



TECNOLÓGICO
NACIONAL DE MÉXICO



SEP
SECRETARÍA DE
EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE AGOSTO - DICIEMBRE 2025
INGENIERÍA EN SISTEMAS COMPUTACIONALES

LENGUAJES Y AUTOMATAS II

GENERACIÓN DE ANÁLISIS DE PROYECTO UNIDAD 3

BAUTISTA TREJO LUZ MARIA - 22210287

CRUZ PATIÑO DIEGO - 22210297

DOCENTE:

Luis Alfonso Gaxiola Vega

04 DE NOVIEMBRE DE 2025

ÍNDICE

INTRODUCCIÓN.....	3
Objetivo General.....	4
Objetivos específicos.....	4
Marco Teórico.....	5
Problemática.....	5
Solución.....	5
Pantallas del programa.....	6
EXPRESIONES REGULARES.....	12
¿Qué es una expresión regular?.....	12
Construcción de expresiones regulares.....	12
¿Cómo funciona una expresión regular?.....	13
Tabla de Tokens.....	14
Reglas Gramaticales.....	17
Autómatas.....	20
Léxico.....	21
Función principal.....	21
Proceso.....	21
Semántico.....	22
Sintáctico.....	23
Características principales de un analizador sintáctico:.....	23
Capturas de pantalla y código.....	25
Árboles Sintácticos.....	30
Conclusiones.....	38
Recomendaciones.....	39
Bibliografía.....	42

ÍNDICE DE TABLAS

Tabla de Tokens.....	15
Árboles Sintácticos.....	31

INDICE DE IMAGENES

Img. 1.1 - Pantalla Principal.....	7
Img. 1.2 - Menú de Exploración de Archivos.....	8
Img. 1.3 - Menú de Exploración de Archivos.....	9
Img. 1.4 - Menú de Exploración de Archivos.....	9
Img. 1.5 - Menú de Selección de Archivos.....	10
Img. 1.6 - Menú Principal con el código cargado.....	10
Img. 1.7 - Programa en ejecución.....	11
Img. 1.8 - Componente del Programa (Salida del código de errores).....	11
Img. 1.9 - Componente de la Tabla que muestra los tokens identificados.....	12
Reglas Gramaticales.....	18
Img. 1.10 - Ejemplo de una tabla de tokens.....	24
Img. 1.11 - Ejemplo de árbol sintáctico.....	25
Img. 1.12 Capturas de pantalla y código.....	26

INTRODUCCIÓN

Este proyecto presenta un analizador de código fuente diseñado para ejecutar las etapas iniciales del proceso de compilación o interpretación. La herramienta desarrollada en Java y con una interfaz gráfica de usuario construida mediante la biblioteca JFlex, Flatlaf, JavaJdk 24, Compiler tools 2.3.7, se centra en proporcionar un análisis léxico, sintáctico y semántico fundamental del código ingresado por el usuario.

El análisis léxico se implementa a través de la tokenización, donde el código fuente se descompone en una secuencia de tokens. Cada token representa una unidad léxica significativa, como identificadores, palabras clave, literales u operadores. Este proceso se rige por un conjunto predefinido de patrones léxicos, aplicados de manera priorizada para resolver posibles ambigüedades.

En el análisis sintáctico se lleva a cabo mediante la verificación de la estructura del código, si bien no se trata de un análisis sintáctico basado en un parecer formal, se implementan reglas heurísticas para validar aspectos como el balanceo de delimitadores y la correcta formación de construcciones sintácticas básicas.

Por otro lado, el análisis semántico se enfoca en la coherencia del código, utilizando una tabla de símbolos para rastrear declaraciones y usos de variables. Se detectan errores relacionados con declaraciones, uso de identificadores no declarados e inconsistencias en el manejo de tipos.

La aplicación ofrece una interfaz intuitiva que facilita la interacción del usuario, la visualización de los resultados del análisis y la gestión del historial de análisis. El historial se almacena en archivos individuales si el usuario lo desea, permitiendo la persistencia y la consulta de análisis previos.

En esencia, este proyecto constituye una herramienta educativa y práctica para comprender los principios fundamentales del análisis de código fuente, simulando las primeras fases de la compilación

Objetivo General

Desarrollar una aplicación de escritorio en Java con interfaz gráfica, que permita realizar análisis léxico, sintáctico y semántico de código fuente, para detectar errores estructurales y semánticos de forma automatizada y didáctica.

Objetivos específicos

1. Implementar un módulo de análisis léxico que identifique y clasifique los distintos tokens del código fuente, incluyendo identificadores, palabras clave, operadores y literales.
2. Desarrollar un analizador sintáctico básico que verifique la correcta estructura gramatical del código, como el balance de delimitadores y la formación de estructuras de control.
3. Construir un analizador semántico que evalúe la coherencia del código en cuanto a declaraciones de variables, tipos de datos, uso e inicialización de identificadores.
4. Diseñar una interfaz gráfica interactiva con las herramientas Java y librerías disponibles que permita al usuario cargar y guardar código, visualizar resultados y proporcionar una lista de tokens y errores en el código analizado.
5. Incorporar funciones de filtrado y persistencia de datos para ofrecer mayor control y personalización en los análisis realizados por el usuario.

Marco Teórico

Problemática

Actualmente los compiladores desempeñan un papel fundamental en el desarrollo de software, ya que son los encargados de traducir el código fuente escrito en lenguajes de alto nivel a un lenguaje máquina que pueda ser comprendido y ejecutado por la computadora. Sin embargo, durante este proceso se presentan diversas dificultades, como la correcta detección y manejo de errores léxicos, sintácticos y semánticos, la optimización del código generado y la compatibilidad con diferentes arquitecturas de hardware y sistemas operativos.

Solución

Para dar solución a esta problemática, se propone el desarrollo de un compilador modular y eficiente para sistemas que basen sus código en lenguaje ensamblador como los sistemas inteligentes y embebidos, dividido en fases claramente definidas: análisis léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización y generación de código objeto y generación de código ensamblador.

Pantallas del programa



Img. 1.1 - Pantalla Principal

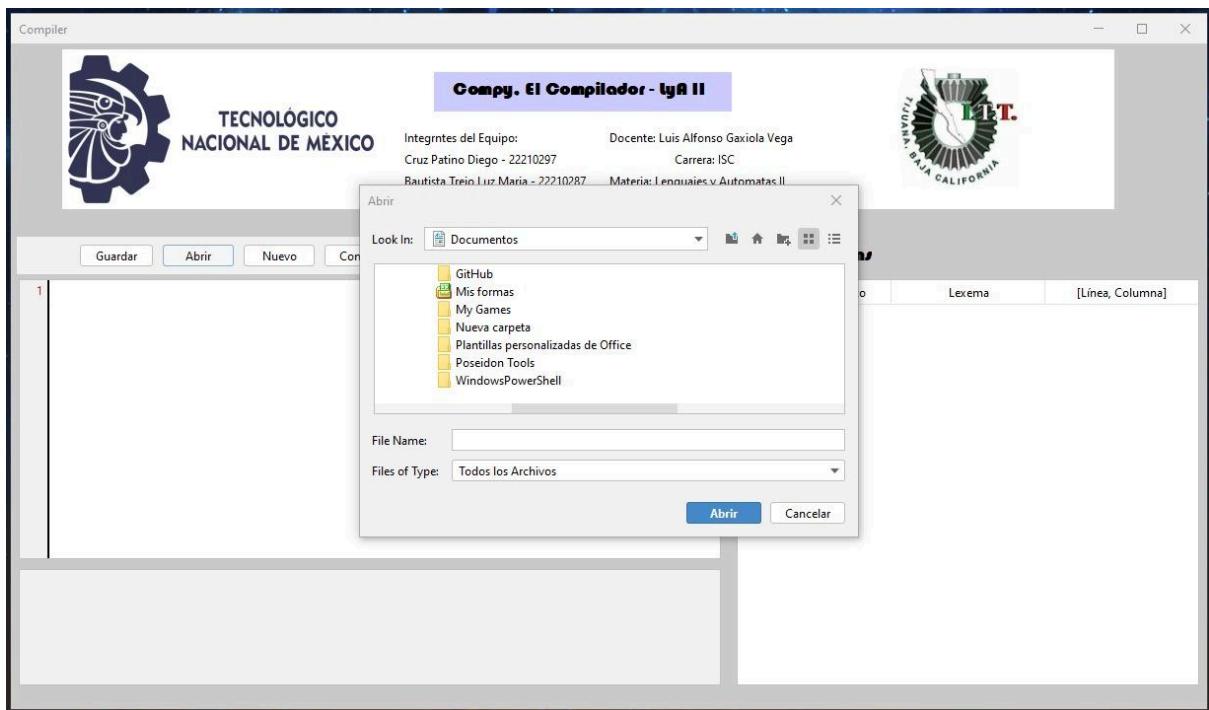
Podemos observar la interfaz en general del compilador realizado, debajo del encabezado tenemos la ventana principal del programa que contiene varios botones de control:

- Guardar: Guarda el código que tengamos.
- Abrir: Nos ayuda abrir un archivo con un código en dado caso que tengamos guardado uno.
- Nuevo:
- Compilar: Ejecuta el código.
- Tripletas: Esto generará un código intermedio optimizado
- CodObj: El botón código objeto muestra el código objeto que se genera al presionar el botón.
- Ensamblador: El botón Ensamblador adapta el código objeto al código en lenguaje ensamblador para él y abre el código en él.

En el centro se observa un área de texto (editor) donde se escribe el código fuente que será compilado.

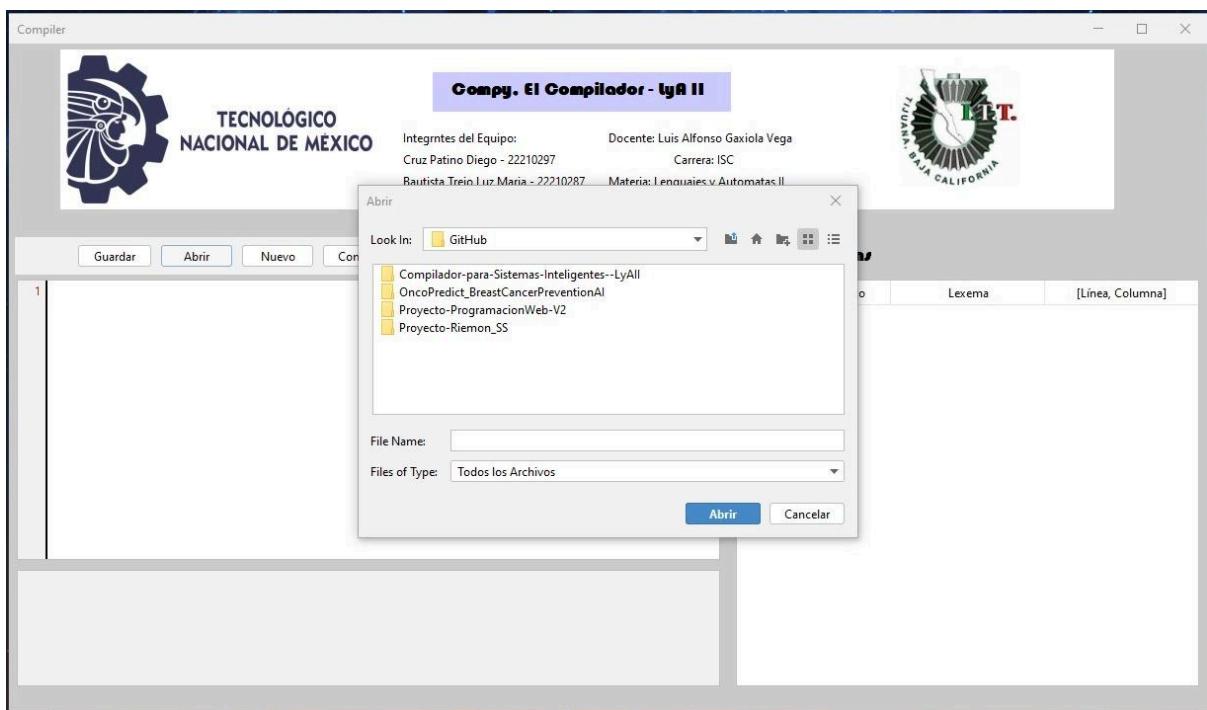
A la derecha hay una sección titulada “Tabla de Tokens”, que tiene columnas para mostrar:

- Componente léxico,
- Lexema,
- [Línea, Columna]
lo que indica que este programa realiza análisis léxico del código.



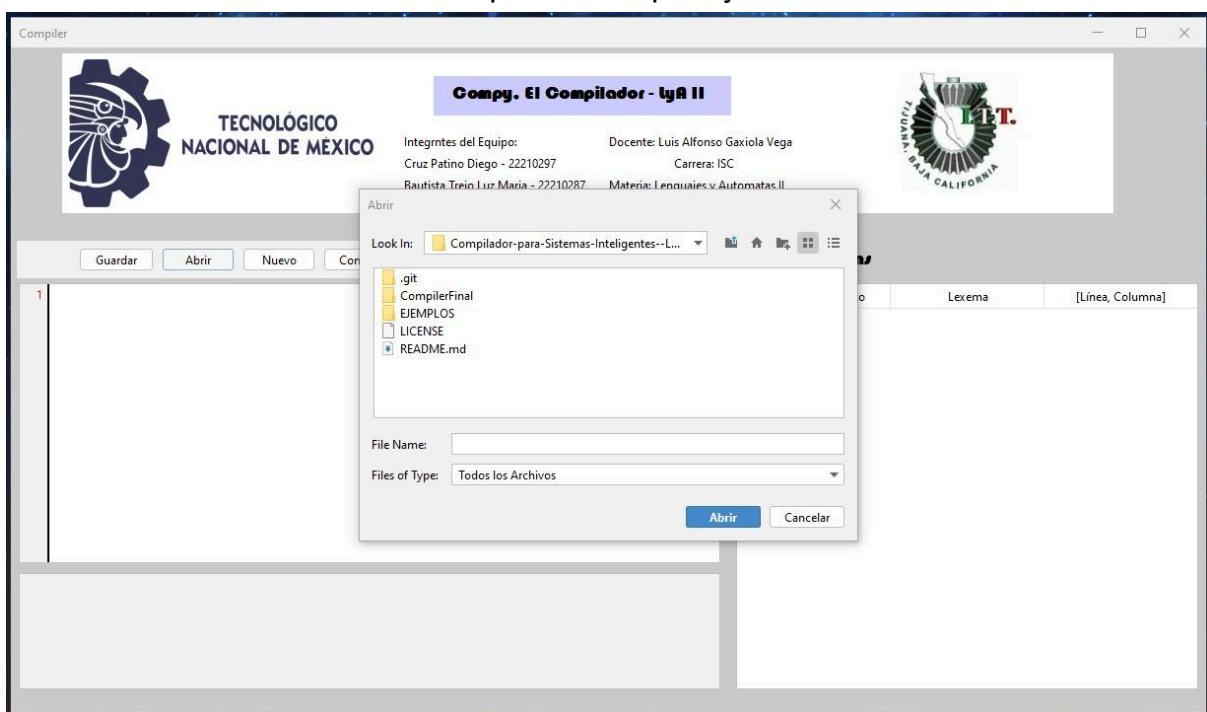
Img. 1.2 - Menu de Exploracion de Archivos

Para comenzar vamos a insertar un código ejemplo desde archivos, comenzamos dando clic en el botón de abrir y seleccionamos la carpeta de GitHub en nuestro caso. También se puede pegar directamente.



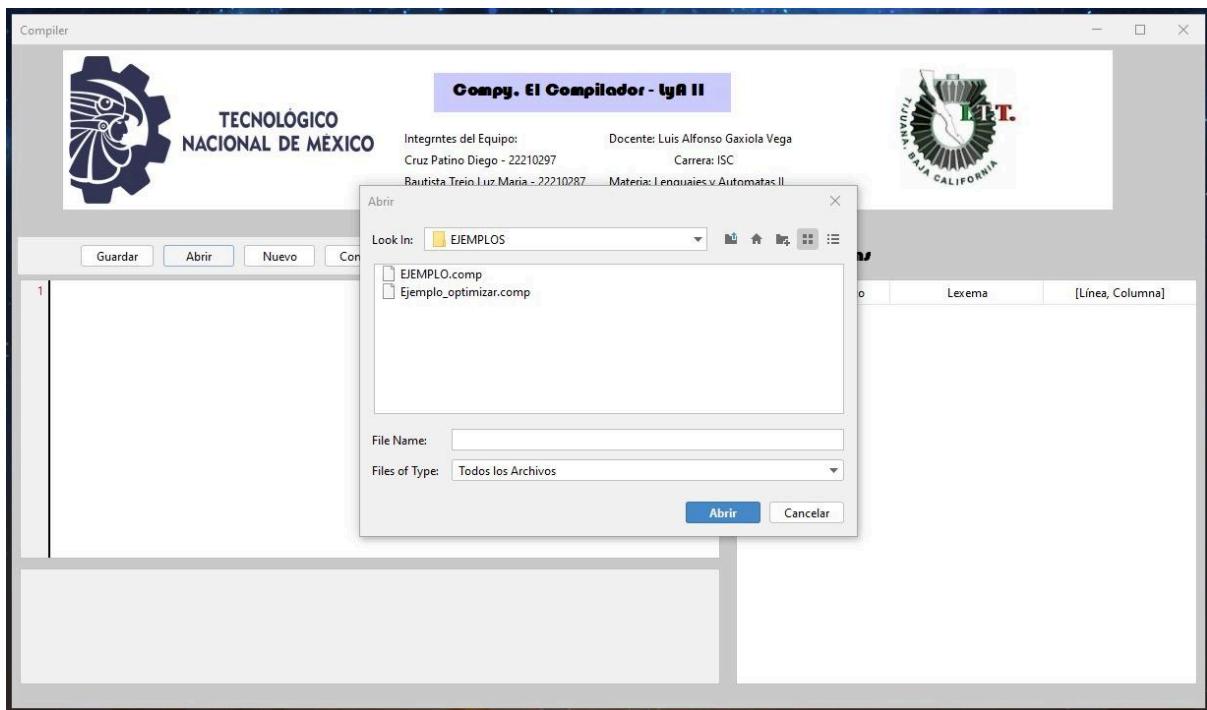
Img. 1.3 - Menu de Exploracion de Archivos

Una vez abierto seleccionamos la primera carpeta y damos en abrir.



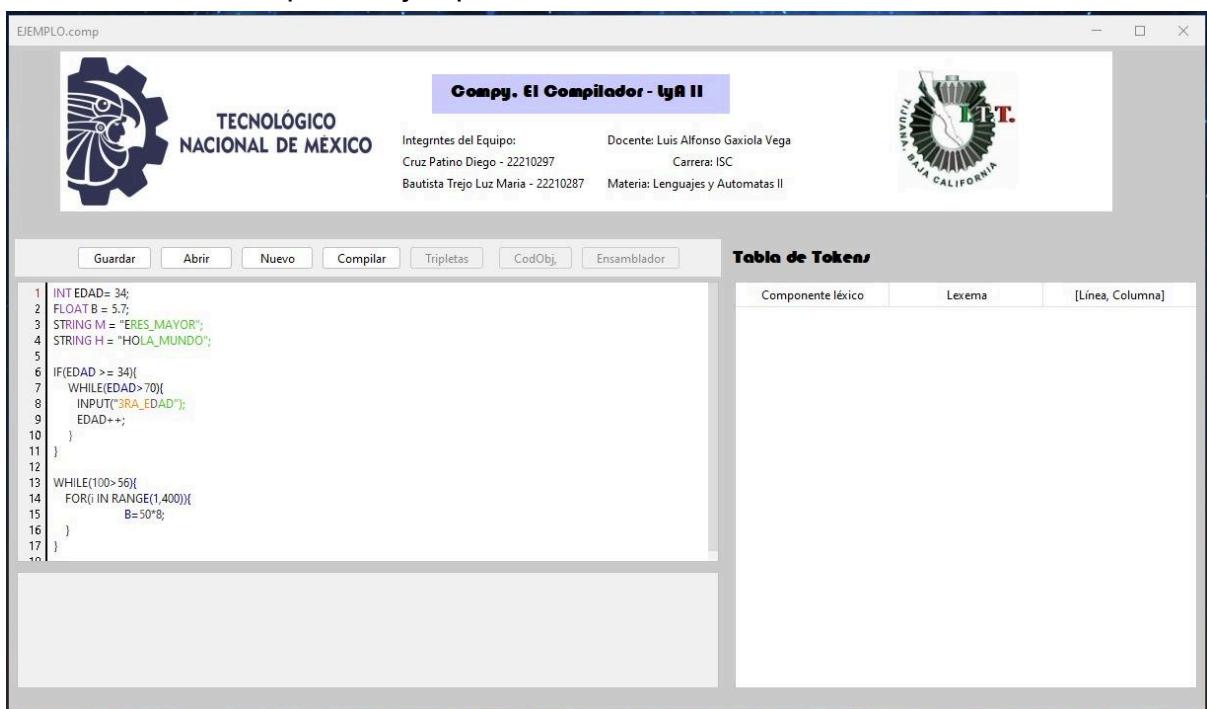
Img. 1.4 - Menu de Exploracion de Archivos

Abrimos la carpeta de EJEMPLOS



Img. 1.5 - Menu de Seleccion de Archivos

Y seleccionamos el primer ejemplo



Img. 1.6 - Menu Principal con el codigo cargado

Que se tiene que ver asi

EJEMPLO.comp

Compy. El Compilador - LyA II

TECNOLÓGICO NACIONAL DE MEXICO

Integrantes del Equipo: Docente: Luis Alfonso Gaxiola Vega
 Cruz Patino Diego - 22210297 Carrera: ISC
 Bautista Trejo Luz María - 22210287 Materia: Lenguajes y Automatas II

Guardar Abrir Nuevo Comilar Tripletas CodObj. Ensamblador

Tabla de Tokens		
Componente léxico	Lexema	[Línea, Columna]
INT	INT	[1, 1]
ID	EDAD	[1, 5]
ASIGNACION	=	[1, 9]
NUMERO	34	[1, 11]
PUNTOCOMA	:	[1, 13]
FLOAT	FLOAT	[2, 1]
ID	B	[2, 7]
ASIGNACION	=	[2, 9]
REAL	5.7	[2, 11]
PUNTOCOMA	:	[2, 14]
STRING	STRING	[3, 1]
ID	M	[3, 8]
ASIGNACION	=	[3, 10]
CADENA	"ERES_MAYOR"	[3, 12]
PUNTOCOMA	:	[3, 24]
STRING	STRING	[4, 1]
ID	H	[4, 8]
ASIGNACION	=	[4, 10]
CADENA	"HOLA_MUNDO"	[4, 12]

```

1 # INT EDAD= 34;
2 FLOAT B = 5.7;
3 STRING M = "ERES_MAYOR";
4 STRING H = "HOLA_MUNDO";
5
6 IF[EDAD >= 34]{
7     WHILE[EDAD>70]{
8         INPUT("3RA_EDAD");
9         EDAD++;
10    }
11 }
12 WHILE[100>56]{
13     FOR[i IN RANGE(1,400)]{
14         B=50*i;
15     }
16 }
17 }

Compilación terminada...
Error semántico : Variable "B" es de tipo FLOAT [15, 2]
La compilación terminó con errores...

```

Img. 1.7 - Programa en ejecucion

Una vez tengamos el código daremos clic en compilar y nos arrojará lo que se muestra en la imagen.

Compilación terminada...

Error semántico : Variable "B" es de tipo FLOAT [15, 2]

La compilación terminó con errores...

Img. 1.8 - Componente del Programa (Salida del codigo de errores)

La parte de abajo nos dirá cuando el compilador haya terminado.

Tabla de Tokens		
Componente léxico	Lexema	[Línea, Columna]
INT	INT	[1, 1]
ID	EDAD	[1, 5]
ASIGNACION	=	[1, 9]
NUMERO	34	[1, 11]
PUNTOCOMA	;	[1, 13]
FLOAT	FLOAT	[2, 1]
ID	B	[2, 7]
ASIGNACION	=	[2, 9]
REAL	5.7	[2, 11]
PUNTOCOMA	;	[2, 14]
STRING	STRING	[3, 1]
ID	M	[3, 8]
ASIGNACION	=	[3, 10]
CADENA	"ERES_MAYOR"	[3, 12]
PUNTOCOMA	;	[3, 24]
STRING	STRING	[4, 1]
ID	H	[4, 8]
ASIGNACION	=	[4, 10]
CADENA	"HOLA_MUNDO"	[4, 12]

```

1 INT EDAD= 34;
2 FLOAT B = 5.7;
3 STRING M = "ERES_MAYOR";
4 STRING H = "HOLA_MUNDO";
5
6 IF(EDAD >= 34){
7   WHILE(EDAD>70){
8     INPUT("3RA_EDAD");
9     EDAD++;
10  }
11 }
12
13 WHILE(100>56){
14   FOR(i IN RANGE(1,400)){
15     B=50*i;
16   }
17 }
18

```

Img. 1.9 - Componente de la Tabla que muestra los tokens identificados

Analizador léxico

Dentro de nuestra pantalla principal podremos observar que existe un tabla que demuestra los Tokens detectados en el código separando cada uno por su componente léxico, su lexema y en qué parte se encuentra en el código separándolo por la línea y la columna en la que se encuentra.

Analizador Sintáctico

En la misma interfaz debajo del área para escribir (el cuadro de texto más grande) se encuentra un cuadro de texto con salidas de tipo OUTPUT que se emite con las mismas reglas gramaticales en caso de errores con la sintaxis en el código mostrando de manera general las posibles soluciones según la gramática implícita en el código.

Analizador Semántico

En el mismo área de OUTPUT que se emiten los errores sintácticos también se muestran todos los errores relacionados con la semántica. Sin embargo, si no se han resuelto todos los problemas de sintaxis en el código, el apartado semántico dará posibles resultados erróneos hasta que la sintaxis sea corregida.

EXPRESIONES REGULARES

¿Qué es una expresión regular?

Las expresiones regulares son un equivalente algebraico para un autómata. Utilizado en muchos lugares como un lenguaje para describir patrones en texto que son sencillos pero muy útiles. Pueden definir exactamente los mismos lenguajes que los autómatas pueden describir: Lenguajes regulares. Además, ofrecen algo que los autómatas no: Manera declarativa de expresar las cadenas que queremos aceptar.

Sirven como lenguaje de entrada a muchos sistemas que procesan cadenas tales como:

- Comandos de búsqueda, e.g., grep de UNIX
- Sistemas de formateo de texto: Usando notación de tipo expresión regular para describir patrones.
- Convierte la expresión regular a un DFA o un NFA y simula el autómata en el archivo de búsqueda
- Generadores de analizadores-léxicos. Como Lex o Flex.
- Los analizadores léxicos son parte de un compilador. Dividen el programa fuente en unidades lógicas (tokens). Tokens como while, números, signos (+, -, <, etc.)
- Produce un DFA que reconoce el token.

Construcción de expresiones regulares

Si E es una expresión regular, entonces $L(E)$ denota el lenguaje que define E . Las expresiones se construyen de la manera siguiente:

1. Las constantes y \emptyset son expresiones regulares que representan a los lenguajes $L() = \emptyset$ y $L(\emptyset) = \emptyset$ respectivamente.
2. Si a es un símbolo, entonces a es una expresión regular que representan al lenguaje: $L(a) = \{a\}$.
1. Si E y F son expresiones regulares, entonces $E + F$ también lo es denotando la unión de $L(E)$ y $L(F)$. $L(E + F) = L(E) \cup L(F)$.
2. Si E y F son expresiones regulares, entonces EF también lo es denotando la concatenación de $L(E)$ y $L(F)$. $L(EF) = L(E)L(F)$

¿Cómo funciona una expresión regular?

Una expresión regular puede estar formada, o bien exclusivamente por caracteres normales (como abc), o bien por una combinación de caracteres normales y metacaracteres (como ab*c). Los metacaracteres describen ciertas construcciones o disposiciones de caracteres: por ejemplo, si un carácter debe estar en el inicio de la línea o si un carácter solo debe o puede aparecer exactamente una vez, más veces o menos. Ambos ejemplos de expresiones regulares funcionan, por ejemplo, de la siguiente manera.

abc: El patrón regex sencillo abc requiere una coincidencia exacta. Por tanto, se buscarán cadenas de caracteres que no solo contengan los caracteres “abc”, sino que también aparezcan en ese orden. Una pregunta como “¿Conoces la plaza ABC?” ofrece la coincidencia buscada por esta expresión.

ab*c. Las expresiones regulares con caracteres especiales funcionan de manera diferente, ya que no solo se buscarán coincidencias exactas, si no también escenarios especiales. En este caso, el asterisco hace que la búsqueda se centre en cadenas de caracteres que empiecen por la letra “a” y que terminen por la letra “c” y entremedias cuenten con cualquier número de caracteres “b”. Así se mostrará como coincidencia tanto “abc”, como la cadena de caracteres “abbbbc” y “cbbabbcba”.

Además, cada regex se puede vincular a una acción concreta, como la ya mencionada “Reemplazar”. Esta acción se ejecuta en todos los lugares en los que se detecta la expresión regular, es decir, en todos los puntos en los que haya una coincidencia similar a la de los ejemplos.

Tabla de Tokens

#	Nombre del Token	Palabras Reservadas	Lexemas
1	TerminadorDeLinea	\r, \n, \r\n	\r \n \r\n
2	EntradaDeCaracter	^\r\n	^\r\n
3	EspacioEnBlanco	\r, \n, \r\n, \t\f	{TerminadorDeLinea} \t\f
4	ComentarioTradicio nal		/* [^*] ~*/ /* *+ */
5	FinDeLineaComenta rio		/* {EntradaDeCaracter}* {TerminadorDeLinea} ?
6	ContenidoComentari o		([^*] *+ [/])*
7	ComentarioDeDocu mentacion		/**/ {ContenidoComentari o} *+ /
8	Comentario		{ComentarioTradicio nal} {FinDeLineaComenta rio} {ComentarioDeDocu mentacion}
9	Letra		[A-Za-zÑñ_ÁÉÍÓÚáéí óúü]
10	Dígito		0 [1-9][0-9]*
11	Identificador		{Letra} ({Letra} {Dígito})*
12	Número		{Dígito}
13	Real		{Dígito}."{Dígito}
14	SUMA	+	+"
15	RESTA	-	"_"
16	DIVISION	/	"/"
17	MULTIPLICACION	*	"*"
18	IGUAL	==	"=="

19	DIFERENTE	<code>!=</code>	" <code>!=</code> "
20	MAYORQUE	<code>></code>	" <code>></code> "
21	MENORQUE	<code><</code>	" <code><</code> "
22	MAYORIGUALQUE	<code>>=</code>	" <code>>=</code> "
23	MENORIGUALQUE	<code><=</code>	" <code><=</code> "
24	PUNTO	<code>.</code>	" <code>.</code> "
25	COMA	<code>,</code>	" <code>,</code> "
26	DOSPUNTOS	<code>:</code>	" <code>:</code> "
27	PUNTOCOMA	<code>;</code>	" <code>;</code> "
28	COMILLASSIMPLE	<code>\</code>	'
29	CADENA		<code>\'' [a-zA-Z0-9_-.]* \'</code>
30	ASIGNACION	<code>=</code>	" <code>=</code> "
31	COMILLADOBLE	<code>"</code>	<code>\\"</code>
32	PARENTESISABIERTO	<code>(</code>	<code>\(</code>
33	PARENTESISCERRADO	<code>)</code>	<code>\)</code>
34	LLAVEABIERTO	<code>{</code>	<code>\{</code>
35	LLAVECERRADO	<code>}</code>	<code>\}</code>
36	CORCHETEABIERTO	<code>[</code>	<code>\[</code>
37	CORCHETECERRADO	<code>]</code>	<code>\]</code>
38	INCREMENTO	<code>++</code>	" <code>++</code> "
39	DECREMENTO	<code>--</code>	" <code>--</code> "
40	IMPORT	IMPORT, import, Import	" <code>IMPORT</code> " " <code>import</code> " " <code>Import</code> "
41	DEF	DEF, def, Def	" <code>DEF</code> " " <code>def</code> " " <code>Def</code> "
42	CLASS	CLASS, class, Class	" <code>CLASS</code> " " <code>class</code> " " <code>Class</code> "
43	IF	IF, if, If	" <code>IF</code> " " <code>if</code> " " <code>If</code> "
44	ELSE	ELSE, else, Else	" <code>ELSE</code> " " <code>else</code> " " <code>Else</code> "

45	FOR	FOR, for, For	"FOR" "for" "For"
46	IN	IN, in, In	"IN" "in" "In"
47	RANGE	RANGE, range, Range	"RANGE" "range" "Range"
48	SELF	SELF, self, Self	"SELF" "self" "Self"
49	WHILE	WHILE, while, While	"WHILE" "while" "While"
50	TRY	TRY, try, Try	"TRY" "try" "Try"
51	EXCEPT	EXCEPT, except, Except	"EXCEPT" "except" "Except"
52	RETURN	RETURN, return, Return	"RETURN" "return" "Return"
53	BREAK	BREAK, break, Break	"BREAK" "break" "Break"
54	NEXT	NEXT, next, Next	"NEXT" "next" "Next"
55	INPUT	INPUT, input, Input	"INPUT" "input" "Input"
56	OUTPUT	OUTPUT, output, Output	"OUTPUT" "output" "Output"
57	PRINT	PRINT, print, Print	"PRINT" "print" "Print"
58	INT	INT, int, Int	"INT" "int" "Int"
59	FLOAT	FLOAT, float, Float	"FLOAT" "float" "Float"
60	BOOLEAN	BOOLEAN, boolean, Boolean	"BOOLEAN" "boolean" "Boolean"
61	STRING	STRING, string, String	"STRING" "string" "String"
62	TRUE	TRUE, true, True	"TRUE" "true" "True"
63	FALSE	FALSE, false, False	"FALSE" "false" "False"
64	POWER	POWER, power, Power	"POWER" "power" "Power"
65	SQRT	SQRT, sqrt, Sqrt	"SQRT" "sqrt" "Sqrt"


```

//ERRORES operacion
gramatica.group("OPERACION_ER", "NUMERO (SUMA|RESTA|MULTIPLICACION|DIVISION)",2,"ERROR_SINTACTICO: se necesita un minimo de 2 valores para realizar la operacion [#, %]");
gramatica.group("OPERACION_ER", "(SUMA|RESTA|MULTIPLICACION|DIVISION) NUMERO",2,"ERROR_SINTACTICO: se necesita un minimo de 2 valores para realizar la operacion [#, %]");
gramatica.group("OPERACION_ER", "ID (SUMA|RESTA|MULTIPLICACION|DIVISION)",2,"ERROR_SINTACTICO: se necesita un minimo de 2 valores para realizar la operacion [#, %]");
gramatica.group("OPERACION_ER", "(SUMA|RESTA|MULTIPLICACION|DIVISION) ID",2,"ERROR_SINTACTICO: se necesita un minimo de 2 valores para realizar la operacion [#, %]");

gramatica.group("DECL_FLOAT", "FLOAT ID PUNTOCOMA",identProd);
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION ID PUNTOCOMA",identProd);
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION REAL PUNTOCOMA",identProd);
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION OPERACION PUNTOCOMA",identProd);

gramatica.group("DECL_INT", "INT ID PUNTOCOMA",identProd);
gramatica.group("DECL_INT", "INT ID ASIGNACION ID PUNTOCOMA",identProd);
gramatica.group("DECL_INT", "INT ID ASIGNACION NUMERO PUNTOCOMA",identProd);
gramatica.group("DECL_INT", "INT ID ASIGNACION OPERACION PUNTOCOMA",identProd);

gramatica.group("DECL_BOOL", "BOOLEAN ID PUNTOCOMA",identProd);
gramatica.group("DECL_BOOL", "BOOLEAN ID ASIGNACION ID PUNTOCOMA",identProd);
gramatica.group("DECL_BOOL", "BOOLEAN ID ASIGNACION (TRUE|FALSE) PUNTOCOMA",identProd);

gramatica.group("DECL_STRING", "STRING ID PUNTOCOMA",identProd);
gramatica.group("DECL_STRING", "STRING ID ASIGNACION ID PUNTOCOMA",identProd);
gramatica.group("DECL_STRING", "STRING ID ASIGNACION CADENA PUNTOCOMA",identProd);
//ERRORES SINTACTICOS-----
//POSIBLES ERRORES AL DECLARAR UNA VARIABLE INT O FLOAT
gramatica.group("DECL_INT", "INT ID ASIGNACION PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA ASIGNAR UN VALOR A LA VARIABLE [#, %]");
gramatica.group("DECL_INT", "INT ID NUMERO PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID ID PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID NUMERO",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID ID",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID OPERACION PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID OPERACION",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID ASIGNACION ID",2,"ERROR_SINTACTICO: PUNTOCOMA(); NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ID ASIGNACION REAL",2,"ERROR_SINTACTICO: VALOR NO ENTERO [#, %]");

gramatica.group("DECL_INT", "INT ID ASIGNACION OPERACION",2,"ERROR_SINTACTICO: PUNTOCOMA(); NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ASIGNACION NUMERO PUNTOCOMA",2,"ERROR_SINTACTICO: ID NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ASIGNACION ID PUNTOCOMA",2,"ERROR_SINTACTICO: ID NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_INT", "INT ASIGNACION OPERACION PUNTOCOMA",2,"ERROR_SINTACTICO: ID NO AGREGADO EN LA DECLARACION [#, %]");

//POSIBLES ERRORES AL DECLARAR UN FLOAT
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA ASIGNAR UN VALOR A LA VARIABLE [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT ID REAL PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT ID REAL",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION REAL",2,"ERROR_SINTACTICO: PUNTOCOMA(); NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT ASIGNACION REAL PUNTOCOMA",2,"ERROR_SINTACTICO: ID NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT ID ASIGNACION NUMERO PUNTOCOMA",2,"Error sintactico: Valor float sin punto decimal [#, %]");

//POSIBLES ERRORES AL DECLARAR UN FLOAT
gramatica.group("DECL_STRING", "STRING ID ASIGNACION PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA ASIGNAR UN VALOR A LA VARIABLE [#, %]");
gramatica.group("DECL_STRING", "STRING ID CADENA PUNTOCOMA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_STRING", "STRING ID CADENA",2,"ERROR_SINTACTICO: FALTA DEL TOKEN DE ASIGNACION EN LA DECLARACION [#, %]");
gramatica.group("DECL_STRING", "STRING ID ASIGNACION CADENA",2,"ERROR_SINTACTICO: PUNTOCOMA(); NO AGREGADO EN LA DECLARACION [#, %]");
gramatica.group("DECL_STRING", "STRING ASIGNACION CADENA PUNTOCOMA",2,"ERROR_SINTACTICO: ID NO AGREGADO EN LA DECLARACION [#, %]");

//ERRORES SEMANTICOS DE VARIABLES -----
gramatica.group("RESERV_INDEB", "(STRING|INT|FLOAT|BOOLEAN) (IMPORT|DEF|CLASS|IF|ELSE|FOR|IN|WHILE|RETURN)",2, "ERROR SEMANTICO \\(): USO INDEBIDO DE PALABRAS RESERVADAS");

gramatica.group("ERROR_OP_STRING", "(SUMA|RESTA|MULTIPLICACION|DIVISION) CADENA",2, "ERROR SEMANTICO \\(): OPERACION NO PERMITIDA PARA CADENA [#, %]");
gramatica.group("ERROR_OP_STRING", "CADENA (SUMA|RESTA|MULTIPLICACION|DIVISION)",2, "ERROR SEMANTICO \\(): OPERACION NO PERMITIDA PARA CADENA [#, %]");
gramatica.group("ERROR_OP_BOOLEAN", "(SUMA|RESTA|MULTIPLICACION|DIVISION) (TRUE|FALSE)",2, "ERROR SEMANTICO \\(): OPERACION NO PERMITIDA PARA BOOLEANO [#, %]");
gramatica.group("ERROR_OP_BOOLEAN", "(TRUE|FALSE) (SUMA|RESTA|MULTIPLICACION|DIVISION)",2, "ERROR SEMANTICO \\(): OPERACION NO PERMITIDA PARA BOOLEANO [#, %]");

gramatica.group("DECL_INT", "(INT ID ASIGNACION REAL PUNTOCOMA)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES ENTERO [#, %]");
gramatica.group("DECL_INT", "(INT ID ASIGNACION CADENA)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES ENTERO [#, %]");
gramatica.group("DECL_INT", "(INT ID ASIGNACION (TRUE|FALSE))",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES ENTERO [#, %]");
gramatica.group("DECL_INT", "INT",2,"ERROR");

gramatica.group("DECL_FLOAT", "(FLOAT ID ASIGNACION CADENA)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES DECIMAL [#, %]");
gramatica.group("DECL_FLOAT", "(FLOAT ID ASIGNACION (TRUE|FALSE))",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES DECIMAL [#, %]");
gramatica.group("DECL_FLOAT", "(FLOAT ID ASIGNACION NUMERO PUNTOCOMA)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES DECIMAL [#, %]");
gramatica.group("DECL_FLOAT", "FLOAT",2,"ERROR");

gramatica.group("DECL_STRING", "(STRING ID ASIGNACION NUMERO)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES CADENA [#, %]");
gramatica.group("DECL_STRING", "(STRING ID ASIGNACION (TRUE|FALSE))",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES CADENA [#, %]");
gramatica.group("DECL_STRING", "STRING",2,"ERROR");

gramatica.group("ERROR_ASIG_BOOL", "(BOOLEAN ID ASIGNACION NUMERO)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES BOOLEANO [#, %]");
gramatica.group("ERROR_ASIG_BOOL", "(BOOLEAN ID ASIGNACION (TRUE|FALSE))",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES BOOLEANO [#, %]");
gramatica.group("ERROR_ASIG_BOOL", "(BOOLEAN ID ASIGNACION CADENA)",2, "ERROR SEMANTICO \\(): VALOR ASIGNADO NO ES BOOLEANO [#, %]");

//ASIGNACION DE UN ID
gramatica.group("PROD_ASIG", "ID ASIGNACION (CADENA|REAL|NUMERO|TRUE|FALSE) PUNTOCOMA",asigProd);
gramatica.group("PROD_ASIG", "ID ASIGNACION OPERACION PUNTOCOMA",asigProd);
gramatica.group("PROD_ASIG_ID", "ID ASIGNACION ID PUNTOCOMA",asigProdConID);

```

```

//-----CONDICION-----
//FORMAS CORRECTAS DE CREAR UNA CONDICION
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) NUMERO");
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) OPERACION");
gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) OPERACION");

gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) NUMERO");
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) REAL");
gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) REAL");
gramatica.group("CONDICION", "(TRUE|FALSE) (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) (TRUE|FALSE)");
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ID",comparaProdDer);
gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ID",comparaProdDer);
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) NUMERO",comparaProdIzq);
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) OPERACION",comparaProdIzq);
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) REAL",comparaProdIzq);
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ID",comparaProdIzq);
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) (TRUE|FALSE)",comparaProdIzq);

gramatica.group("CONDICION", "CONDICION (AND|OR) CONDICION");

//ERRORES SEMANTICOS
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) CADENA",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");
gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) CADENA",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");
gramatica.group("CONDICION", "CADENA (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) NUMERO",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");
gramatica.group("CONDICION", "CADENA (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) REAL",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) (TRUE|FALSE)",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");
gramatica.group("CONDICION", "REAL (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) (TRUE|FALSE)",2,"ERROR_SEMANTICO \\(): DATOS INCOMPATIBLES PARA SU COMPARACION");

//-----WHILE Y IF-----
//FORMAS CORRECTAS DE DECLARAR UN IF
gramatica.group("INSTR_IF", "IF PARENTESISABIERTO CONDICION PARENTESISCERRADO LLAVEABIERTO",true,ifProd);
//FORMAS CORRECTAS DE DECLARAR UN WHILE
gramatica.group("INSTR_WHILE", "WHILE PARENTESISABIERTO CONDICION PARENTESISCERRADO LLAVEABIERTO",whileProd);
//POSIBLES ERRORES AL DECLARAR UN IF
gramatica.group("INSTR_IF", "IF PARENTESISABIERTO PARENTESISCERRADO LLAVEABIERTO",true,4,"ERROR_SINTACTICO: FALTA LA CONDICION [#,%]");
gramatica.group("INSTR_IF", "IF CONDICION PARENTESISCERRADO LLAVEABIERTO",true,4,"ERROR_SINTACTICO: FALTA EL PARENTESIS ABIERTO EN LA CONDICION [#,%]");
gramatica.group("INSTR_IF", "IF PARENTESISABIERTO CONDICION PARENTESISCERRADO",true,4,"ERROR_SINTACTICO: FALTA DE LLAVE DE APERTURA [#,%]");
gramatica.finalLineColumn();
gramatica.group("INSTR_IF", "IF PARENTESISABIERTO CONDICION",true,4,"ERROR_SINTACTICO: ERROR EN LA CONDICION O FALTA DEL PARENTESIS [#,%]");
gramatica.initialLineColumn();
gramatica.group("INSTR_IF", "IF",2,"ERROR");

//POSIBLES ERRORES DE WHILE
gramatica.group("INSTR_WHILE", "WHILE PARENTESISABIERTO PARENTESISCERRADO",true,4,"ERROR_SINTACTICO: FALTA LA CONDICION [#,%]");
gramatica.group("INSTR_WHILE", "WHILE CONDICION PARENTESISCERRADO",true,4,"ERROR_SINTACTICO: FALTA EL PARENTESIS ABIERTO EN LA CONDICION [#,%]");
gramatica.group("INSTR_WHILE", "WHILE PARENTESISABIERTO CONDICION PARENTESISCERRADO",true,4,"ERROR_SINTACTICO: FALTA DE LLAVE DE APERTURA [#,%]");
gramatica.finalLineColumn();
gramatica.group("INSTR_WHILE", "WHILE PARENTESISABIERTO CONDICION",true,4,"ERROR_SINTACTICO: ERROR EN LA CONDICION O FALTA DEL PARENTESIS [#,%]");
gramatica.initialLineColumn();
gramatica.group("INSTR_WHILE", "WHILE",2,"ERROR WHILE");

//POSIBLES ERRORES EN LAS CODICIONES
gramatica.group("CONDICION", "NUMERO (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ",2,"ERROR_SINTACTICO: ERROR EN LA CONDICION [#,%]");
gramatica.group("CONDICION", "ID (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ",2,"ERROR_SINTACTICO: ERROR EN LA CONDICION [#,%]");
gramatica.group("CONDICION", " (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) NUMERO ",2,"ERROR_SINTACTICO: ERROR EN LA CONDICION [#,%]");
gramatica.group("CONDICION", " (IGUAL|DIFERENTE|MAYORQUE|MENORQUE|MAYORIGUALQUE|MENORIGUALQUE) ID ",2,"ERROR_SINTACTICO: ERROR EN LA CONDICION [#,%]");
gramatica.group("CONDICION", "CONDICION (AND|OR)",2,"ERROR_SINTACTICO: ERROR EN LA CONDICION [#,%]");

//INPUT-----
//FORMA CORRECTA
gramatica.group("INSTR_INPUT", "INPUT PARENTESISABIERTO CADENA PARENTESISCERRADO PUNTOCOMA ",funcProd);
gramatica.group("INSTR_INPUT", "INPUT PARENTESISABIERTO ID PARENTESISCERRADO PUNTOCOMA ",funcProd);

gramatica.group("INSTR_OUTPUT", "OUTPUT PARENTESISABIERTO CADENA PARENTESISCERRADO PUNTOCOMA ",funcProd);
gramatica.group("INSTR_OUTPUT", "OUTPUT PARENTESISABIERTO ID PARENTESISCERRADO PUNTOCOMA ",funcProd);

//ERRORES SINTACTICOS
gramatica.group("INSTR_INPUT", "INPUT PARENTESISABIERTO CADENA PARENTESISCERRADO",2,"ERROR_SINTACTICO \\(): FALTA DEL TOKEN (:) [#,%]");
gramatica.group("INSTR_INPUT", "INPUT CADENA PARENTESISCERRADO PUNTOCOMA",2,"ERROR_SINTACTICO \\(): FALTA DEL PARENTESIS ABIERTO [#,%]");

//ERROR SEMANTICO
gramatica.group("INSTR_INPUT", "INPUT PARENTESISABIERTO (NUMERO|REAL) PARENTESISCERRADO PUNTOCOMA ",2,"ERROR_SEMANTICO \\(): VALOR INVALIDO EN INPUT[#,%]");
gramatica.group("INSTR_INPUT", "INPUT",2,"ERROR INPUT");
//ERRORES SINTACTICOS
gramatica.group("INSTR_INPUT", "OUTPUT PARENTESISABIERTO CADENA PARENTESISCERRADO",2,"ERROR_SINTACTICO \\(): FALTA DEL TOKEN (:) [#,%]");
gramatica.group("INSTR_INPUT", "OUTPUT CADENA PARENTESISCERRADO PUNTOCOMA",2,"ERROR_SINTACTICO \\(): FALTA DEL PARENTESIS ABIERTO [#,%]");

//ERROR SEMANTICO
gramatica.group("INSTR_INPUT", "OUTPUT PARENTESISABIERTO (NUMERO|REAL) PARENTESISCERRADO PUNTOCOMA ",2,"ERROR_SEMANTICO \\(): VALOR INVALIDO EN INPUT[#,%]");
gramatica.group("INSTR_INPUT", "OUTPUT",2,"ERROR INPUT");

//FORMAS DE CREAR UNA FUNCION CORRECTAMENTE
gramatica.group("PARAMETROS", "ID (COMA ID)+");
gramatica.group("PARAMETRO", "ID");
gramatica.group("FUNCION", "DEF ID PARENTESISABIERTO (PARAMETRO | PARAMETROS) ? PARENTESISCERRADO ", true);

gramatica.group("LLAMAR_FUNCION", "ID PARENTESISABIERTO (PARAMETRO | PARAMETROS) ? PARENTESISCERRADO ", true);

//posibles errores al declarar una funcion
gramatica.group("FUNCION", "DEF ID (PARAMETRO | PARAMETROS) ? PARENTESISCERRADO ", true,3,"ERROR_SINTACTICO: FALTA PARENTESIS ABIERTO [#,%]");
gramatica.group("FUNCION", "DEF ID PARENTESISABIERTO (PARAMETRO | PARAMETROS) ? ", true,3,"ERROR_SINTACTICO: FALTA PARENTESIS CERRADO O UN PARAMETRO ESTA MAL DECLARADO [#,%]");
gramatica.group("FUNCION", "DEF PARENTESISABIERTO (PARAMETRO | PARAMETROS) ? PARENTESISCERRADO ", true,3,"ERROR_SINTACTICO: FALTA NOMBRAR LA FUNCION [#,%]");


```

Autómatas

¿Que es un autómata?

Un autómata es un modelo matemático para una máquina de estado finito, en el que dada una entrada de símbolos, busca mediante una serie de estados de acuerdo a una función de transición. Esta función de transición indica a qué estado cambiar dados el estado actual y el símbolo leído. En la cual se clasifican de distintas formas como:

- Autómatas finitos(AF)

Consiste en grafos como los diagramas de transición de estados con algunas diferencias. Son reconocedores solo dicen si o no en relación con cada posible cadena de entrada.

- Autómata Finito Determinístas (AFD)

Es un autómata finito que además es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

- Autómata Finito No Deterministas (AFN)

Es un autómata finito que, a diferencia de los autómatas finitos deterministas, posee al menos un estado, en el que para un símbolo del alfabeto, existe más de una transición posible.

¿Cómo funciona un autómata?

Como se mencionó anteriormente un autómata busca indicar a qué estado al que va cambia el estado actual junto a su símbolo, esta función se puede aplicar en distintas cuestiones como:

- La realización de un analizador léxico, concretamente en el componente del compilador que separa el texto de entrada en unidades lógicas, tal como identificadores, id y signos.
- Software para diseñar y probar el comportamiento de circuitos digitales.
- Software para verificar sistemas de todo tipo que tengan un número finito de estados diferentes.
- Simulador robótico con lenguaje de programación para robots.
- Generador de analizador sintáctico (YACC, JAVACC).
- Investigación y desarrollo.

Léxico

Un analizador léxico, también conocido como lexer o scanner, es una parte fundamental en el proceso de compilación o interpretación de un lenguaje de programación. Su tarea principal es tomar el código fuente (escrito por un programador en un lenguaje de alto nivel) y dividirlo en componentes más pequeños llamados tokens.

Función principal

El analizador léxico lee el código fuente de manera secuencial y lo descompone en unidades significativas (tokens), que son los elementos básicos del lenguaje. Estos tokens pueden ser, por ejemplo:

- Palabras clave (como `if`, `while`, `for`, `int`, etc.)
- Identificadores (nombres de variables, funciones, clases, etc.)
- Operadores (como `+`, `-`, `*`, `=`, etc.)
- Literales (números, cadenas de texto)
- Delimitadores (como paréntesis `()`, llaves `{}`, comillas `""`, etc.)
- Comentarios (que se ignoran durante la compilación)

Proceso

1. **Entrada:** El analizador léxico recibe el código fuente como texto plano.
2. **Análisis:** Analiza el texto para reconocer las secuencias de caracteres que corresponden a los tokens válidos del lenguaje.
3. **Salida:** Devuelve una secuencia de tokens que representan el código fuente de manera estructurada.

El analizador léxico podría descomponerlo en los siguientes tokens:

- `int` → Palabra clave
- `a` → Identificador
- `=` → Operador de asignación
- `10` → Literal numérico
- `;` → Delimitador (punto y coma)

Semántico

Es una técnica fundamental del procesamiento del lenguaje natural (PLN) que se enfoca en interpretar el significado de las palabras, frases y textos completos, y no solo su estructura o gramática.

Las herramientas basadas en el análisis semántico pueden ayudar a las empresas a extraer automáticamente información útil de datos no estructurados, como correos electrónicos, solicitudes de asistencia y comentarios de los consumidores.

Sintáctico

Un **analizador sintáctico** (también conocido como parser) es un componente esencial en el campo de la informática, especialmente en la programación y el procesamiento de lenguajes. Su función principal es analizar la estructura de una secuencia de símbolos (como un código fuente) para determinar su conformidad con las reglas de una gramática formal.

En términos más simples, un analizador sintáctico recibe una cadena de texto (por ejemplo, código de un lenguaje de programación o una frase en un lenguaje natural) y verifica si está estructurada de acuerdo con las reglas gramaticales del lenguaje que se está utilizando. Si la cadena es válida según esas reglas, el analizador la descompone en componentes más pequeños y establece una estructura jerárquica que refleja la organización sintáctica del texto.

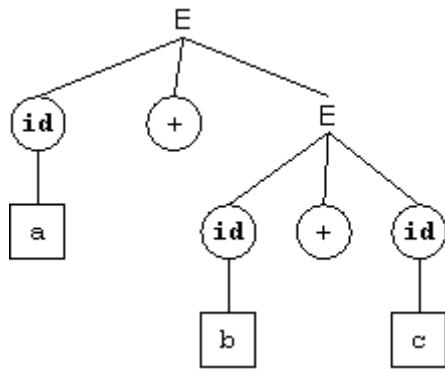
Características principales de un analizador sintáctico:

1. **Entrada:** Un conjunto de tokens (elementos léxicos) generados previamente por un analizador léxico (lexer). Estos tokens son las unidades básicas del lenguaje.

Token	Expresión Regular	Lexema(s) Ejemplo(s)
Palabra clave	\b(int	float
Identificador	[a-zA-Z_][a-zA-Z0-9_]*	x, myVariable, totalAmount
Literal numérico	\b\d+(\.\d*)?\b	5, 123, 3.14, 0.001
Operador aritmético	[\+\-*/%]	+, -, *, /, %
Operador de comparación	`==	!=
Operador lógico	`&&	
Delimitador (punto y coma)	;	;
Delimitador (paréntesis)	`()`
Delimitador (corchete)	`{	
Comillas (cadena)	\".*?\"	"Hola Mundo", "Variable1"
Comentario de una línea	//.*\$	// Esto es un comentario
Comentario multilínea	/*[\s\S]*?*/	/* Este es un comentario multilínea */
Espacio en blanco	[\t\n\r]+	(espacios, tabulaciones, saltos de línea)
Coma	,	,

Img. 1.10 - Ejemplo de una tabla de tokens

2. **Salidas:** Un **árbol de sintaxis** o **árbol de derivación**, que es una estructura de datos que muestra cómo se deben agrupar los tokens de acuerdo con las reglas de la gramática.



Img. 1.11 - Ejemplo de árbol sintáctico

3. **Objetivo:** Verificar si el texto de entrada es gramaticalmente correcto y construir su representación estructural.

Img. 1. 12 Capturas de pantalla y código

```
private void lexicalAnalysis() {
    // Extraer tokens
    Lexer lexer;
    try {
        File codigo = new File("code.encrypter");
        FileOutputStream output = new FileOutputStream(codigo);
        byte[] bytesText = jtpCode.getText().getBytes();
        output.write(bytesText);
        BufferedReader entrada = new BufferedReader(new InputStreamReader(new FileInputStream(codigo), "UTF8"));
        lexer = new Lexer(entrada);
        while (true) {
            Token token = lexer.yylex();
            if (token == null) {
                break;
            }
            tokens.add(token);
        }
    } catch (FileNotFoundException ex) {
        System.out.println("El archivo no pudo ser encontrado... " + ex.getMessage());
    } catch (IOException ex) {
        System.out.println("Error al escribir en el archivo... " + ex.getMessage());
    }
}
```

podemos observar una parte del código (usando java), de nuestro programa

```
private void compile() {
    btnTripletas.setEnabled(true);
    jButton3.setEnabled(true);
    jButton4.setEnabled(true);
    clearFields();
    lexicalAnalysis();
    fillTableTokens();
    syntacticAnalysis();
    semanticAnalysis();

    printConsole();
    int resp = JOptionPane.showConfirmDialog(null, "?Desea optimizar el código?", "", JOptionPane.YES_NO_OPTION);
    if (resp == JOptionPane.YES_OPTION) {
        jButton4.setEnabled(false);
        optimizacion();
    } else {
        codigoIntermedio();

        //codigoIntermedio();
        codeHasBeenCompiled = true;
    }
}
```

```
public void archivot(String ruta, String Objeto) {  
    try {  
        String contenido = Objeto;  
        File file = new File(ruta);  
        // Si el archivo no existe es creado  
        if (!file.exists()) {  
            file.createNewFile();  
        }  
        FileWriter fw = new FileWriter(file);  
        BufferedWriter bw = new BufferedWriter(fw);  
        bw.write(contenido);  
        bw.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
} //archivot  
  
public void abrirarchivo(String archivo){  
    try {  
        File objetofile = new File (archivo);  
        Desktop.getDesktop().open(objetofile);  
    }catch (IOException ex) {  
        System.out.println(ex);  
    }  
} //fin_AbrirArchivo
```

```

private void btnCompilarActionPerformed(java.awt.event.ActionEvent evt) {
    directorio.New();
    clearFields();
}

private void btnCompilar1ActionPerformed(java.awt.event.ActionEvent evt) {
    codObjComp.clear();
    variables.clear();
    compile();
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    if (directorio.Save()) {
        clearFields();
    }
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    if (directorio.Open()) {
        colorAnalysis();
        clearFields();
    }
}

private void btnTripletasActionPerformed(java.awt.event.ActionEvent evt) {
    JTextArea textArea = new JTextArea(codigoIntermedio);
    JScrollPane scrollPane = new JScrollPane(textArea);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    textArea.setEditable(false);
    scrollPane.setPreferredSize( new Dimension( 400, 500 ) );
    JOptionPane.showMessageDialog(null, scrollPane, "Tripletas", JOptionPane.PLAIN_MESSAGE);
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    String obj = "-----CODIGO OBJETO-----\n";
    for(String obj1: codObjComp){ obj+=obj1; }
    JTextArea textArea = new JTextArea(obj);
    //generamos el codigo ensamblador
    ensamblador = ensamblador(obj.replaceAll("-----CODIGO OBJETO-----\n\n\n", ""));
    JScrollPane scrollPane = new JScrollPane(textArea);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    textArea.setEditable(false);
    scrollPane.setPreferredSize( new Dimension( 400, 500 ) );
    JOptionPane.showMessageDialog(null, scrollPane, "Codigo Objeto", JOptionPane.PLAIN_MESSAGE);
}

```

```

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    //Creamos la ventana emergente para presentar el código ensamblador
    JTextArea textArea = new JTextArea(ensamblador);
    JScrollPane scrollPane = new JScrollPane(textArea);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    textArea.setEditable(false);
    scrollPane.setPreferredSize( new Dimension( 400, 500 ) );
    JOptionPane.showMessageDialog(null, scrollPane, "Tripletas", JOptionPane.PLAIN_MESSAGE);
    //Creamos el archivo y lo ejecutamos
    archivoT(".\\COD_OBJ.asm",ensamblador);
    abrirarchivo(".\\COD_OBJ.asm");
}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    optimizacion();
    JTextArea textArea = new JTextArea(codigoOptimizado);
    JScrollPane scrollPane = new JScrollPane(textArea);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    textArea.setEditable(false);
    scrollPane.setPreferredSize( new Dimension( 400, 500 ) );
    JOptionPane.showMessageDialog(null, scrollPane, "Optimizacion", JOptionPane.PLAIN_MESSAGE);
    codeHasBeenCompiled = true;
}

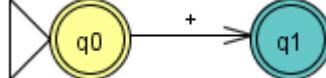
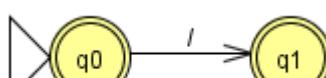
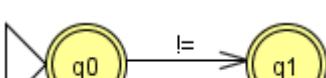
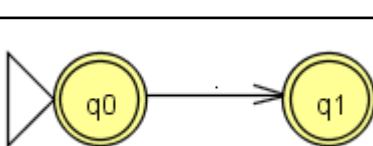
```

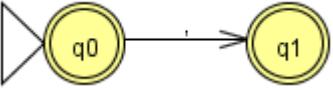
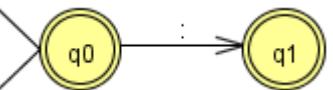
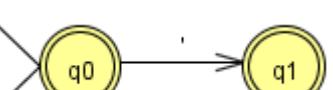
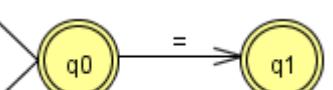
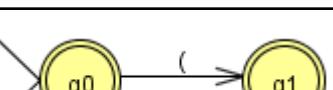
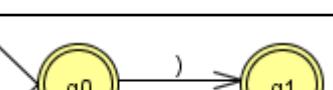
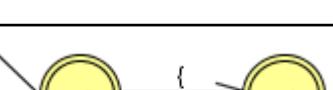
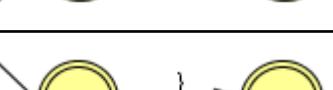
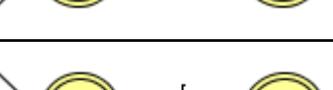
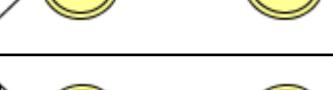
Generación de la tabla de símbolo y de direcciones

Las tablas de símbolos (también llamadas tablas de identificadores y tablas de nombres), realizan dos importantes funciones en el proceso de traducción: verificar que la semántica sea correcta y ayudar en la generación apropiada de código. Ambas funciones se realizan insertando o recuperando desde la tabla de símbolos los atributos de las variables usadas en el programa fuente.

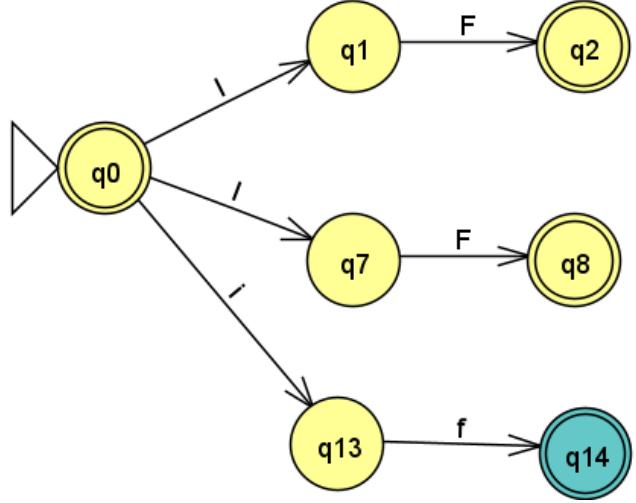
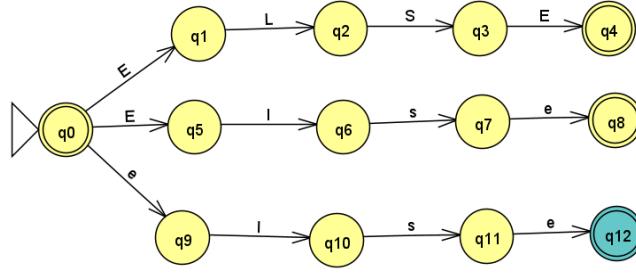
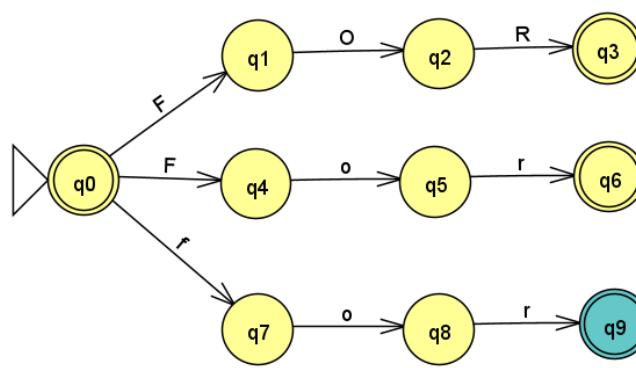
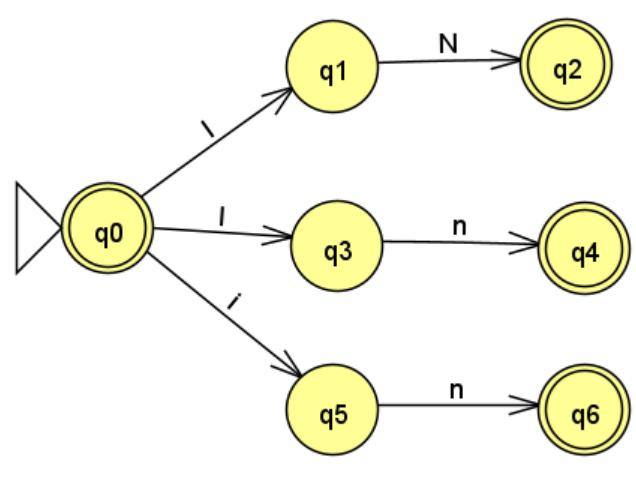
Estos atributos, tales como: el nombre, tipo, dirección de almacenamiento y dimensión de una variable, usualmente se encuentran explícitamente en las declaraciones o más implícitamente a través del contexto en que aparecen los nombres de variables en el programa.

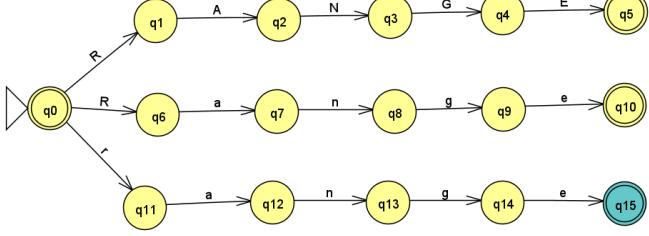
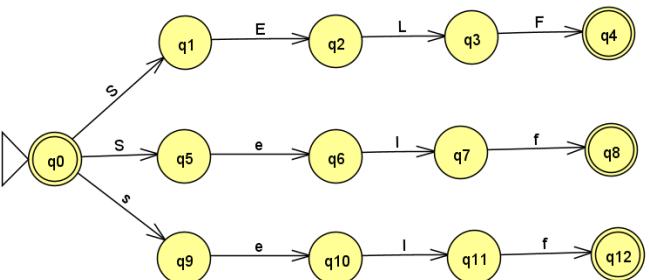
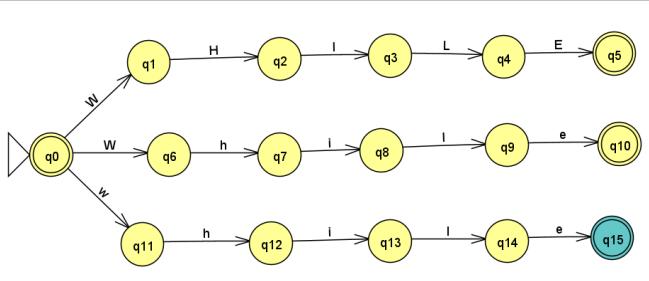
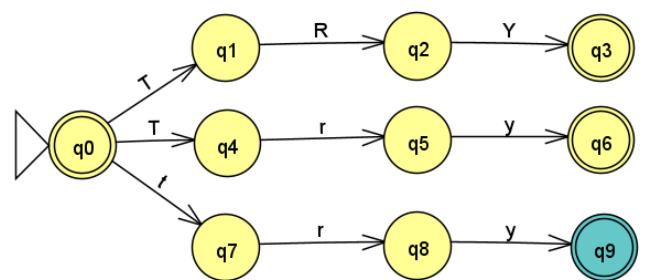
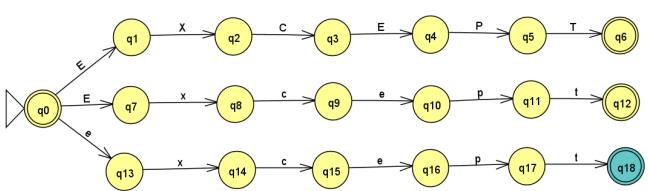
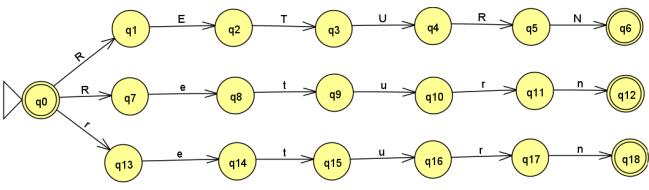
Árboles Sintácticos

#	Nombre del Token	Árbol Sintáctico
1	SUMA	
2	RESTA	
3	DIVISION	
4	MULTIPLICACION	
5	IGUAL	
6	DIFERENTE	
7	MAYORQUE	
8	MENORQUE	
9	MAYORIGUALQUE	
10	MENORIGUALQUE	
11	PUNTO	

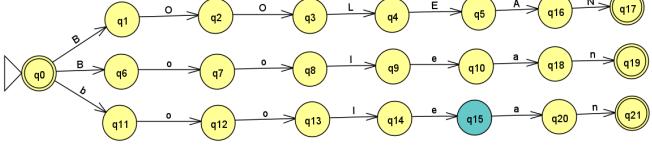
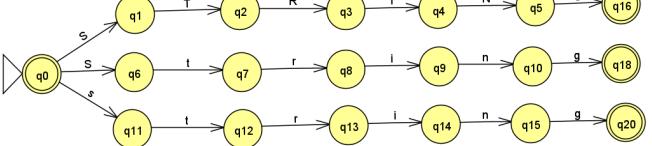
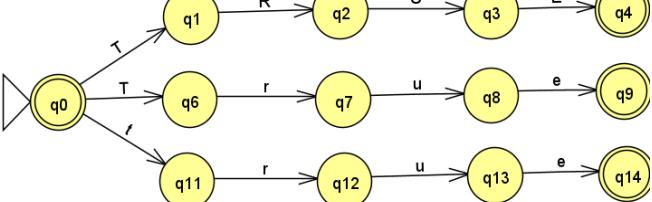
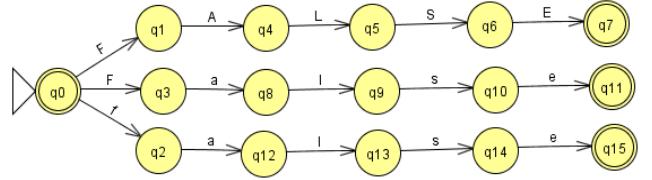
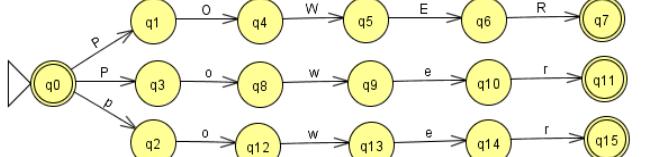
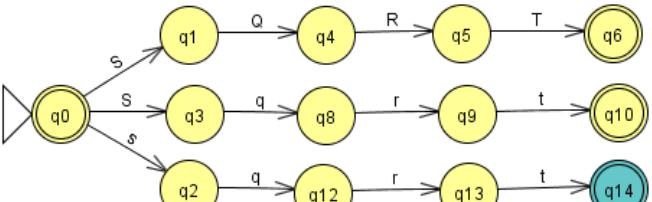
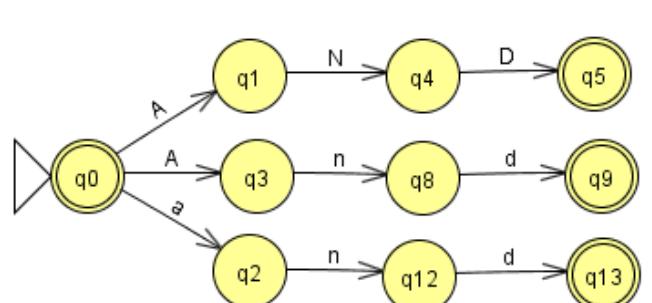
12	COMA	
13	DOSPUNTOS	
14	PUNTOCOMA	
15	COMILLASSIMPLE	
16	CADENA	
17	ASIGNACION	
18	COMILLADOBLE	
19	PARENTESISABIERTO	
20	PARENTESISCERRADO	
21	LLAVEABIERTO	
22	LLAVECERRADO	
23	CORCHETEABIERTO	
24	CORCHETECERRADO	

25	INCREMENTO	
26	DECREMENTO	
27	IMPORT	
28	DEF	
29	CLASS	

30	IF	 <pre> graph LR q0((q0)) -- l --> q1((q1)) q0 -- l --> q7((q7)) q0 -- f --> q13((q13)) q1 -- F --> q2((q2)) q7 -- F --> q8((q8)) q13 -- f --> q14(((q14))) </pre>
31	ELSE	 <pre> graph LR q0((q0)) -- E --> q1((q1)) q0 -- E --> q5((q5)) q0 -- e --> q9((q9)) q1 -- l --> q2((q2)) q2 -- s --> q3((q3)) q3 -- E --> q4((q4)) q5 -- l --> q6((q6)) q6 -- s --> q7((q7)) q7 -- e --> q8((q8)) q9 -- l --> q10((q10)) q10 -- s --> q11((q11)) q11 -- e --> q12(((q12))) </pre>
32	FOR	 <pre> graph LR q0((q0)) -- f --> q1((q1)) q0 -- F --> q4((q4)) q0 -- f --> q7((q7)) q1 -- o --> q2((q2)) q2 -- R --> q3((q3)) q3 -- R --> q4((q4)) q4 -- o --> q5((q5)) q5 -- r --> q6((q6)) q6 -- r --> q7((q7)) q7 -- o --> q8((q8)) q8 -- r --> q9(((q9))) </pre>
33	IN	 <pre> graph LR q0((q0)) -- l --> q1((q1)) q0 -- l --> q3((q3)) q0 -- l --> q5((q5)) q1 -- N --> q2((q2)) q3 -- n --> q4((q4)) q5 -- n --> q6((q6)) </pre>

34	RANGE	 A Non-deterministic Finite Automaton (NFA) with 16 states (q0 to q15). The start state is q0. Transitions from q0 include R to q1, R to q6, and r to q11. From q1, A leads to q2. From q2, N leads to q3. From q3, G leads to q4. From q4, E leads to q5. From q6, a leads to q7. From q7, n leads to q8. From q8, g leads to q9. From q9, e leads to q10. From q11, a leads to q12. From q12, n leads to q13. From q13, g leads to q14. From q14, e leads to q15.
35	SELF	 A Non-deterministic Finite Automaton (NFA) with 12 states (q0 to q11). The start state is q0. Transitions from q0 include S to q1, S to q5, and s to q9. From q1, E leads to q2. From q2, L leads to q3. From q3, F leads to q4. From q5, e leads to q6. From q6, l leads to q7. From q7, f leads to q8. From q9, e leads to q10. From q10, l leads to q11. From q11, f leads to q12.
36	WHILE	 A Non-deterministic Finite Automaton (NFA) with 16 states (q0 to q15). The start state is q0. Transitions from q0 include W to q1, W to q6, and w to q11. From q1, H leads to q2. From q2, I leads to q3. From q3, L leads to q4. From q4, E leads to q5. From q6, h leads to q7. From q7, i leads to q8. From q8, l leads to q9. From q9, e leads to q10. From q11, h leads to q12. From q12, i leads to q13. From q13, l leads to q14. From q14, e leads to q15.
37	TRY	 A Non-deterministic Finite Automaton (NFA) with 9 states (q0 to q8). The start state is q0. Transitions from q0 include T to q1, T to q4, and t to q7. From q1, R leads to q2. From q2, Y leads to q3. From q3, Y leads to q5. From q4, r leads to q5. From q5, y leads to q6. From q7, r leads to q8. From q8, y leads to q9.
38	EXCEPT	 A Non-deterministic Finite Automaton (NFA) with 18 states (q0 to q17). The start state is q0. Transitions from q0 include E to q1, E to q7, and e to q13. From q1, x leads to q2. From q2, c leads to q3. From q3, E leads to q4. From q4, P leads to q5. From q5, T leads to q6. From q7, x leads to q8. From q8, c leads to q9. From q9, e leads to q10. From q10, P leads to q11. From q11, t leads to q12. From q13, x leads to q14. From q14, c leads to q15. From q15, e leads to q16. From q16, P leads to q17. From q17, t leads to q18.
39	RETURN	 A Non-deterministic Finite Automaton (NFA) with 19 states (q0 to q18). The start state is q0. Transitions from q0 include R to q1, R to q7, and r to q13. From q1, E leads to q2. From q2, T leads to q3. From q3, U leads to q4. From q4, R leads to q5. From q5, N leads to q6. From q7, e leads to q8. From q8, t leads to q9. From q9, u leads to q10. From q10, r leads to q11. From q11, n leads to q12. From q13, e leads to q14. From q14, t leads to q15. From q15, u leads to q16. From q16, r leads to q17. From q17, n leads to q18.

40	BREAK	<pre> graph LR q0((q0)) -- R --> q1((q1)) q0 -- B --> q6((q6)) q0 -- δ --> q11((q11)) q1 -- E --> q3((q3)) q6 -- r --> q7((q7)) q11 -- r --> q12((q12)) q3 -- A --> q4((q4)) q7 -- e --> q8((q8)) q12 -- e --> q13((q13)) q4 -- K --> q5((q5)) q8 -- a --> q9((q9)) q13 -- a --> q14((q14)) q9 -- k --> q10((q10)) q14 -- k --> q15((q15)) </pre>
41	NEXT	<pre> graph LR q0((q0)) -- E --> q1((q1)) q0 -- N --> q5((q5)) q0 -- n --> q9((q9)) q1 -- X --> q3((q3)) q5 -- x --> q6((q6)) q9 -- x --> q10((q10)) q3 -- T --> q4((q4)) q6 -- t --> q7((q7)) q10 -- t --> q11((q11)) q7 -- t --> q12((q12)) </pre>
42	INPUT	<pre> graph LR q0((q0)) -- N --> q1((q1)) q0 -- n --> q6((q6)) q0 -- n --> q11((q11)) q1 -- P --> q2((q2)) q6 -- p --> q7((q7)) q11 -- p --> q12((q12)) q2 -- U --> q3((q3)) q7 -- p --> q8((q8)) q12 -- p --> q13((q13)) q3 -- T --> q4((q4)) q8 -- u --> q9((q9)) q13 -- u --> q14((q14)) q4 -- T --> q5((q5)) q9 -- t --> q10((q10)) q14 -- t --> q15((q15)) </pre>
43	OUTPUT	<pre> graph LR q0((q0)) -- o --> q1((q1)) q0 -- o --> q6((q6)) q0 -- u --> q11((q11)) q1 -- T --> q2((q2)) q6 -- p --> q7((q7)) q11 -- p --> q12((q12)) q2 -- P --> q3((q3)) q7 -- p --> q8((q8)) q12 -- p --> q13((q13)) q3 -- U --> q4((q4)) q8 -- p --> q9((q9)) q13 -- p --> q14((q14)) q4 -- T --> q5((q5)) q9 -- t --> q10((q10)) q14 -- t --> q15((q15)) q5 -- T --> q16((q16)) q10 -- t --> q17((q17)) q15 -- t --> q18((q18)) </pre>
44	PRINT	<pre> graph LR q0((q0)) -- P --> q1((q1)) q0 -- P --> q6((q6)) q0 -- r --> q11((q11)) q1 -- I --> q2((q2)) q6 -- r --> q7((q7)) q11 -- r --> q12((q12)) q2 -- N --> q3((q3)) q7 -- r --> q8((q8)) q12 -- r --> q13((q13)) q3 -- T --> q4((q4)) q8 -- n --> q9((q9)) q13 -- n --> q14((q14)) q4 -- T --> q5((q5)) q9 -- t --> q10((q10)) q14 -- t --> q15((q15)) </pre>
45	INT	<pre> graph LR q0((q0)) -- N --> q1((q1)) q0 -- I --> q6((q6)) q0 -- n --> q11((q11)) q1 -- T --> q2((q2)) q6 -- t --> q7((q7)) q11 -- t --> q12((q12)) q2 -- T --> q3((q3)) q7 -- t --> q8((q8)) q12 -- t --> q13((q13)) </pre>
46	FLOAT	<pre> graph LR q0((q0)) -- F --> q1((q1)) q0 -- r --> q6((q6)) q0 -- l --> q11((q11)) q1 -- O --> q2((q2)) q6 -- o --> q7((q7)) q11 -- o --> q12((q12)) q2 -- A --> q3((q3)) q7 -- o --> q8((q8)) q12 -- o --> q13((q13)) q3 -- T --> q4((q4)) q8 -- a --> q9((q9)) q13 -- a --> q14((q14)) q4 -- T --> q5((q5)) q9 -- t --> q10((q10)) q14 -- t --> q15((q15)) </pre>

47	BOOLEAN	
48	STRING	
49	TRUE	
50	FALSE	
51	POWER	
52	SQRT	
53	AND	

54	OR	<pre> graph LR q0((q0)) -- O --> q1((q1)) q0 -- O --> q3((q3)) q0 -- O --> q2((q2)) q1 -- R --> q4((q4)) q3 -- r --> q8((q8)) q2 -- r --> q12((q12)) </pre>
55	NOT	<pre> graph LR q0((q0)) -- N --> q1((q1)) q0 -- N --> q3((q3)) q0 -- N --> q2((q2)) q1 -- T --> q4((q4)) q3 -- t --> q8((q8)) q2 -- t --> q12((q12)) </pre>
56	BEGIN	<pre> graph LR q0((q0)) -- B --> q1((q1)) q0 -- B --> q3((q3)) q0 -- b --> q2((q2)) q1 -- G --> q4((q4)) q4 -- G --> q5((q5)) q5 -- I --> q6((q6)) q6 -- N --> q7((q7)) q3 -- e --> q8((q8)) q8 -- g --> q9((q9)) q9 -- i --> q10((q10)) q10 -- n --> q11((q11)) q2 -- e --> q12((q12)) q12 -- g --> q13((q13)) q13 -- i --> q14((q14)) q14 -- n --> q15((q15)) </pre>
57	END	<pre> graph LR q0((q0)) -- E --> q1((q1)) q0 -- E --> q3((q3)) q0 -- e --> q2((q2)) q1 -- N --> q4((q4)) q4 -- D --> q5((q5)) q3 -- n --> q8((q8)) q8 -- d --> q9((q9)) q2 -- n --> q12((q12)) q12 -- d --> q13((q13)) </pre>

Conclusiones

En el proyecto final que se presenta integra los conocimientos adquiridos durante el semestre, para la creación de un proyecto de Compilador de código para sistemas inteligentes enfocados en desarrollo de lenguaje ensamblador.

Este programa ejecuta las etapas del procesamiento de código fuente, al ingresar código en su interfaz gráfica desarrollada en el IDE Apache Netbeans bajo el lenguaje “Java”, el programa realiza un análisis léxico y tokenización. En esta fase, identifica y clasifica secuencias de caracteres significativos (identificadores, palabras clave, números, símbolos, etc.) mediante patrones predefinidos y priorizados, generando tokens con información de tipo, valor y ubicación en el código, mostrándonos mediante una tabla.

A continuación, se efectúa un análisis estructural básico (análisis sintáctico), verificando el balanceo de símbolos (paréntesis, llaves, corchetes) y la disposición esperada de los tokens en construcciones comunes (como condiciones en estructuras if o while), alertando sobre posibles omisiones o faltas de componentes léxicos o tokens.

Luego, se lleva a cabo un análisis semántico utilizando una tabla de símbolos. Esta tabla registra las declaraciones (variables, etc.) con su nombre, tipo y ámbito. El análisis verifica la existencia y el uso coherente de estos elementos (compatibilidad de tipos, inicialización) y detecta errores como declaraciones o elementos no utilizados. El manejo de ámbitos permite distinguir elementos con el mismo nombre en diferentes partes del código.

Finalmente, la aplicación permite un guardado de archivos mediante archivos locales de cada compilación en un formato propio con terminación “.comp” que incluye todo el código que se tenga sobre el canva de escritura. En resumen, el programa ofrece una solución práctica y sencilla de los conceptos fundamentales de las fases de la compilación.

Recomendaciones

1. Análisis Léxico

- Robustecimiento del Manejo de Errores Léxicos:
 - Implementar un mecanismo de recuperación de errores que, ante la detección de un lexema inválido, consuma la mayor cantidad posible de caracteres erróneos consecutivos. Esto permitirá la emisión de diagnósticos más informativos, tales como "Lexema mal formado" o "Identificador inválido", en lugar de simplemente señalar el primer carácter problemático.
- Ampliación del Soporte de Literales:
 - Extender la capacidad del compilador para reconocer y procesar una gama más amplia de literales, incluyendo valores booleanos, el valor nulo y potencialmente otros tipos de literales específicos
- Flexibilización de la Configuración de Patrones:
 - Considerar la posibilidad de externalizar la definición de los patrones léxicos, quizás mediante un archivo de configuración. Esta medida aumentaría la adaptabilidad del analizador a diferentes lenguajes de programación y no únicamente a lenguaje ensamblador.

2. Análisis Sintáctico

- Construcción del Árbol de Sintaxis Abstracta (AST):
 - La generación de un AST es un paso fundamental para facilitar los análisis semánticos subsiguientes y la eventual generación de código.
- Mejora de la Recuperación de Errores Sintácticos:
 - Perfeccionar los mecanismos de recuperación de errores sintácticos para que el compilador, en lugar de detenerse ante el primer error, intente continuar el análisis e identificar errores adicionales. Se debe prestar especial atención a evitar la entrada en bucles infinitos.
- Extensión del Soporte Gramatical:
 - Ampliar la cobertura del compilador para abarcar construcciones sintácticas más complejas del lenguaje, como declaraciones de funciones, definiciones de clases y otras estructuras gramaticales avanzadas.

3. Análisis Semántico

- Desarrollo de un Sistema de Tipos Robusto:
 - Implementar un sistema de tipos más completo que incluya la verificación de tipos en expresiones, llamadas a funciones, y conversiones de tipo (tanto implícitas como explícitas).
- Verificación Exhaustiva de Funciones:
 - Asegurar la correcta declaración y uso de funciones, validando el número y tipo de argumentos, así como el tipo de retorno.
- Manejo de Clases y Objetos:
 - En el caso de lenguajes orientados a objetos, implementar la verificación de clases, relaciones de herencia, acceso a miembros y otras características relevantes.
- Análisis de Flujo Básico:
 - Incorporar un análisis de flujo básico para detectar potenciales errores, como el uso de variables no inicializadas o la presencia de código inalcanzable.
- Optimización de la Tabla de Símbolos:
 - Refinar la estructura y la funcionalidad de la tabla de símbolos para almacenar una mayor cantidad de información (atributos de variables, tipos de funciones, etc.) y optimizar las operaciones de búsqueda.

4. Interfaz Gráfica - (Java Swing Controls)

- Perfeccionamiento de la Visualización de Errores:
 - Emplear recursos visuales, como colores, iconos o paneles diferenciados, para presentar de manera clara y efectiva los errores y advertencias generados en cada fase del análisis.
- Implementación del Resaltado de Sintaxis:
 - Integrar el resultado de sintaxis en el área de edición de código para mejorar la legibilidad y facilitar la identificación de elementos del lenguaje.
- Navegación Intuitiva de Errores:
 - Proporcionar la funcionalidad de navegar directamente a la línea de código correspondiente al hacer clic en un mensaje de error en la salida del análisis.

- Opciones de Configuración:
 - Ofrecer a los usuarios la posibilidad de personalizar la apariencia de la interfaz, incluyendo la configuración del tamaño de la fuente, la paleta de colores y otros aspectos visuales.
- Funcionalidades de Edición de Código:
 - Incorporar funciones básicas de edición de código, como cortar, copiar, pegar, deshacer yrehacer.

5. Aspectos Generales

- Documentación Exhaustiva
 - Documentar el código de manera clara y completa para facilitar su comprensión para poder considerarla como una herramienta de uso libre para la comunidad que le interese tener algún compilador hecho a medida.
- Optimización del Rendimiento
 - Mejorar el rendimiento del compilador, especialmente en el procesamiento de códigos que tengan un gran contenido de líneas o que requieran de múltiples hilos para procesamiento matemático.
- Diseño de Extensibilidad
 - Desarrollar el compilador con la previsión de futuras extensiones, facilitando la adición de nuevas funcionalidades o el soporte para diferentes lenguajes de programación.
- Mejoras en el manejo de archivos
 - Ampliar las opciones para la exportación de los resultados del compilador en diversos formatos.
- Historial de compilaciones integrado
 - Implementar un historial de compilaciones con funcionalidades avanzadas de búsqueda y filtrado.

6. Extensiones Avanzadas

- Integración con entornos de desarrollo integrados (IDE's)
 - Explorar la viabilidad de integrar el analizador como un plugin en IDE's populares, lo que proporciona una experiencia de desarrollo más fluida.

Bibliografía

<https://www.reflection.uniovi.es/ortin/publications/semanitico.pdf>

<https://www.questionpro.com/blog/es/analisis-semanitico/>

<http://itpn.mx/recursosisc/7semestre/leguajesyautomatas2/Unidad%20I.pdf>