

INSTITUTO TECNOLÓGICO DE CELAYA

Lenguajes y Autómatas II

Equipo 2

Gomez Cabañas Anthony 20030057



Ortega Hernández Cristhian Alberto 20031091

Ponce Morales Alma Gabriela 20030274

Ruiz López Miguel Ángel 20030935

DOCUMENTACIÓN PROYECTO FINAL COMPILADOR

I.S.C. Ricardo González González

 TECNOLÓGICO NACIONAL DE MÉXICO	INSTITUTO TECNOLÓGICO DE CELAYA	
PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II	AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López	



[Handwritten signatures and initials]

CARRERA	NOMBRE DE LA ASIGNATURA
INGENIERIA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTÓMATAS II

PROYECTO NO.	NOMBRE DEL PROYECTO	FECHA DE ENTREGA
1	Desarrollo de un compilador	30/11/2023

VIDEO EXPLICATIVO: [LINK](#)
 REPOSITORIO CÓDIGO: [LINK](#)

1	INTRODUCCION
	<p>En el siguiente proyecto, el equipo desarrollará un lenguaje de programación propio a fin de implementarlo en un compilador, el cual se realizarán las tres fases de análisis básicas: análisis léxico, análisis sintáctico y análisis semántico.</p> <p>En el lenguaje de programación se incluirán elementos básicos de cualquier lenguaje, como estructuras de control, estructuras de decisión, palabras reservadas, operadores aritméticos y lógicos, identificadores, entre otras cosas.</p> <p>Para los diferentes análisis no se utilizarán herramientas de desarrollo de terceros como FLEX o YACC, sino que todas las herramientas utilizadas serán de desarrollo propio del equipo, a fin de hacer un compilador de autoría propia y original, pues las herramientas únicamente se revisarán como base teórica.</p> <p>Para el análisis léxico, se verificará que las cadenas introducidas sean correctas, al analizar que se respeten palabras reservadas, estructuras de control o símbolos necesarios para crear operaciones matemáticas y lógicas.</p> <p>Para el análisis sintáctico, el equipo se basará en herramientas, conceptos y notaciones que corroboren que la cadena introducida posea y mantenga una estructura cohesiva y coherente, a fin de que el compilador pueda crear exitosamente un ejecutable del código introducido en el lenguaje propio.</p> <p>Además de las fases de análisis léxico, sintáctico y semántico, nuestro equipo también se enfocará en aspectos adicionales que son esenciales para el éxito de este proyecto.</p> <p>Este proyecto representa un emocionante desafío técnico y creativo para el equipo. Crear un lenguaje de programación único y un compilador de autoría propia significa para el equipo un reto de gran importancia, pues así se podrá visualizar cómo funcionan de mejor manera los compiladores que existen para los diferentes lenguajes de programación.</p> <p>Por último, se implementará una interfaz gráfica para que el usuario pueda visualizar un semáforo de los diferentes análisis, es decir, que cuando se ejecute bien un análisis, su semáforo se ponga en verde; también se incluirá una “consola” en donde se imprimirán los resultados de las operaciones o instrucciones dadas en el código o, en caso de errores, se imprimirán los errores de código. De</p>



 TECNOLÓGICO NACIONAL DE MÉXICO	INSTITUTO TECNOLÓGICO DE CELAYA		
PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II		AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López	

[Handwritten signatures and initials]

igual manera, se incluirá una tabla de símbolos con diferentes columnas, entre las cuales estarán los tipos de datos, id de los tokens, valores, nombres, etcétera.

2	OBJETIVO
	<p>El objetivo de este proyecto es desarrollar un compilador que ejecute secuencialmente las tres etapas de análisis iniciando por el análisis léxico, seguido del análisis sintáctico para finalmente hacer el análisis semántico. Este compilador no involucrará las etapas posteriores de los análisis de un compilador como la etapa de generación de código intermedio o la optimización, únicamente constará de las etapas de análisis.</p> <p>Cada etapa de análisis tiene su objetivo específico, para el análisis léxico, su objetivo será crear tokens mediante el reconocimiento de palabras claves, operadores e identificadores; el análisis sintáctico verificará si las secuencias de tokens creados por el análisis léxico forman expresiones y declaraciones válidas según la gramática del lenguaje; finalmente, el análisis semántico realizará verificaciones semánticas para asegurar que las operaciones y estructuras del programa tengan sentido y cumplan con las reglas del lenguaje de programación. Esto incluye la validación de tipos, la resolución de identificadores, entre otras reglas.</p>



3	FUNDAMENTO
	<p>La historia de los compiladores se remonta a la década de 1950, cuando se introdujo el término "compilador" y se realizaron los primeros trabajos en relación con la traducción de fórmulas aritméticas a código de máquina. Desde entonces, los compiladores han evolucionado significativamente, integrando técnicas de análisis léxico, sintáctico y semántico para procesar código de alto nivel en código de máquina.</p> <p>El objetivo central de este proyecto es desarrollar un compilador que se centre exclusivamente en las etapas de análisis: léxico, sintáctico y semántico. Cada una de estas etapas tiene un objetivo específico como ya se mencionó en el punto anterior sobre los objetivos.</p> <p>Este proyecto se justifica por la necesidad de entender y manipular mejor los lenguajes de programación. Un compilador que se centra en las etapas de análisis proporciona una visión detallada de cómo un lenguaje de programación interpreta y procesa el código fuente. Además, este proyecto también puede servir como una base para futuras ampliaciones, como la incorporación de etapas posteriores de compilación, como la generación de código intermedio y la optimización.</p> <p>Este proyecto es relevante en el campo de la informática y la programación, ya que proporciona una comprensión profunda de cómo funcionan los lenguajes de programación. Además, el conocimiento adquirido a través de este proyecto puede ser aplicado en la creación de nuevos lenguajes de programación o en la mejora de los existentes.</p> <p>Por último, el alcance de este proyecto se limita a las etapas de análisis en un compilador, específicamente el análisis léxico, sintáctico y semántico. No se abordarán las etapas posteriores de un compilador, como la generación de código intermedio y la optimización. Aunque el proyecto se</p>

 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures and initials]

centra en estas etapas de análisis, puede servir como una base para futuras expansiones o investigaciones en el campo de la compilación.

4	MARCO TEÓRICO
<p>Un compilador es un programa en informática esencial en el desarrollo de software que realiza una tarea fundamental: traducir el código fuente de un programa, escrito por un programador en un lenguaje de programación específico, a un formato que pueda ser comprendido y ejecutado por la máquina o el dispositivo de destino. El código fuente, escrito en un lenguaje de alto nivel como C, C++, Java o Python, es comprensible para los programadores, pero no para la máquina.</p> <p>Cuando un programador completa la escritura del código fuente, el compilador entra en juego. El compilador analiza el código fuente y realiza una serie de transformaciones que lo traducen a un lenguaje de bajo nivel, denominado código objeto o código máquina. Este código objeto es específico para la arquitectura de la máquina en la que se ejecutará el programa. En otras palabras, el compilador actúa como un traductor que convierte las instrucciones escritas en un lenguaje de alto nivel en instrucciones comprensibles por el hardware subyacente.</p> <p>La ventaja de este enfoque radica en la portabilidad y eficiencia. El mismo código fuente puede compilarse para ejecutarse en diferentes plataformas, siempre que haya un compilador disponible para cada una. Además, la traducción a código máquina permite una ejecución más rápida y eficiente del programa, ya que las instrucciones son directamente ejecutables por la máquina sin necesidad de una interpretación adicional.</p> <p>El funcionamiento de un compilador se puede dividir en varias fases o etapas, cada una de las cuales realiza una tarea en específico, entre las cuales destacan las etapas de análisis, que en conjunto verifican que el código sea correcto para continuar con su procesamiento. Estas etapas de análisis son las siguientes:</p> <ol style="list-style-type: none"> 1. Análisis Léxico: La fase inicial del proceso de compilación es el análisis léxico. En esta etapa, el compilador examina el código fuente y lo descompone en una secuencia de tokens, que son las unidades fundamentales del lenguaje de programación. Posteriormente, estos tokens se transmiten a la siguiente fase para su procesamiento adicional. 2. Análisis Sintáctico: La segunda etapa del proceso de compilación involucra el análisis sintáctico. Aquí, el compilador toma la secuencia de tokens generada por el análisis léxico y verifica si cumple con la gramática del lenguaje de programación. Como resultado de esta fase, se obtiene un Árbol de Sintaxis Abstracta (AST), que representa la estructura lógica del programa. 3. Análisis Semántico: La tercera fase en el proceso de compilación es el análisis semántico. En esta etapa, el compilador evalúa la corrección semántica del código, verificando si cumple con el sistema de tipos del lenguaje y otras reglas semánticas. Durante esta etapa, se examina el significado del 	

 TECNOLÓGICO NACIONAL DE MÉXICO	INSTITUTO TECNOLÓGICO DE CELAYA		
PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II		AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López	



[Handwritten signatures and initials]

código fuente para asegurarse de que sea coherente. La comprobación de tipos se realiza para garantizar un uso adecuado de las variables y la realización de operaciones en tipos de datos compatibles.

Después de las fases de análisis, se siguen las siguientes etapas en el proceso de compilación, entre ellas está la Generación de Código Intermedio, que constituye la cuarta fase de un compilador. En esta etapa, se crea una representación intermedia del código fuente que puede ser traducida fácilmente a código máquina. La Optimización, fase cinco, aplica diversas técnicas para mejorar el rendimiento del código intermedio, buscando eficiencias y mejoras. Finalmente, la Generación de Código concluye el proceso. En esta sexta fase, se toma el código intermedio optimizado y se genera el código máquina real que puede ejecutarse en el hardware objetivo. A pesar de estas fases adicionales que pueden incrementar la eficiencia y rendimiento, nuestro enfoque se limitará a las fases de análisis para cumplir con los objetivos específicos de nuestro proyecto.

5	MATERIALES NECESARIOS
	<ul style="list-style-type: none"> • Lenguaje de programación: Java • Librerías: Swing de Java • Editores de Código: NetBeans IDE • Editor de Diagramas: easyUML para Java • Repositorios: Git y GitHub

6	DESARROLLO
	1. ALFABETO UTILIZADO El alfabeto utilizado para el compilador está constituido principalmente por los caracteres del alfabeto latino básico (no incluye 'ñ'), mayúsculas y minúsculas, además de números y de otros signos: LETRAS [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] NÚMEROS [0 1 2 3 4 5 6 7 8 9] SIGNOS DE PUNTUACIÓN [; "] SIGNOS DE AGRUPACIÓN [() [] { } @] OPERADORES ARITMÉTICOS [+ - * /] OPERADORES LÓGICOS [&] OPERADORES RELACIONALES [< > = !]

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>		
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>		<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	



[Handwritten signatures and initials]

1.1 TABLA DE SIGNIFICADOS DE SIGNOS Y OPERADORES

SIGNOS		
	SÍMBOLO	SIGNIFICADO
Pun	;	Indica fin de línea
	“	Delimita cadenas de texto
Agrup	()	Agrupar expresiones y controlan jerarquía de operadores
	{ }	Delimitan bloques de código
	@@	Comentarios de una sola línea
OPERADORES		
	SÍMBOLO	SIGNIFICADO
Aritméticos	+	Suma
	-	Resta
	*	Multiplicación
	/	División
Lóg	&	AND
		OR
Relacionales	<	Menor que
	>	Mayor que
	=	Asigna valor a una variable
	==	Compara dos valores
	>=	Mayor o igual que
	<=	Menor o igual que
	!=	Distinto de

1.2 TABLA DE PALABRAS RESERVADAS

PALABRAS RESERVADAS	
TIPOS DE DATOS	
NOMBRE	DESCRIPCIÓN
int	Número entero
booleano	Valor verdadero o falso
text	Cadena de texto
ESTRUCTURAS DE DECISIÓN O CÍCLICAS	
if	Equivalente a 'if'
aslong	Equivalente a 'while'

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures and initials]

OTRAS	
show	Imprime en pantalla valores
principal	Denota al método principal

2. GRAMÁTICAS

A continuación, se mostrarán algunas gramáticas de algunas estructuras, tipos de datos, etcétera:

GRAMÁTICA DE 'int'

$$G_i = \{ (R, U), (0-9), p, s \}$$
$$p \{$$
$$\quad \langle R \rangle \rightarrow \langle U \rangle$$
$$\quad \langle U \rangle \rightarrow \langle U \rangle | \langle [0-9] \rangle$$
$$\}$$

GRAMÁTICA DE 'text'

$$G_t = \{ (L, B, F), (\Sigma), p, s \}$$
$$p \{$$
$$\quad \langle L \rangle \rightarrow \langle _ \rangle \langle S \rangle$$
$$\quad \langle B \rangle \rightarrow \langle [a-z] | [A-Z] | [0-9] | \Sigma \rangle (\langle S \rangle | \langle F \rangle)$$
$$\quad \langle F \rangle \rightarrow \langle _ \rangle$$
$$\}$$

GRAMÁTICA DE 'booleano'



$$G_b = \{ (L), (true, false), p, s \}$$
$$p \{$$
$$\quad \langle L \rangle \rightarrow \langle true \rangle | \langle false \rangle$$
$$\}$$

DECLARACIÓN DE UN 'int'

$$G_{vi} = \{ (B, N, A, U, F), (a-z, A-Z, 0-9), p, s \}$$
$$p \{$$
$$\quad \langle B \rangle \rightarrow \langle N \rangle$$
$$\quad \langle N \rangle \rightarrow \langle [a-z] | [A-Z] \rangle (\langle N \rangle | \langle A \rangle)$$
$$\quad \langle A \rangle \rightarrow \langle = \rangle \langle U \rangle$$
$$\quad \langle U \rangle \rightarrow \langle [0-9] \rangle (\langle U \rangle | \langle F \rangle)$$
$$\quad \langle F \rangle \rightarrow \langle ; \rangle$$
$$\}$$

DECLARACIÓN DE UN 'text'

$$G_{vt} = \{ (B, L, A, T, F), (a-z, A-Z, 0-9, \Sigma), p, s \}$$
$$p \{$$
$$\quad \langle B \rangle \rightarrow \langle L \rangle$$
$$\quad \langle L \rangle \rightarrow \langle [a-z] | [A-Z] \rangle (\langle N \rangle | \langle A \rangle)$$
$$\quad \langle A \rangle \rightarrow \langle = \rangle \langle _ \rangle \langle T \rangle$$
$$\quad \langle T \rangle \rightarrow \langle [a-z] | [A-Z] | [0-9] | \Sigma \rangle (\langle U \rangle | \langle F \rangle)$$
$$\quad \langle F \rangle \rightarrow \langle _ \rangle \langle ; \rangle$$
$$\}$$

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures]

<pre> } GRAMÁTICA DE 'aslong' Gca = { (B, N, L, C, E, F), p, s } p { → <aslong> <(<L> <)> <{> <N> <N> → <INSTRUCCIONES> <}> <L> → <E> <F> <E> → <variable> <C> <C> → <op. relacional> <F> <F> → <variable> <T> <T> → <op. lógico> (<F> <E>) } </pre> <p>Cabe aclarar que las gramáticas representadas anteriormente no son todas las existentes y, en algunos casos, son muy generalizadas, es decir, no son totalmente específicas con los elementos que pueden o deben llevar en cierta parte de la estructura real; esto debido a que se buscó representar a grandes rasgos su funcionamiento, pero en el código las estructuras sí están mejor definidas y no generalizadas.</p> <p>3. CASOS DE USO</p> <p>Declaración de variables 'int'</p> <pre> int a = 4 ; int b = 17 ; </pre> <p>Declaración de variables 'booleano'</p> <pre> booleano lt = true ; booleano lf = false ; </pre> <p>Declaración de variables 'text'</p> <pre> text d = "Hola, mundo" ; </pre> <p>Imprimir datos en la consola del compilador</p> <pre> show "a" ; </pre> <p>Esto imprime en la consola la letra 'a'</p> <pre> show "a: "+a ; </pre> <p>Suponiendo que 'a' vale 4, esto imprime en la consola: a: 4</p> <p>Actualización de valor de variables ya declaradas</p> <pre> a = b * (14 + 8) - 11 ; </pre> <p>Aquí, el valor de 'a' se vuelve a calcular, pero su tipo no cambia</p> <p>Declaración de una estructura de decisión 'if'</p> <pre> if (a<b b>=a){ show "Condicion 1" ; } if (lt & lf){ show "Condicion 2" ; } if (a >= b b > a){ show "Condicion 3" ; } </pre>



Alma Gabriela Ponce Morales
Anthony Gómez Cabañas
Cristhian Alberto Ortega Hernández
Miguel Ángel Ruiz López

```
}
if (lt | lf & a<b){
    show "Condicion 4" ;
}
```

Esto quiere decir que, mientras que 'a' sea menor que 'b', imprimirá el valor de 'a' y lo volverá a calcular sumándole 1.

```
principal {
    @@Escribe código aquí
}
```

4. DIAGRAMA DE CLASES

[illegible]

Además, como se puede observar, la clase 'lexico_tokens.java' hace una instancia/objeto de los tipos de datos que se pueden recibir para autenticarlos. Esta clase también tiene una conexión con 'lexico_lexema' pues, al hacerle una instancia, puede hacer uso de su método 'getLexemas'. Posteriormente, el analizador sintáctico (analizadorSintactico.java) hace una instancia de los tokens



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II

AUTORES:

Alma Gabriela Ponce Morales
Anthony Gómez Cabañas
Cristhian Alberto Ortega Hernández
Miguel Ángel Ruiz López

del analizador léxico mediante la variable 'token', del tipo 'lexico_token'. El analizador sintáctico también tiene instancias de la Pila de Errores y de la Tabla de Símbolos. Éstas no aparecen en el diagrama porque, aunque sus clases existen y son parte del código, su uso se limita a insertar, borrar y obtener sus datos por lo que, al hacer solo estas operaciones, tenerlas en el diagrama de clases causaría estorbos.

Realmente todas las clases ahí presentes, salvo por las del analizador léxico, hacen uso de la pila de errores y de la tabla de símbolos para estar ingresando, obteniendo o eliminando datos de estos componentes constantemente.

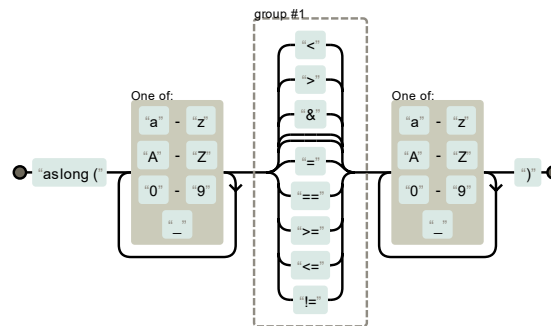
5. DIAGRAMAS DE LOS AUTÓMATAS

ESTRUCTURA DE ITERACIÓN "ASLONG"

Expresión Regular:

$aslong \setminus ([a-zA-Z0-9_]+ (<|>|&||=|==|>=|<=|!=) [a-zA-Z0-9_]+) \setminus$

Autómata:

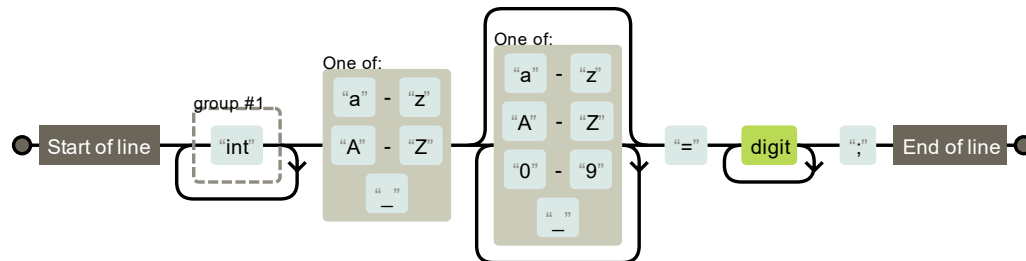


VARIABLE INT

Expresión Regular:

$^ (int) + [a-zA-Z_][a-zA-Z0-9_]* = \setminus d+ ; \$$

Autómata:



VARIABLE BOOLEANO

Expresión Regular:



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



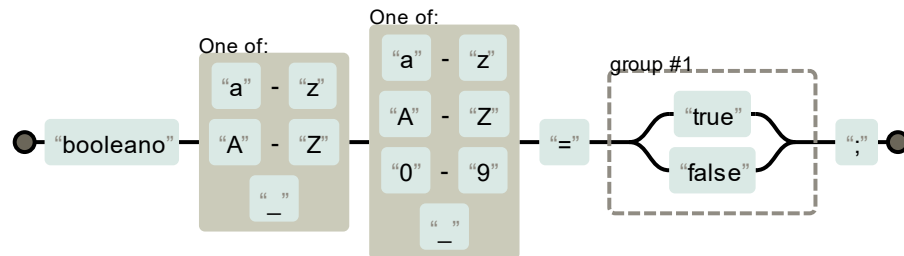
PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II

AUTORES:

Alma Gabriela Ponce Morales
Anthony Gómez Cabañas
Cristhian Alberto Ortega Hernández
Miguel Ángel Ruiz López

$^{\wedge}booleano \backslash s + [a - zA - Z_][a - zA - Z0 - 9_]* \backslash s * = \backslash s * (true|false) \backslash s *, \$$

Autómata:

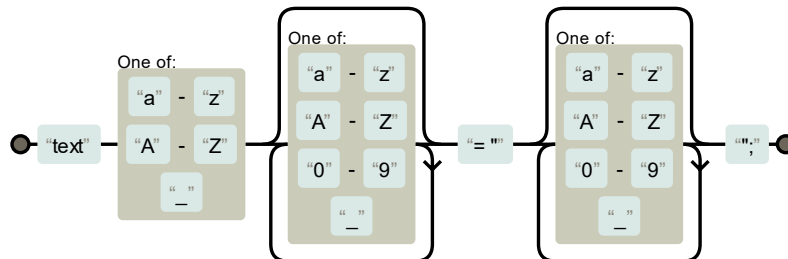


VARIABLE TEXT

Expresión Regular:

$text[a - zA - Z_][a - zA - Z0 - 9_]* = "[a - zA - Z0 - 9_]*$

Autómata:

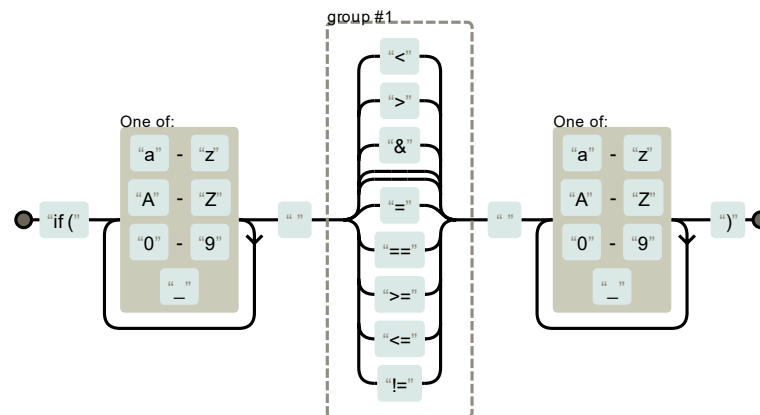




ESTRUCTURA DE DECISIÓN "IF"

Expresión Regular:

$if \backslash ([a - zA - Z0 - 9_]+ (< | > | \& | | = | > = | < = | ! =) [a - zA - Z0 - 9_]+ \backslash)$

Autómata:



 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

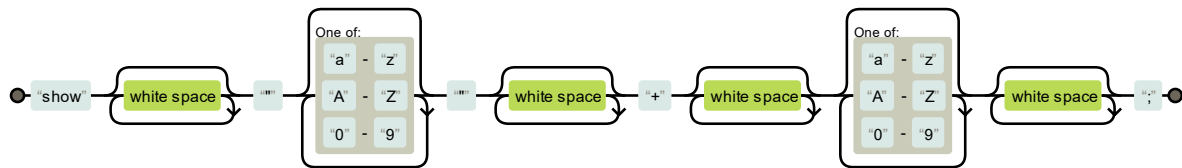
Handwritten signatures and initials.

COMANDO DE IMPRESIÓN EN PANTALLA “SHOW”

Expresión Regular:

`show\s *\"[a - zA - Z0 - 9] *\"s *+ \s * [a - zA - Z0 - 9] * \s *;`

Autómata:



6. IMPLEMENTACIÓN DEL COMPILADOR

Para realizar este proyecto, se utilizó una librería muy importante para poder presentar de manera gráfica la caja de texto en donde se escribe el código, la consola para presentar resultados y errores y la tabla de símbolos en donde se almacenan los diferentes elementos o símbolos encontrados en cada compilación.



Esta librería es Swing, exclusiva de Java, la cual permite crear una interfaz gráfica para las aplicaciones de Java.

6.1 MENÚ SUPERIOR

Es con esta librería que, para la parte del menú superior de la aplicación, se implementó un componente llamado JMenuBar que, como su nombre lo indica, es una barra de menú que puede contener múltiples elementos, en el caso de este proyecto solo contiene tres elementos: “Archivo”, “Limpiar”, “Ayuda” y “Acerca de”

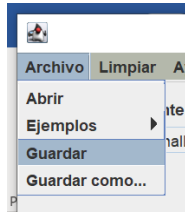
Al hacer clic en cada una de estas opciones, se mostrará un menú desplegable con diversas opciones. La primera de estas opciones, llamada *Archivo*, muestra cuatro opciones, las cuales son:

1. **Abrir:**
Al seleccionarla, abre una ventana del explorador de archivos que permite seleccionar un archivo .txt para cargar su código contenido al compilador para poder realizarle modificaciones o ejecutarlo.
2. **Ejemplos:**
Ejemplos es otro menú desplegable que, al pasar el cursor sobre él, muestra tres opciones, cada una de las cuales, si se presiona, cargará un archivo de ejemplo en el editor de código. El código cargado estará en el lenguaje definido por los alumnos.
3. **Guardar:**
Guarda el código que esté en el editor de texto. La opción ‘Guardar’ tiene dos comportamientos: En caso de que el código que se cargó al editor de texto proviniera de un archivo existente, entonces se guardará en ese mismo archivo que se abrió anteriormente; Pero en caso de que no se haya abierto ningún archivo, entonces desplegará una ventana para guardar un nuevo archivo.
4. **Guardar cómo:**
Guarda el código que esté dentro del editor de texto como un nuevo archivo o, en caso de que se prefiera, seleccionar uno ya existente.

 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

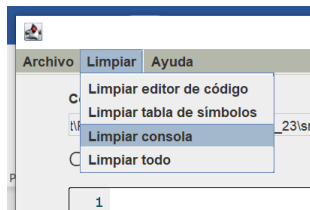
[Handwritten signatures]

5. ¿Cómo se ve? Opción Archivo:





La segunda opción disponible en el menú superior del compilador se llama *Limpiar*. Limpiar sirve para administrar el estado de la consola, tabla de símbolos y del editor de texto, es decir, si borrar el contenido de la consola, el editor de texto, la tabla de símbolos o el de todos al mismo tiempo, a continuación, se detalla más cómo funciona esto:

- Limpiar editor de código:**
Al seleccionar esta opción, todo aquello que se encuentre en el editor de código o texto será borrado para regresar al estado normal al editor, es decir, a un estado vacío.
- Limpiar tabla de símbolos:**
Borra todo el contenido de la tabla de símbolos, esto incluye a la información almacenada por compilaciones anteriores.
- Limpiar consola:**
Borra todo el texto que se haya presentado en la consola y, al igual que el punto anterior, también borrará datos o resultados de compilaciones pasadas.
- Limpiar todo:**
Borra al mismo tiempo el contenido de todos estos componentes de la interfaz, dejándolos vacíos y listos para nuevas compilaciones.
- ¿Cómo se ve? Opción Limpiar:**



La tercera opción de este menú se llama *Ayuda*, al hacer clic sobre ésta, se desplegará un menú con cuatro opciones diferentes, cada una con diferente información. El propósito de esta opción es reenviar al usuario a determinado archivo PDF (manual teórico hecho por los mismos estudiantes), dependiendo del tópico en el que necesiten ayuda:

- Análisis Léxico:** Al presionar esta opción, el usuario será enviado a un documento PDF hecho por los estudiantes en donde se documenta todo el fundamento teórico y parte del práctico del análisis léxico.
- Análisis Sintáctico:** Al igual que la opción anterior, se reenviará al usuario a un PDF con bases teóricas y prácticas sobre el análisis sintáctico, donde incluso vienen ejemplos de YACC o LEX.
- Análisis Semántico:** Del mismo modo que los anteriores, se mostrará un PDF con información

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

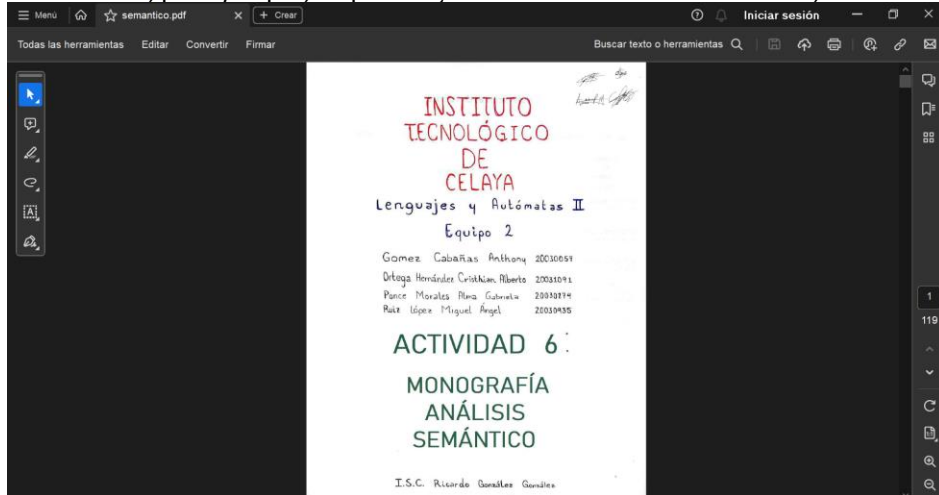
[Handwritten signatures]

sobre el análisis semántico.

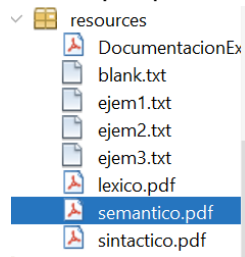
4. **Compilador:** Muestra un PDF con información sobre el compilador.
5. **¿Cómo se ve? Opción Ayuda:**



Si el usuario, por ejemplo, requiere ayuda sobre el análisis semántico, se le mostrará esto:





Esto es porque el PDF ya está guardado dentro de los recursos del proyecto en Java:



Finalmente, la cuarta y última opción que se muestra en el menú superior es la de Acerca de:

Acerca de

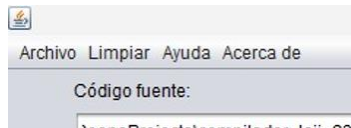
Contiene información sobre el equipo de desarrollo del presente proyecto. La información incluida es sobre los nombres de los integrantes, la materia para la cual se desarrolló el compilador, el nombre de la escuela y el número de equipo. La vista es la siguiente:

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

Handwritten signatures and initials.



Al final, el menú del compilador debe verse así:

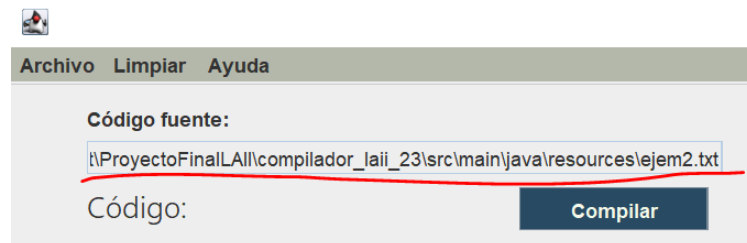


6.2 ENCABEZADO DE LA VISTA

En el encabezado de esta interfaz gráfica se pueden encontrar varios elementos, de entre los cuales destacan:



1. TextField:

Este TextField muestra la ruta origen del archivo importado (en caso de que se haya cargado un archivo). Está deshabilitado, por lo que no es posible modificarlo en sí.



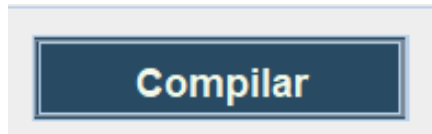
2. Botón Compilar:

Cuando se presiona este botón, inicia el proceso de compilación y los tres análisis comienzan a analizar el código proporcionado. Todo esto teniendo como código fuente al código introducido en el editor de código. El proceso de compilación inicia con el análisis léxico, si el

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures and initials]

análisis léxico detecta algún fallo como un carácter desconocido, detiene el proceso y no permite avanzar al siguiente análisis; si no ocurre ningún fallo, entonces se prosigue con el análisis siguiente



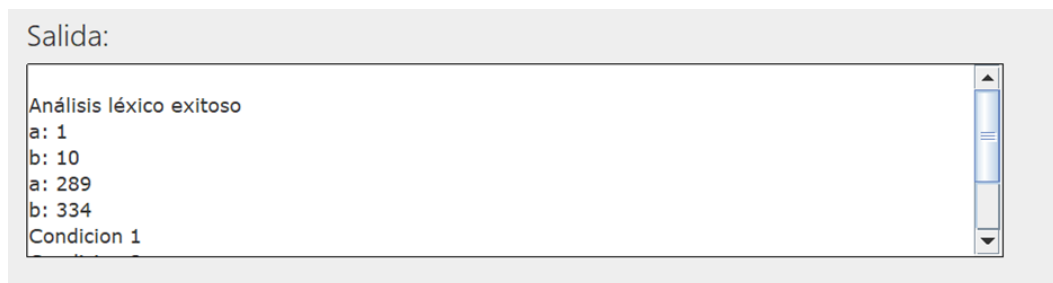
Los componentes o recursos que se ven “afectados” por presionar este botón son directamente la tabla de símbolos, la consola y el semáforo de los análisis, pues funcionan al pasarles datos.

6.3 CUERPO DE LA INTERFAZ GRÁFICA

El cuerpo de la aplicación, además del editor de código, cuenta con tres elementos muy importantes a la hora del desarrollo del compilador, estos tres componentes son:

1. Consola:

La consola es la encargada de mostrar la información del código fuente proveniente del editor de códigos. Por ejemplo, muestra si el código no presentó errores y se ejecutó correctamente y, en caso de que sí mostrara errores, la consola debe desplegar información sobre los errores como el código de errores y en qué análisis ocurrió el error.



2. Tabla de Símbolos:

La tabla de símbolos es un componente cuya función principal es recoger los tokens creados durante el análisis léxico y guardarlos en la tabla de símbolos, acomodándolos según sus campos y valores.



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II

AUTORES:

Alma Gabriela Ponce Morales
Anthony Gómez Cabañas
Cristhian Alberto Ortega Hernández
Miguel Ángel Ruiz López

Tabla de símbolos:

ID	No. Token	Token	Descripción	Lexema
1	20	PRINCIPAL	Palabra reserv...	principal
2	61	BLOQUEAPE...	Bloque apertura	{
5	11	INT	Palabra reserv...	int
6	75	VARIABLE	Variable	a
7	51	OPASIGNACI...	Operador de a...	=
9	60	DELIMITADOR	Delimitador	:
15	11	INT	Palabra reserv...	int
16	75	VARIABLE	Variable	b
17	51	OPASIGNACI...	Operador de a...	=
19	60	DELIMITADOR	Delimitador	:
25	13	BOOLEANO	Palabra reserv...	booleano
26	75	VARIABLE	Variable	ban
27	51	OPASIGNACI...	Operador de a...	=
28	75	VARIABLE	Variable	true
29	60	DELIMITADOR	Delimitador	:
35	13	BOOLEANO	Palabra reserv...	booleano
36	75	VARIABLE	Variable	opc
37	51	OPASIGNACI...	Operador de a...	=
38	75	VARIABLE	Variable	false
39	60	DELIMITADOR	Delimitador	:
45	4	SHOW	Palabra reserv...	show
48	60	DELIMITADOR	Delimitador	:
53	4	SHOW	Palabra reserv...	show
56	60	DELIMITADOR	Delimitador	:
62	51	OPASIGNACI...	Operador de a...	=
64	60	DELIMITADOR	Delimitador	:
70	51	OPASIGNACI...	Operador de a...	=

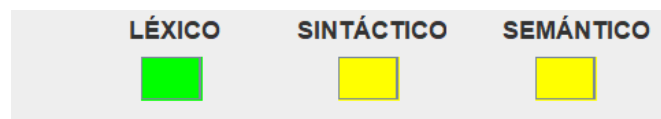
3. Semáforo:

El semáforo es un elemento importante que se había solicitado implementar desde inicios del proyecto, su función principal es la de monitorear si los diferentes análisis están funcionando correcta y secuencialmente. El semáforo tiene tres estados: Verde es bien, amarillo es esperando y rojo/naranja es que se detuvo el 'proceso.

Verde:

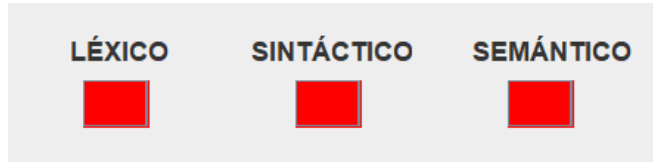


Amarillo:



[Handwritten signatures]

Rojo:



6.4 RESULTADOS

A continuación, se mostrará un ejemplo de cómo el compilador se ejecuta si no hay errores en su código fuente:

Archivo Limpiar Ayuda

Código fuente:
 ?ProyectoFinal\AI\compilador_lai_23\src\main\java\resources\ejem1.txt

Código: Compilar

```

1 principal {
2   int a = 10 ;
3   int b = 20 ;
4   if (a > b) {
5     show "variable" ;
6   }
7   aslong (a < b) {
8     a = a+1 ;
9   }
10  @@@ Esto es un comentario
11 }
12
```

PROYECTO - LAII

LÉXICO

SINTÁCTICO

SEMÁNTICO

Salida:



```

Análisis léxico exitoso
Análisis sintáctico exitoso
Análisis semántico exitoso
```

Tabla de símbolos:

ID	No. Token	Token	Descripción	Lexema
1	20	PRINCIPAL	Palabra reserv...	principal
2	61	BLOQUEAPER...	Bloque apertura	{
5	11	INT	Palabra reserv...	int
6	75	VARIABLE	Variable	a
7	51	OPASIGNACION	Operador de a...	=
9	60	DELIMITADOR	Delimitador	:
15	11	INT	Palabra reserv...	int
16	75	VARIABLE	Variable	b
17	51	OPASIGNACION	Operador de a...	=
19	60	DELIMITADOR	Delimitador	:
25	1	IF	Palabra reserv...	if
27	41	OPCOMPARA...	Operador de c...	>
29	61	BLOQUEAPER...	Bloque apertura	{
35	4	SHOW	Palabra reserv...	show
37	60	DELIMITADOR	Delimitador	:
41	62	BLOQUECIER...	Bloque cierre	}
43	3	ASLONG	Palabra reserv...	aslong
45	41	OPCOMPARA...	Operador de c...	<
47	61	BLOQUEAPER...	Bloque apertura	{
54	51	OPASIGNACION	Operador de a...	=
56	60	DELIMITADOR	Delimitador	:
61	62	BLOQUECIER...	Bloque cierre	}
63	62	BLOQUECIER...	Bloque cierre	}

Y si se ejecuta el compilador con un error en el análisis sintáctico, mandará el siguiente error:

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

Archivo Limpiar Ayuda

Código fuente:
!ProyectoFinalLAI\compilador_lai_23\src\main\java\resources\ejem2.txt

Código:

```

1 principal {
2   ñ
3   int a = 1;
4   int b = 10;
5   booleano ban = true;
6   booleano opc = false;
7   show "a: "+a;
8   show "b: "+b;
9   a = b*(23+6)-1;
10  b = 2*23+a-1;
11  show "a: "+a;
12  show "b: "+b;
13  if (a<b|b>=a) {
14    show "Condicion 1";
15  }
16  if (ban&opc) {
17    show "Condicion 2";
18  }

```

Salida:

Línea 2
 Carácter no permitido [ñ] con código de error: 41

PROYECTO - LAII



LÉXICO ☒ SINTÁCTICO ☒ SEMÁNTICO ☒

Tabla de símbolos:

ID	No. Token	Token	Descripción	Lexema
1	20	PRINCIPAL	Palabra reservada	principal
2	61	BLOQUEAPER...	Bloque apertura	{

Así es como funciona el compilador desarrollado por el equipo dos, realmente tiene muchos más resultados que los mostrados en el documento, pero con esos dos se puede ejemplificar bien su uso del proyecto.

7 BÍTACORA DE INCIDENCIAS		
Fecha	Problema encontrado	Solución
16/11/2023	Los errores mostrados en consola se estaban estructurando todos en la misma línea.	El equipo se dio cuenta que se estaba utilizando un Label que no era óptimo para mostrar este tipo de mensajes de error extensos, por lo que se decidió cambiarlo por un TextArea inhabilitado que impidiera modificaciones por el usuario.
17/11/2023	El análisis sintáctico devolvía muchos errores de una misma línea que, según lo que definimos, estaba sintácticamente correcta.	Se revisó la clase del analizador sintáctico cómo estaban siendo identificados los tokens y cómo afectaba esto al momento del análisis sintáctico.
19/11/2023	Durante la ejecución del análisis sintáctico se presentó la problemática de una revisión ciclada de los errores que podrían presentarse en la estructura del programa	Se hizo un depurado del código para saber en qué parte del código se estaba realizando el ciclado de revisión
21/11/2023	Al declarar una variable con un tipo de dato incompatible, era considerada como un token indefinido más adelante en la ejecución del programa.	Verificar con ayuda del depurador cómo se estaba identificando la variable en la comparación de errores donde se descubrió que era considerado un



 TECNOLÓGICO NACIONAL DE MÉXICO	INSTITUTO TECNOLÓGICO DE CELAYA		
PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II		AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López	








		carácter a imprimir. Entonces, en esta estructura, se implementó que identificara si era una letra para poder retornar un error más adecuado.
21/11/2023	Al compilar el código proporcionado y ejecutar el análisis léxico, no se detenía la ejecución del compilador al detectar un error.	Después de realizar el proceso y retornar su estatus, se agregó una sentencia return para detener el proceso de compilación.
23/11/2023	En el análisis léxico, al generar la tabla de símbolos, se detectó que los operadores aritméticos no eran debidamente mostrados porque una operación aritmética entera se tomaba como un token.	Se realizó una separación de lexemas para poder extraer el operador aritmético existente en la cadena.
25/11/2023	Los comentarios y los símbolos “@@” que los denotan eran mostrados en la tabla de símbolos, cosa que no debía suceder porque los comentarios existen en parte para ser ignorados por el compilador	Se identificó la presencia del token “@@” que indicaba que lo posterior a ese era una línea de comentario e ignorara todos los tokens siguientes.

8	CONCLUSIÓN
<p>En este proyecto, se desarrolló un lenguaje de programación y su compilador desde cero. Se optó por evitar el uso de herramientas externas, creando el equipo así herramientas propias para dar soluciones en cada fase del análisis. Desde la inclusión de elementos básicos en el lenguaje hasta la verificación en el análisis léxico, se buscó ofrecer a los usuarios una experiencia completa.</p> <p>La atención se centró en el análisis sintáctico, respaldado por herramientas y notaciones rigurosas. Además, se implementó una interfaz gráfica, una consola detallada y una tabla de símbolos para mejorar la experiencia del usuario y la transparencia en el proceso de compilación. Este proyecto proporciona una visión detallada de la optimización de compiladores para diferentes lenguajes de programación.</p> <p>Habiendo realizado lo propuesto, concluimos este proyecto con un profundo sentido de logro y la confianza en haber superado un desafío técnico y creativo significativo.</p>	

 <p>TECNOLÓGICO NACIONAL DE MÉXICO</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures and initials]

9	REFERENCIAS
	<p>[1] RicardoGeek, & RicardoGeek. (2016, 15 diciembre). Fases del compilador RicardoGeek. RicardoGeek. https://ricardogeek.com/fases-del-compilador/</p> <p>[2] Antecedentes de compiladores. (s. f.). Scribd. https://es.scribd.com/document/56016543/Antecedentes-de-compiladores</p> <p>[3] Navaz-López, E. (2022). Implementación de un Compilador Didáctico para un súper-conjunto de PL/o. Disponible en: https://arxiv.org/pdf/2207.08972.pdf</p> <p>[4] Contexto de un compilador – compiladores. (s. f.). Compiladores. https://loscompiladores.wordpress.com/category/contexto-de-un-compilador/</p> <p>[5] Gonzalvez, A., Alicia, M. (2016). El contexto, elemento de análisis para enseñar. Disponible en: https://www.redalyc.org/pdf/853/85350504004.pdf</p> <p>[6] Universitat Jaume I. (2016). Ingeniería Informática II26 Procesadores de lenguaje Estructura de los compiladores e intérpretes. Disponible en: https://repositori.uji.es/xmlui/bitstream/handle/10234/5876/estructura.apun.pdf</p> <p>[7] R, J. L. (2019, 21 febrero). Compilador - que es, como funciona y fases. 247 Tecno. https://247tecno.com/compilador-que-es-como-functiona-fases/</p>

 <p>TECNOLÓGICO NACIONAL DE MÉXICO®</p>	<p>INSTITUTO TECNOLÓGICO DE CELAYA</p>	
<p>PRÁCTICA DE LABORATORIO - LENGUAJES Y AUTÓMATAS II</p>	<p>AUTORES: Alma Gabriela Ponce Morales Anthony Gómez Cabañas Cristhian Alberto Ortega Hernández Miguel Ángel Ruiz López</p>	

[Handwritten signatures and initials]

<p>[8] GeeksforGeeks. (2023, 8 noviembre). Phases of a compiler. https://www.geeksforgeeks.org/phases-of-a-compiler/</p>
--