
413 if (siguiente != null && !siguiente.isFin()) {
414     actual = siguiente;
415     sb.append(caracter);
416     continue;
417 } else if (siguiente != null && siguiente.isFin()) {
418     id++;
419     sb.append(caracter);
420     TablaSimbolos.tabla.insertar(new Token(id, "Palabra reservada", sb.toString(), siguiente.getToken()));
421     actual = Estados.Q0;
422     sb = null;
423     sb = new StringBuilder();
424     continue;
425 }
426 //Omite los espacios
427 if (caracter == ' ' || caracter == '\t' || caracter == '\n' || caracter == '\u0000')
428     continue;
429
430 System.out.println("Error caracter no valido "+caracter +"Pos "+posicion);
431 PilaErrores.pila_errores.push(new Error(100, "Caracter no valido "+caracter, posicion));
432 break;
433 } while (caracter != '\u0000');

Lexico.java
Source History
436     public void leerTransicionActual(Map<Character, Estados> transiciones) {
437         for (Map.Entry<Character, Estados> e : transiciones.entrySet()) {
438             System.out.println(e.getKey() + " - " + e.getValue());
439         }
440     }
441
442     public char caracter() {
443         char caracter = '\u0000';
444         try {
445             caracter = (char) raf.readByte();
446         } catch (IOException e) {
447             System.out.println(e.getMessage());
448         }
449         return caracter;
450     }
451
452     public static void main(String... arg) {
453         Lexico l;
454         String var = "#Este es un comentario#";
455         try {
456             File archivo = new File("archivo");
457             BufferedWriter salida = new BufferedWriter(new FileWriter(archivo));
458             salida.write(var);
459             salida.close();
460             archivo = new File("archivo");
461             RandomAccessFile raf = new RandomAccessFile(archivo, "rw");
462             l = new Lexico(raf);
463             l.analisis();
464         } catch (IOException ex) {
465             System.out.println(ex.getMessage());
466         }
}

```

Lexico.java

```
451     }
452     public static void main(String... arg) {
453         Lexico l;
454         String var = "#Este es un comentario#";
455         try {
456             File archivo = new File("archivo");
457             BufferedWriter salida = new BufferedWriter(new FileWriter(archivo));
458             salida.write(var);
459             salida.close();
460             archivo = new File("archivo");
461             RandomAccessFile raf = new RandomAccessFile(archivo, "rw");
462             l = new Lexico(raf);
463             l.analisis();
464         } catch (IOException ex) {
465             System.out.println(ex.getMessage());
466         }
467         TablaSimbolos.tabla.verTabla();
468     }
469 }
470 }
```

## Gui.java

Gui.java

```
1 package com.meta.compilador.ui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.io.BufferedReader;
8 import java.io.BufferedWriter;
9 import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.io.FileReader;
12 import java.io.FileWriter;
13 import java.io.IOException;
14 import java.io.PrintWriter;
15 import java.io.RandomAccessFile;
16
17 import javax.swing.JButton;
18 import javax.swing.JFileChooser;
19 import javax.swing.JFrame;
20 import javax.swing.JLabel;
21 import javax.swing.JMenu;
22 import javax.swing.JMenuBar;
23 import javax.swing.JMenuItem;
24 import javax.swing.JPanel;
25 import javax.swing.JScrollPane;
26 import javax.swing.JTable;
27 import javax.swing.JTextArea;
28 import javax.swing.JToolBar;
29 import javax.swing.table.DefaultTableModel;
30
31 import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
32 import org.fife.ui.rsyntaxtextarea.SyntaxConstants;
```

```
import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
import org.fife.ui.rsyntaxtextarea.SyntaxConstants;
import org.fife.ui.rtextarea.RTextScrollPane;

private static final long serialVersionUID = 1L;

private JPanel principal;
private RSyntaxTextArea editor;
private DefaultTableModel model;
private final String[] campos = { "ID", "Tipo", "LEXEMA", "DESCRIPCION" };
private JTextArea consola;
private Lexico analisisLexico;
private JButton lexico, sintactico, semantico, run;
private File archivo;

public Gui(boolean visible) {
    super("Editor");

    init();
    setJMenuBar(menu());

    pack();
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setVisible(visible);
}

public void init() {
    principal = new JPanel(new BorderLayout());

    // Editor
    editor = new RSyntaxTextArea(20, 60);
    editor.setSyntaxEditingStyle(SyntaxConstants.SYNTAX_STYLE_JAVA);
    editor.setCodeFoldingEnabled(true);
    RTextScrollPane sp = new RTextScrollPane(editor);
    principal.add(sp, BorderLayout.CENTER);

    // Tabla
    JTable tabla = new JTable();
    model = (DefaultTableModel) tabla.getModel();
    for (String campo : campos)
        model.addColumn(campo);
    principal.add(new JScrollPane(tabla), BorderLayout.EAST);

    // Consola
    JPanel pnlConsola = new JPanel(new BorderLayout());
    consola = new JTextArea(10, 10);
    consola.setEditable(false);
    pnlConsola.add(new JLabel("Consola"), BorderLayout.NORTH);
    pnlConsola.add(new JScrollPane(consola), BorderLayout.CENTER);
    principal.add(pnlConsola, BorderLayout.SOUTH);

    JToolBar tools = herramientas();
    principal.add(tools, BorderLayout.NORTH);

    setContentPane(principal);
}
```

Gui.java

```
Source History □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

```
96     public JMenuBar menu() {
97         JMenuBar menuBar = new JMenuBar();
98         // Menu archivo
99         JMenuItem archivo = new JMenuItem("Archivo");
100        JMenuItem abrir = new JMenuItem("Abrir");
101        abrir.addActionListener(new Oyente(0));
102
103        JMenuItem guardar = new JMenuItem("Guardar");
104        guardar.addActionListener(new Oyente(1));
105        JMenuItem salir = new JMenuItem("salir");
106        salir.addActionListener(new Oyente(2));
107
108        JMenuItem nuevo = new JMenuItem("Nuevo");
109
110        nuevo.addActionListener(new Oyente(3));
111        archivo.add(nuevo);
112        archivo.add(abrir);
113        archivo.add(guardar);
114        archivo.addSeparator();
115        archivo.add(salir);
116        menuBar.add(archivo);
117
118    }
119 }
```

Gui.java

```
Source History □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

```
119     public JToolBar herramientas() {
120         JToolBar tools = new JToolBar();
121         lexico = new JButton("Lexico");
122         sintactico = new JButton("Sintactico");
123         semantico = new JButton("Semantico");
124         run = new JButton("Run");
125         tools.add(run);
126         tools.add(lexico);
127         lexico.addActionListener(new Oyente(4));
128         tools.add(sintactico);
129         tools.add(semantico);
130
131     }
132 }
```

**Guia.java**

```

134     public void guardarArchivo() {
135         String texto = editor.getText();
136         if(archivo != null)
137             {
138                 try {
139                     PrintWriter pw = new PrintWriter(archivo);
140                     pw.print(texto);
141                     pw.close();
142                     String path = archivo.getPath();
143                     archivo = new File(path);
144
145                 }catch(IOException io) {
146                     System.out.println("No se pudo guardar");
147                 }
148             }else {
149                 JFileChooser chooser = new JFileChooser();
150                 chooser.showSaveDialog(this);
151                 archivo = chooser.getSelectedFile();
152                 if(archivo != null)
153                     try {
154                         FileWriter fw = new FileWriter(archivo);
155                         PrintWriter pw = new PrintWriter(fw);
156                         pw.print(texto);
157                         pw.close();
158                     } catch (Exception e) {
159                         // TODO Auto-generated catch block
160                         e.printStackTrace();
161                     }
162
163             }
164
165         }

```

---

**Guia.java**

```

167     public void abrirArchivo() {
168         JFileChooser chooser = new JFileChooser();
169         String texto, aux;
170         aux = texto = "";
171         chooser.showOpenDialog(this);
172         chooser.setCurrentDirectory(new File("./ejemplos"));
173         archivo = chooser.getSelectedFile();
174         if (archivo != null) {
175             FileReader fr;
176             try {
177                 fr = new FileReader(archivo);
178                 BufferedReader br = new BufferedReader(fr);
179                 while ((aux = br.readLine()) != null)
180                     texto+= aux +"\n";
181             } catch (Exception e) {
182                 // TODO Auto-generated catch block
183                 e.printStackTrace();
184             }
185         }
186         editor.setText(texto);
187
188     public void run() {
189
190     }

```

Gui.java

```
193     public void hacerLexico() {
194         TablaSimbolos.tabla.limpiar();
195         PilaErrores.pila_errores.clear();
196         String errores = "";
197         try {
198             File temp = new File("temp.team");
199             BufferedWriter salida = new BufferedWriter(new FileWriter(temp));
200             salida.write(editor.getText());
201             salida.close();
202             temp = new File("temp.team");
203             RandomAccessFile raf = new RandomAccessFile(temp, "rw");
204             analisisLexico = new Lexico(raf);
205             analisisLexico.analisis();
206         } catch (Exception e) {
207             // TODO Auto-generated catch block
208             e.printStackTrace();
209         }
210         if(!PilaErrores.pila_errores.isEmpty())
211             lexico.setBackground(Color.RED);
212         else
213             lexico.setBackground(Color.GREEN);
214         while(!PilaErrores.pila_errores.isEmpty())
215             errores+= PilaErrores.pila_errores.pop().toString();
216         consola.setText(errores);
217         for(int i = 0 ; i < model.getRowCount();i++)
218         {
219             model.removeRow(i);
220         }
221         TablaSimbolos.tabla.verTablaModel(model);
222     }
```

Gui.java

```
223     private class Oyente implements ActionListener{
224         private int ID;
225         public Oyente(int ID) {
226             this.ID = ID;
227         }
228         @Override
229         public void actionPerformed(ActionEvent e) {
230             switch (ID) {
231                 //Abrir
232                 case 0:
233                     abrirArchivo();
234                     break;
235                 //Guardar
236                 case 1:
237                     guardarArchivo();
238                     break;
239                 //Salir
240                 case 2:
241                     System.exit(0);
242                     break;
243                 case 3:
244                     editor.setText("");
245                     archivo = null;
246                     break;
247                 case 4:
248                     hacerLexico();
249                     break;
250             }
251         }
252     }
253 }
```

## Error.java

The screenshot shows a Java code editor with the file `Error.java` open. The code defines a class `Error` with private attributes `id`, `tipo`, and `posicion`. It includes a constructor, an overridden `toString` method, and getters and setters for each attribute. The code is well-formatted with proper indentation and syntax highlighting.

```
1 package com.meta.compilador.utils;
2
3 public class Error {
4     private int id;
5     private String tipo;
6     private int posicion;
7
8
9
10    public Error(int id, String tipo, int posicion) {
11        super();
12        this.id = id;
13        this.tipo = tipo;
14        this.posicion = posicion;
15    }
16
17    @Override
18    public String toString() {
19        return "Error [id=" + id + ", tipo=" + tipo + ", posicion=" + posicion + "]";
20    }
21    public int getId() {
22        return id;
23    }
24    public void setId(int id) {
25        this.id = id;
26    }
27    public String getTipo() {
28        return tipo;
29    }
30    public void setTipo(String tipo) {
31        this.tipo = tipo;
32    }
33
34    public int getPosition() {
35        return posicion;
36    }
37    public void setPosition(int posicion) {
38        this.posicion = posicion;
39    }
40}
```

## PilaErrores.java

The screenshot shows a Java code editor with the file `PilaErrores.java` open. The code defines a class `PilaErrores` that contains a static variable `pila_errores` of type `Stack<Error>`. This is a simple implementation of a stack using the `java.util.Stack` class.

```
1 package com.meta.compilador.utils;
2
3 import java.util.Stack;
4
5 public class PilaErrores {
6     public static Stack<Error> pila_errores = new Stack<>();
7 }
```

## TablaSimbolos.java

Token.java

The screenshot shows a Java code editor with the following details:

- Title Bar:** The title bar displays "Token.java x".
- Toolbar:** A standard toolbar with icons for file operations (New, Open, Save, Print, Find, Replace, Copy, Paste, Cut, Undo, Redo), a search field, and other development tools.
- Code Area:** The main area contains the `Token.java` source code. The code defines a class `Token` with methods for string representation, constructor, and getters.
- Code Structure:** The code is annotated with several **Annotations** (represented by small circles with numbers) and **Outlines** (represented by small squares with numbers). Annotations are placed on lines 1, 4, 6, 8, 12, 15, 17, 20, 22, 23, 24, 25, 27, and 28. Outlines are placed on lines 1, 4, 6, 8, 12, 15, 17, 20, 22, 23, 24, 25, 27, and 29.

```
1 package com.meta.compilador.utils;
2
3 public class Token {
4     @Override
5     public String toString() {
6         return "Token [id=" + id + ", tipo=" + tipo + ", lexema=" + lexema + ", carac=" + carac + "]";
7     }
8     private int id;
9     private String tipo,lexema,carac;
10
11
12     public Token() {
13
14     }
15     public String[] getTokenModelRow()
16     {
17         String [] row = {id,"",tipo,lexema,carac};
18         return row;
19     }
20     public Token(int id, String tipo, String lexema, String carac) {
21         super();
22         this.id = id;
23         this.tipo = tipo;
24         this.lexema = lexema;
25         this.carac = carac;
26     }
27     public int getId() {
28         return id;
29     }
}
```

The screenshot shows a Java code editor with the file `Token.java` open. The code defines a class `Token` with several methods for setting and getting attributes: `id`, `tipo`, `lexema`, and `carac`. The code is well-formatted with color-coded syntax highlighting.

```
30     public void setId(int id) {
31         this.id = id;
32     }
33     public String getTipo() {
34         return tipo;
35     }
36     public void setTipo(String tipo) {
37         this.tipo = tipo;
38     }
39     public String getLexema() {
40         return lexema;
41     }
42     public void setLexema(String lexema) {
43         this.lexema = lexema;
44     }
45     public String getCarac() {
46         return carac;
47     }
48     public void setCarac(String carac) {
49         this.carac = carac;
50     }
51 }
52 }
53 }
```

## AppTest.java

```
1 package com.meta.compilador;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8 * Unit test for simple App.
9 */
10 public class AppTest
11     extends TestCase
12 {
13     /**
14      * Create the test case
15      *
16      * @param testName name of the test case
17      */
18     public AppTest( String testName )
19     {
20         super( testName );
21     }
22
23     /**
24      * @return the suite of tests being tested
25      */
26     public static Test suite()
27     {
28         return new TestSuite( AppTest.class );
29     }
30
31     /**
32      * Rigourous Test :-)
33      */
34     public void testApp()
35     {
36         assertTrue( true );
37     }
38 }
39
```

- 4. PARA EL INCISO C DEL PUNTO 2 ANTERIOR, SE DEBERÁN CREAR PROGRAMAS DE MÍNIMO 25 LÍNEAS (SIN CONSIDERAR LOS COMENTARIOS) CADA UNO, EN LOS CUALES SE CODIFIQUE EN EL LENGUAJE PROTOTIPO, INSTRUCCIONES CON LÓGICA QUE EJEMPLIFIQUEN LOS ERRORES EN CADA CASO DE ESTUDIO QUE SE PROPONGAN.**
- TALES PROGRAMAS DEBEN DE ESTAR PERFECTAMENTE DOCUMENTADOS Y CORRELACIONADOS EN LOS CASOS DE ESTUDIO CORRESPONDIENTES.**

#### PROGRAMAS CON NUESTRO RESPECTIVO LENGUAJE-P

```
#programa que dice en que rango esta un numero del 1 al 100#
main() {
    print("Dame un numero");
    Int $i=read(); #lee una variable entera#
    while($i<=100) {
        print(i);

        if($i==1){ #compara si la variable es un uno entonces
es el primer numero#
            print("es el primer numero");
        }
        if($i>1 && $i<50){ #compara si la variable esta entre
1 y 50#
            print("el numero es menor que 50");
            if($i>10 && $i<30){ #compara si la variable esta
entre 10 y 30#
                print("el numero esta entre 10 y 30");
            }
        }
        if($i>50){ #compara si la variable es mayor a 50#
            print("el numero es mayor a 50");
            if($i>50 && $i<70){ #compara si la variable esta
entre 50 y 70#
                print("el numero esta entre 50 y 70");
            }
        }
    }
}
```

```

#te dice si dos números son amigos es decir el 220 sus divisores
son 1 ,2,4,5,10,11,22,110,20,55 su suma es 284 y el numero 284 sus
divisores son 1,2,4,71,142 su suma es 220 y también nos dice si un
número es perfecto o no es decir 6 sus divisores 1,2,3 su suma es 6
y 6 es igual al número entonces si es perfecto#

main() {
    Int $A=0,$B=0,$i=0,$sumaA=0,$sumaB=0;#declaramos las
variables que se utilizan para los numeros amigos#
    print("NUMEROS AMIGOS");
    print ("Teclea el primer numero: ");
    Int $A= read();#en esta parte la funcion no es correcta
(read())->read#
    print("Teclea el segundo numero: ");
    Int $B= read();
    mientras($i<=$A)#la estructura "mientras" no existe
(mientras())#
    {
        if ($A%$i==0) {$sumaA+=$i;}
        $i++;
    }
    while($i<=$B)
    {
        if ($B%$i==0) {$sumaB+=$i;}
        $i++ #no hemos puesto (;) al final de la instrucción#
    }
    if ($A==$sumaB && $B==$sumaA) # la variable B no tiene#
{
    print("Los numeros".$A." Y ".$B." son amigos");
}
else
{
    print("Los numeros ".$A." Y ".$B." son amigos");
}
print("NUMEROS PERFECTOS");
print("Teclea el numero a confirmar si es perfecto o no lo
es: ");
$A=read();#leemos el numero desde teclado#
$i=1;

```

```
while($i<=$A) #comparamos si i es menor o igual que el numero A#
{
    if ($A%i==0) {$sumaA+=$i;}
    i++;
}
if ($A==$sumaA)
{
    print("El numero "+$A+" es un numero perfecto");
}
else
{
    print("El numero "+$A+" no es un numero perfecto");
}

}
```

```

# programa que hace el factorial de un numero
#Este programa no se ejecutara pues el inicio de programa esta
incorrecto (main(){} ) -> MAIN {}#
MAIN() {
    Int $factorial = 1,$num; #Declaramos las variables#
    print("Ingrese un numero positivo para obtener su
factorial");#Pedimos un numero para calcular su factorial#
    $num = read();#leemos desde teclado un valor entero y se
asigna a una variable entera#
    if ($num==0) {
        print("se calculo 0 != " + $factorial); #si el numero
es cero su factorial es 1#
    }
    else {
        if ($num < 0) { #Avisamos que solo se puede
tener el factorial de un numero que sea positivo#
            print("Unicamente puede introducir numeros
positivos");
        }
        else {
            while ($num > 0) {
                $factorial = $factorial * $num;
                $num--;
            }
            print($num + " != " + $factorial);#se muestra el
factorial#
        }
    }
}
main() {
    print("Este programa de prueba no calcula el factorial de
un numero");
    print("Hola mundo");
}

```

```

#este programa da el puntaje apartir de cierta letra escogida por
el usuario#
main(){
print("Seleccione la calificacion(letra) para conocer su
puntaje\nA\nB\nC\nD\nE\nF");
String $letra=read();
if($letra=="A" || $letra=="a") { #se compara si la letra es
mayúscula o minúscula#
    print("La calificacion es 100");
}
if($letra=="B" || $letra=="b") { #se compara si la letra es
mayúscula o minúscula#
    print("La calificación es 90");
}
if($letra=="C" || $letra=="c") { #se compara si la letra es
mayúscula o minúscula#
    print("La calificación es 80");
}
if($letra=="D" || $letra=="d") { #se compara si la letra es
mayúscula o minúscula#
    print("La calificación es 75");
}
if($letra=="E" || $letra=="e") { #se compara si la letra es
mayúscula o minúscula#
    print("La calificación es 70");
}
Int $bandera=1;
if($bandera==1){
    fi($letra=="f")#la palabra reservada está mal escrita(if)#
    {
        print("La calificación no es aprobatoria esta reprobado");
    }
}
}

```

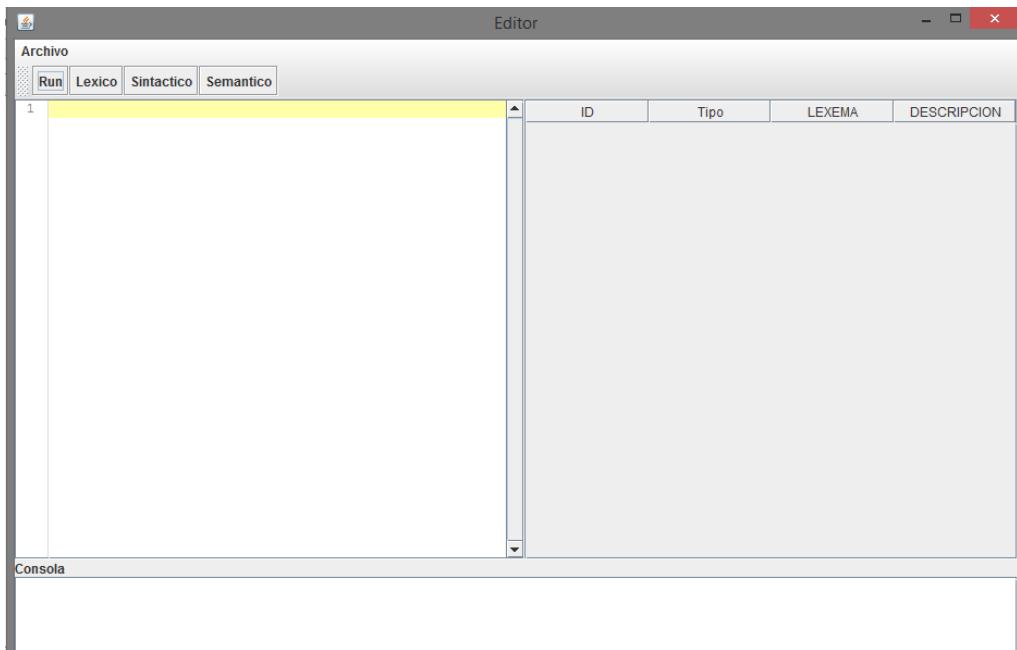
```

# Programa que lea un número entero y muestre si el número es múltiplo
de 10#
mainn (){ #palabra reservada mal escrita#
    Int $NN=0;
    Int $N;
    print("Número entero: ");
    $N = read();
    if($N%10==0)
        pritn("Es múltiplo de 10"); #palabra reservada mal escrita#
    else
        if($NN€==0){ #caracteres no definidos en el lenguaje#
            print("No es múltiplo de 10");
        }
    }
while($NN>=0){
    if($NN%2==0){
        print("numero par");
    }
    else{
        print("numero impar");
    }
}
while($N>=0){ #compara si la variable NN es mayor o igual a cero y
si N es par o impar según sea el caso lo informa#
    if($N%2==0){
        print("numero par");
    }
    else{
        print("numero impar");
    }
}
}

```

## Pruebas generadas sobre los códigos anteriormente descritos

Interfaz en la que se estará trabajando.



Aquí podemos cargar el programa y ejecutarlo cuando deseemos

A screenshot of the same 'Editor' interface as above, but now containing a program. The code is as follows:

```
1 # Programa que lee un número entero y muestre si el número es par o impar
2 mainn (){ #palabra reservada mal escrita#
3     Int $NN=0;
4     Int $N;
5     print("Número entero: ");
6     $N = read();
7     If($N%10==0)
8         pritn("Es múltiplo de 10"); #palabra reservada mal escrita#
9     else
10        if($NN!=0){ #caracteres no definidos en el leng
11            print("No es múltiplo de 10");
12        }
13    }
14 while($NN>=0){
15     if($NN%2==0){
16         print("numero par");
17     }
18     else{
19         print("numero impar");
20     }
21 }
22 while($N>=0){ #compara si la variable NN es mayor o igual
23     if($N%2==0){
24         print("numero par");
25     }
26 }
```

The 'Consola' panel below the editor is currently empty.

Una vez ejecutado podemos observar un error en main, ya que esta escrito de manera incorrecta mainn

The screenshot shows a code editor window titled "Editor". The menu bar has "Archivo" (File) selected. Below the menu is a tab bar with "Run", "Lexico" (Lexical), "Sintactico" (Syntax), and "Semantico". The "Lexico" tab is highlighted with a red border. The main area contains the following C-like pseudocode:

```
1 # Programa que lee un número entero y muestre si el número es múltiplo de 10
2 mainn (){ #palabra reservada mal escrita#
3     Int $NN=0;
4     Int $N;
5     print("Número entero: ");
6     $N = read();
7     if($NN%10==0)
8         printr("Es múltiplo de 10"); #palabra reservada mal escrita#
9     else
10        if($NN%2==0){
11            print("numero par");
12        }
13    }
14    while($NN>=0){
15        if($NN%2==0){
16            print("numero par");
17        }
18        else{
19            print("numero impar");
20        }
21    }
22    while($N>=0){ #compara si la variable NN es mayor o igual que N
23        if($N%2==0){
24            print("numero par");
25        }
26    }
}
```

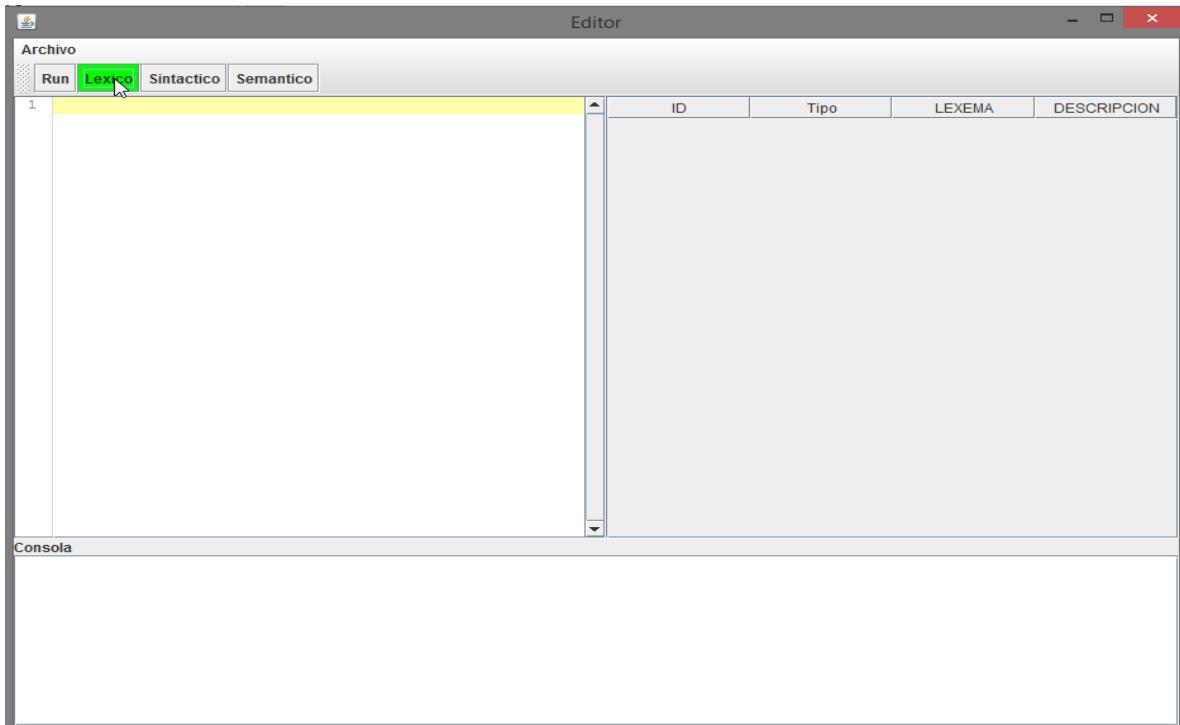
To the right of the code, there is a table titled "Reporte Lexico" (Lexical Report) with the following data:

ID	Tipo	LEXEMA	DESCRIPCION
2	Palabra reservada	main	main

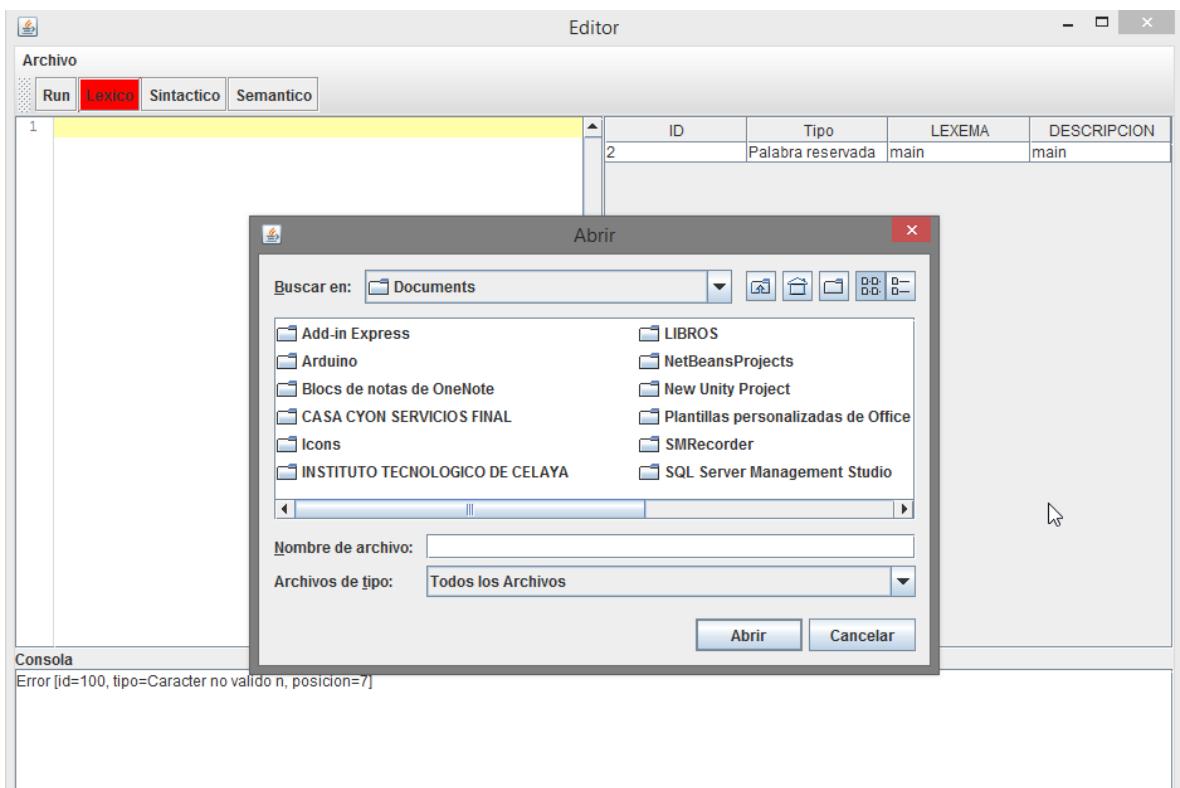
Below the code editor is a "Consola" (Console) window showing the error message: "Error [id=100, tipo=Caracter no valido n, posicion=7]".

Para crear un nuevo programa vamos a la pestaña archivo y seleccionamos nuevo





**De igual manera podemos cargar un archivo generado anteriormente y ejecutarlo**



Editor

Archivo

Lexico Sintactico Semantico

```

1 #este programa da el puntaje apartir de cierta letra escogida
2 main(){
3     print("Seleccione la calificacion(letra) para conocer su puntaje");
4     String $letra=read();
5     if($letra=="A" || $letra=="a") { #se compara si la letra es A
6         print("La calificación es 100");
7     }
8     if($letra=="B" || $letra=="b") { #se compara si la letra es B
9         print("La calificación es 90");
10    }
11   if($letra=="C" || $letra=="c") { #se compara si la letra es C
12       print("La calificación es 80");
13   }
14   if($letra=="D" || $letra=="d") { #se compara si la letra es D
15       print("La calificación es 75");
16   }
17   if($letra=="E" || $letra=="e") { #se compara si la letra es E
18       print("La calificación es 70");
19   }
20   Int $bandera=1;
21   if($bandera==1){
22       ...
23   }

```

ID	Tipo	LEXEMA	DESCRIPCION
2	Palabra reservada	main	main
3	Palabra reservada	(	abre parentesis
4	Palabra reservada	)	cierra parentesis
5	Palabra reservada	{	abre llave
6	Palabra reservada	print	print
7	Palabra reservada	(	abre parentesis

Consola

Error [id=100, tipo=Caracter no valido (, posicion=42]

Editor

Archivo

Lexico Sintactico Semantico

```

1 # programa que hace el factorial de un numero
2 #Este programa no se ejecutara pues el inicio de programa es incorrecto
3 MAIN() {
4     Int $factorial = 1,$num; #Declaramos las variables#
5     print("Ingrese un numero positivo para obtener su factorial");
6     $num = read();#leemos desde teclado un valor entero
7     if ($num==0) {
8         print("se calculo 0!= " + $factorial); #si el numero es 0
9     }
10    else {
11        if ($num < 0) { #Avisamos que solo se puede calcular numeros positivos
12            print("Unicamente puede introducir numeros positivos");
13        }
14        else {
15            while ($num > 0) {
16                $factorial = $factorial * $num;
17                $num--;
18            }
19        }
20        print(num + "!=" + $factorial);#se muestra el resultado
21    }
22 }
23 main(){
24     print("Este programa de prueba no calcula el factorial");
25     print("Hola mundo");

```

ID	Tipo	LEXEMA	DESCRIPCION
----	------	--------	-------------

Consola

Error [id=100, tipo=Caracter no valido (, posicion=1]

Editor

Archivo Run Lexico Sintactico Semantico

```

1 #te dice si dos números son amigos es decir el 220 sus divisores
2 main() {
3     Int $A=0,$B=0,$i=0,$sumaA=0,$sumaB=0;#declaramos las variables
4     print("NUMEROS AMIGOS");
5     print ("Te pido el primer numero: ");
6     Int $A= read();#en esta parte la funcion no es correcta
7     print ("Te pido el segundo numero: ");
8     Int $B= read();
9     mientras($i<=$A){#la estructura "mientras" no existe
10    {
11        if($A%$i==0){$sumaA+=$i;}
12        $i++;
13    }
14    while($i<=$B)
15    {
16        if($B%$i==0){$sumaB+=$i;}
17        $i++ #no hemos puesto (;) al final de la instrucción
18    }
19    if($A==$sumaB && $B==$sumaA){# la variable B no tiene
20        print("Los numeros "+$A+" Y "+$B+" son amigos");
21    }
22    else
23    {
24        print("Los numeros "+$A+" Y "+$B+" son amigos");
25    }
26 }

```

Consola

ID	Tipo	LEXEMA	DESCRIPCION
2	Palabra reservada	main	main
3	Palabra reservada	(	abre parentesis
4	Palabra reservada	)	cierre parentesis
5	Palabra reservada	{	aLLave
6	Palabra reservada	Int	Int
7	identificador	\$A=	identificador
8	ENTERO	0,	ENTERO
9	identificador	\$B=	identificador
10	ENTERO	0,	ENTERO
11	identificador	\$i=	identificador
12	ENTERO	0,	ENTERO
13	identificador	\$sumaA=	identificador
14	ENTERO	0,	ENTERO
15	identificador	\$sumaB=	identificador
16	ENTERO	0;	ENTERO
17	Palabra reservada	print	print
18	Palabra reservada	(	abre parentesis

Editor

Archivo Run Lexico Sintactico Semantico

```

1 #programa que dice en que rango esta un numero del 1 al 100#
2 main(){
3     print("Dame un numero");
4     Int $i=read(); #lee una variable entera#
5     while($i<=100) {
6         print(i);
7
8         if($i==1){ #compara si la variable es un numero
9             print("es el primer numero");
10        }
11        if($i>1 && $i<50){ #compara si la variable es menor que 50
12            print("el numero es menor que 50");
13            if($i>10 && $i<30){ #compara si la variable es entre 10 y 30
14                print("el numero esta entre 10 y 30");
15            }
16        }
17        if($i>50){ #compara si la variable es mayor que 50
18            print("el numero es mayor a 50");
19            if($i>50 && $i<70){ #compara si la variable es entre 50 y 70
20                print("el numero esta entre 50 y 70");
21            }
22        }
23    }
24 }

```

Consola

Error [id=100, tipo=Caracter no valido ], posicion=114]

ID	Tipo	LEXEMA	DESCRIPCION
2	Palabra reservada	main	main
3	Palabra reservada	(	abre parentesis
4	Palabra reservada	)	cierre parentesis
5	Palabra reservada	{	aLLave
6	Palabra reservada	print	print
7	Palabra reservada	(	abre parentesis
8	Palabra reservada	"Dameunnumero"	String dato
9	Palabra reservada	)	cierre parentesis
10	Palabra reservada	:	punto y coma
11	Palabra reservada	Int	Int
12	identificador	\$i=	identificador
13	Palabra reservada	read	read
14	Palabra reservada	(	abre parentesis
15	Palabra reservada	)	cierre parentesis
16	Palabra reservada	:	punto y coma
17	Palabra reservada	while	while
18	Palabra reservada	(	abre parentesis
19	identificador	\$i<	identificador
20	Palabra reservada	=	igual
21	ENTERO	100)	ENTERO
22	Palabra reservada	{	aLLave
23	Palabra reservada	print	print
24	Palabra reservada	(	abre parentesis

## Calendario de Actividades realizadas.

La actividad se separó en dos partes, la primer semana fue para realizar las investigaciones necesarias para el desarrollo del análisis léxico, así como la revisión continua del documento, con el fin de verificar que el contenido fuera el adecuado y contuviera lo solicitado.

Cabe mencionar que para la semana de investigación, como equipo debatimos los puntos indicados, ya que en nuestra investigación surgieron opiniones diferentes, de esta manera nos dimos a la tarea de realizar anotaciones sobre esto, enseguida cada uno opinaba sobre lo que se anotaba al respecto y daba una conclusión, en cuanto a las pocas investigaciones individuales realizadas, se realizaba una retroalimentación sobre estas y se cuestionaba el porqué, de igual manera cada uno generaba una conclusión y se exponía ante el equipo, esto para no dejar con dudas a ningún integrante del equipo y agilizar el proceso de investigación.

En cuanto a la semana de programación, hubo contratiempos, ya que fue un poco complicado organizarnos para realizar esto, tomamos la opción de programar en parejas y realizar los test necesarios para verificar su funcionalidad.

Al realizar los autómatas, llegamos a la conclusión de que sería mejor generar un autómata general, este identifica nuestras palabras reservadas, números, caracteres, delimitadores, operadores, etc.

Fecha:						
febrero de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	1	2	3
4	5	6	7	8	9	10

Fecha:						
marzo de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
25	26	27	28	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

**Nota: Los días enmarcados de color rojo hacen referencia a la semana de investigación, los días marcados en azul se refiere a la semana de programación.**

A continuación se muestra nuestro calendario de actividades realizadas.

SEMANA	ACTIVIDADES PLANIFICADAS	ACTIVIDADES REALIZADAS	TIEMPO PLANEADO	TIEMPO REAL	DIFERENCIA
<b>SEMANA DE INVESTIGACIÓN</b>					
LUNES 26 DE FEBRERO	INVESTIGACIÓN DEL PUNTO 2 INCISO ABC	INVESTIGACIÓN DEL PUNTO 2 INCISO ABC	4 HRS	4:40 HRS	40 MIN
MARTES 27 DE FEBRERO	INVESTIGACIÓN DEL PUNTO 2 INCISO D Y E	INVESTIGACIÓN DEL PUNTO 2 INCISO D	2 HRS	3:30 HRS	130 HRS
MIERCOLES 28 DE FEBRERO	DESARROLLO DEL PUNTO 3 Y 4	DESARROLLO DEL PUNTO D, 3 Y 4	2 HRS	4:00 HRS	2 HRS
JUEVES 1 DE MARZO	RETROALIMENTACIÓN DEL PUNTO 5 INCISOS ABC	RETROALIMENTACIÓN DEL PUNTO 5 INCISOS ABC	2 HRS	2 HRS	0
VIERNES 2 DE MARZO	RETROALIMENTACIÓN DEL PUNTO 5 INCISOS DEF	RETROALIMENTACIÓN DEL PUNTO 5 INCISOS DEF	2 HRS	2 HRS	0
SÁBADO 3 DE MARZO	CONCLUSIÓN Y ACOMODO DE DOCUMENTO(.pdf)	REVISIÓN DEL DOCUMENTO	2 HRS	3 HRS	60 MIN
DOMINGO 4 DE MARZO	DESARROLLO DEL PUNTO 5 INCISO G Y H	DESARROLLO DEL PUNTO 5 INCISO G Y H	3 HRS	4 HRS	60 MIN
<b>SEMANA DE PROGRAMACIÓN</b>					
LUNES 5 DE MARZO	DESARROLLO DEL PUNTO 5 INCISO IJK	DESARROLLO DEL PUNTO 5 INCISO IJK	3 HRS	5 HRS	2 HRS
MARTES 6 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	REVISIÓN DEL DOCUMENTO	2 HRS	60 MIN	60 MIN
MIERCOLES 7 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	3 HRS	4:30 HRS	130 HRS
JUEVES 8 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	3 HRS	4:00 HRS	60 MIN
VIERNES 9 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	3 HRS	2 HRS	60 MIN
SÁBADO 10 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	3 HRS	2 HRS	60 MIN
DOMINGO 11 DE MARZO	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	DESARROLLO Y CORRELACIÓN DEL PUNTO 1 Y PUNTO 5 INCISO L	3 HRS	2 HRS	60 MIN

## Reporte de incidencias.

NUMERO	FECHA	INCIDENCIA	SOLUCION
1	07-mar-18	Desbordamiento de memoria, para simular el autómata generamos diferentes estados, esto haciendo que vallan de un estado a otro mediante métodos vacíos, pero al ser más de 60 estados con sus respectivos tipos de datos y estructuras de control dentro de cada método, provoco desbordamiento de memoria.	Cambiamos la forma en que hacen las transiciones y los estados, esto mediante otras estructuras de datos existentes en Java, tales como los Map que son estructuras clave-valor, generando un Map un poco personalizado, de tal manera que quede se la siguiente forma: clave-valor-clave donde clave son los estados y el valor son los caracteres para la transición. Además de esto usamos los tipos de datos Enumerados (enum) que se usan para tener comportamiento y datos, estos usados para contener los estados del autómata.
2	08-mar-18	Al momento de leer el archivo no se hacía carácter a carácter, por lo cual se había recurrido a usar Tokenizer o .split, sin embargo esto era erróneo, ya que la idea es poder recorrer el archivo carácter a carácter.	Transformar el archivo a un archivo aleatorio (binario) de esta manera pudimos leer carácter a carácter de una forma más sencilla, y así de esta manera evitar el uso de clases que no se deberían de usar. Cabe recalcar que el archivo de código fuente no se transforma a binario, sino más bien se genera un archivo temp.team para que se pueda hacer la lectura. Se puede decir que es un archivo temporal.
3	08-mar-18	Definimos que las variables de tipo cadenas (String \$var = "das asd a") comenzarían con " y contendrían todo tipo de carácter valido, sin embargo causaba problemas. ya que al igual que anteriormente se podían confundir de un estado a otro. (una transición múltiples estados)	Redujimos la cantidad de caracteres que pueden contener las cadenas, ahora solo estarán definidos de la siguiente manera [A-Z][a-z][0-9]_- aceptando solo esos rangos de símbolos.
4	09-mar-18	Al momento de diseñar el autómata que identifique todas las palabras reservadas se tuvo la problemática de que existían palabras las cuales contenían inicios similares como int, init, if por lo que causo problemas al momento de hacer las transiciones, pues de un mismo estado podíamos dirigirnos a diferentes	Lo que hicimos fue lo más sencillo, dado que el lenguaje lo diseñamos nosotros, optamos por rediseñar y cambiar algunas palabras para que de esta manera no existieran problemas de que una transición nos conduzca a más de otro estado. Esto significa que palabras como int se cambió a Int, init por main, e if se dejó tal cual.

5	10-mar-18	<p>Al momento de hacer las transiciones necesarias para la identificación de los identificadores, al igual que lo ocurrido en palabras reservadas que iniciaban con una letra del alfabeto que iniciaba en otra palabra reservada ocurría exactamente lo mismo un estado que tiene múltiples destinos con la misma letra</p>	<p>Cambiamos el inicio de los identificadores, de tal manera que ahora todos los identificadores comienzan con el símbolo \$, de esta manera podemos hacer que las transiciones sean más sencillas.</p>
6	11-mar-18	<p>Si genera un vacío dentro del archivo binario, es decir coloca un carácter el cual no definimos en ningún momento, dicho carácter es el vacío. Esto provocando errores durante el análisis</p>	<p>Buscamos como se representa ese carácter en char por lo que encontramos que se define de la siguiente manera '\u0000' por lo que lo añadimos a la parte de código que lo omite</p>