Universidad Autónoma del Estado de Hidalgo

ICBI

2.4 Análisis sintáctico. Ejercicios

PROFESOR:

Dr. Eduardo Cornejo-Velázquez

ALUMNO:

Mayte Eridani Hernández Carlos

1. Introducción

En el tema 3.9 del capítulo 3 se abordan los ejercicios prácticos relacionados con el análisis sintáctico en la fase de compilación. Estos ejercicios tienen como objetivo permitir que los estudiantes comprendan y apliquen los conceptos clave del análisis sintáctico, como la construcción de árboles sintácticos, la manipulación de gramáticas y la verificación de cadenas de acuerdo con reglas gramaticales. El análisis sintáctico es esencial en el proceso de compilación, ya que determina si la secuencia de tokens generada por el análisis léxico es válida según las reglas gramaticales del lenguaje fuente.

2. Marco teórico

El análisis sintáctico es la fase del compilador que organiza los componentes léxicos (tokens) en estructuras jerárquicas llamadas árboles sintácticos. Estos árboles representan la relación gramatical entre los componentes del programa, según las reglas de la gramática definida para el lenguaje de programación. Para la correcta implementación de un analizador sintáctico, se utilizan varias técnicas y estructuras de datos, entre ellas las gramáticas libres de contexto, árboles de derivación y diagramas de transición de estados.
Las gramáticas libres de contexto (CFG) son fundamentales en el análisis sintáctico, ya que proporcionan una especificación formal para la sintaxis de un lenguaje de programación. Los árboles de análisis sintáctico son estructuras que se utilizan para representar cómo se descompone una expresión o instrucción en sus componentes básicos, basándose en la gramática del lenguaje. En los ejercicios del tema, se exploran diversas formas de representar y manipular estas estructuras, además de enfrentarse a situaciones como la ambigüedad y la eliminación de recursión por la izquierda, que son comunes al trabajar con gramáticas.

3. Herramientas a Utilizar:

Descripción	**Especificaciones**	**Observaciones**
Lenguaje de programación	Java	
Librerías	Graphics, Image, ImageIcon, Swing	
Entorno de desarrollo	NetBeans IDE	Versión 22/24

Table 1: Herramientas empleadas en el ejercicio

4. Desarrollo

3.9 Ejercicios y actividades

Ejercicio 1

1. a) Escriba una gramática que genere el conjunto de cadenas

$$\{s;, s; s;, s; s; s; s;, \dots\}$$

$$Gramática : S- > s; S|s;$$

b) Genere un árbol sintáctico para la cadena $s; s;$

Árbol Sintáctico

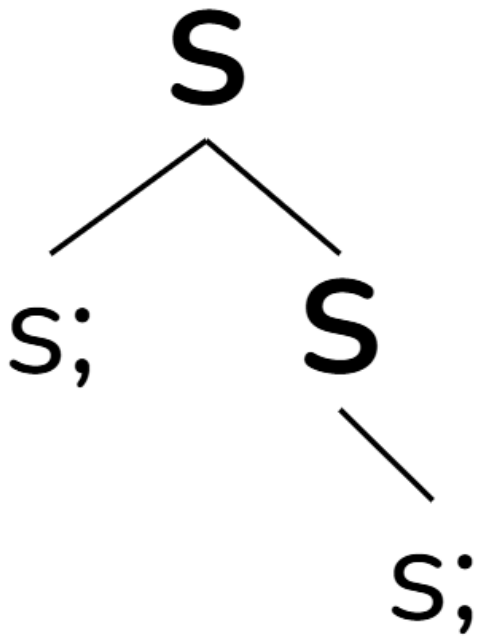

Figure 1: Árbol sintáctico generado

Ejercicio 2

2. Considere la siguiente gramática:
rexp → rexp "—" rexp
— rexp rexp
— rexp "*"
— "(" rexp ")"
— letra

a) Genere un árbol sintáctico para la expresión regular (ab—b)*.

Árbol Sintáctico

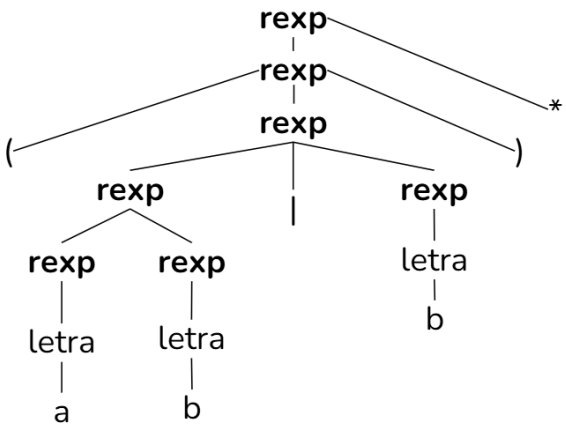

Figure 2: Árbol sintáctico generado

Ejercicio 3

3. De las siguientes gramáticas, describa el lenguaje generado por la gramática y genere árboles sintácticos con las respectivas cadenas.
a) S → S S + — S S * — a con la cadena aa+a*.

Está gramática nos permite colocar dos "S" consecutivas, seguidas de un carácter "+" o dos "S" consecutivas, seguidas de un carácter "+" o en el ultimo caso, colocar solo una "a";

Árbol Sintáctico

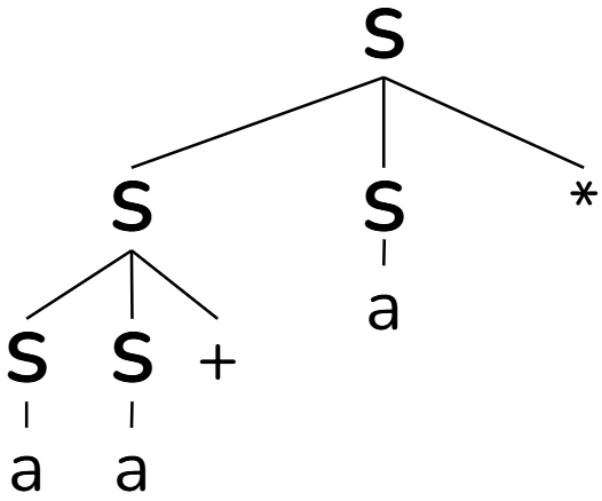

Figure 3: Árbol sintáctico generado

b) S → 0 S 1 — 0 1 con la cadena 000111.

Está gramática permite generar cadenas que simplemente contengan 0 1 o que inicien con 0, seguidas de "S" y finalizen con 1;

Árbol Sintáctico

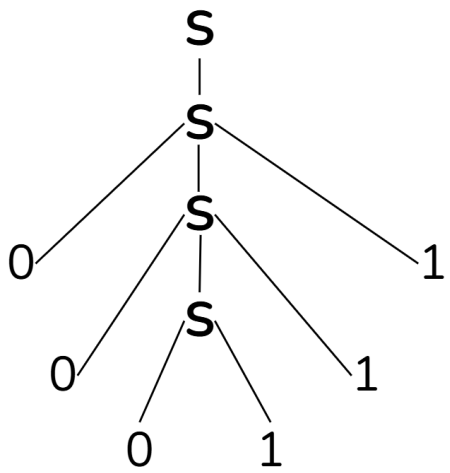

Figure 4: Árbol sintáctico generado

c) S → + S S — * S S — a con la cadena + * aaa

Está gramática permite generar cadenas que contengan "+" seguido de dos "S" o cadenas que empiecen por "*" seguido de dos "S" o en el ultimo caso solo "a";

Árbol Sintáctico

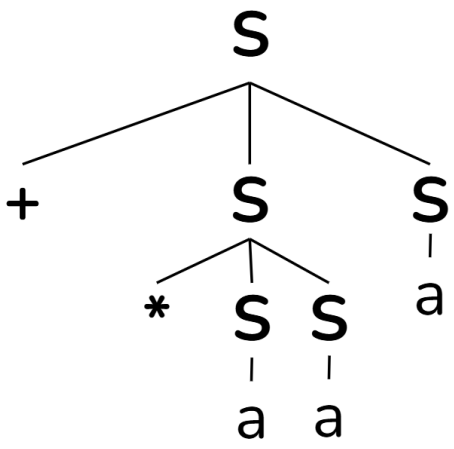

Figure 5: Árbol sintáctico generado

Ejercicio 4

¿Cuál es el lenguaje generado por la siguiente gramática? S → xSy — e

La gramatica nos permite aceptar cadenas que comiencen con "X", seguidos de "S" y finalizadas con "Y" o en caso contrario no contener ningun caracter, como por ejemplo, se aceptara la cadena: "", XY, XXYY, XXXYYY.

Ejercicio 5

5. Genere el árbol sintáctico para la cadena zazabzbz utilizando la siguiente gramática:

S → zMNz
M → aNa
N → bNb
N → z

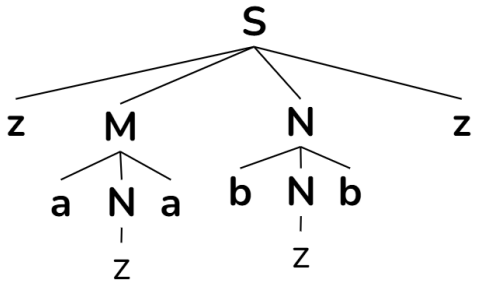

Figure 6: Árbol sintáctico generado

Ejercicio 6

6. Demuestre que la gramática que se presenta a continuación es ambigua, mostrando que la cadena ictictses tiene derivaciones que producen distintos árboles de análisis sintáctico.
S → ictS
S → ictSeS
S → s

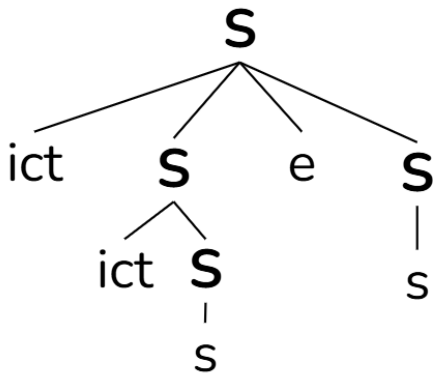

Figure 7: Primer árbol sintáctico generado

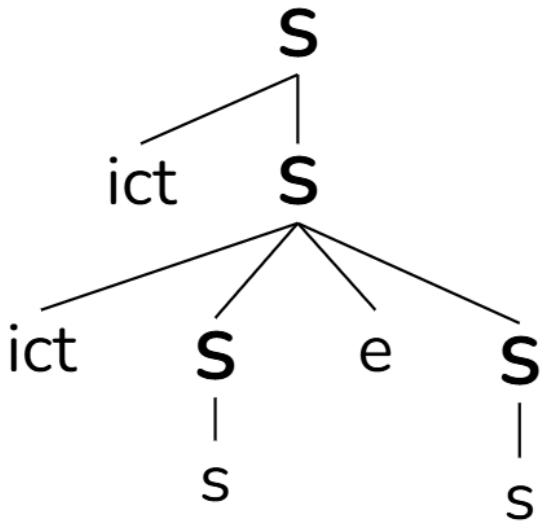

Figure 8: Segundo árbol sintáctico generado

Ejercicio 7

7. Considere la siguiente gramática
S → (L) — a

L → L , S — S

Encuéntrense árboles de análisis sintáctico para las siguientes frases:

a) (a, a)

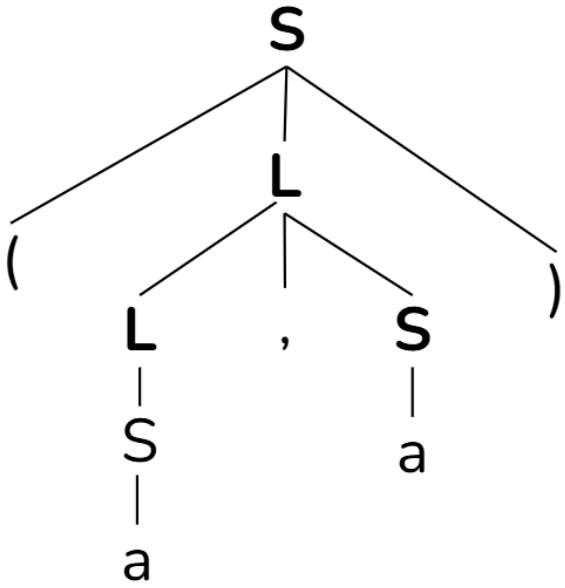

Figure 9: Árbol sintáctico generado

b) (a, (a, a))

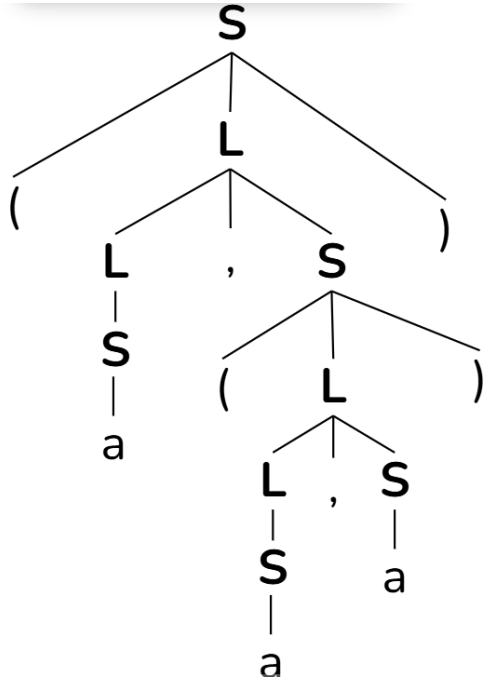

Figure 10: Árbol sintáctico generado

c) (a, ((a, a), (a, a)))

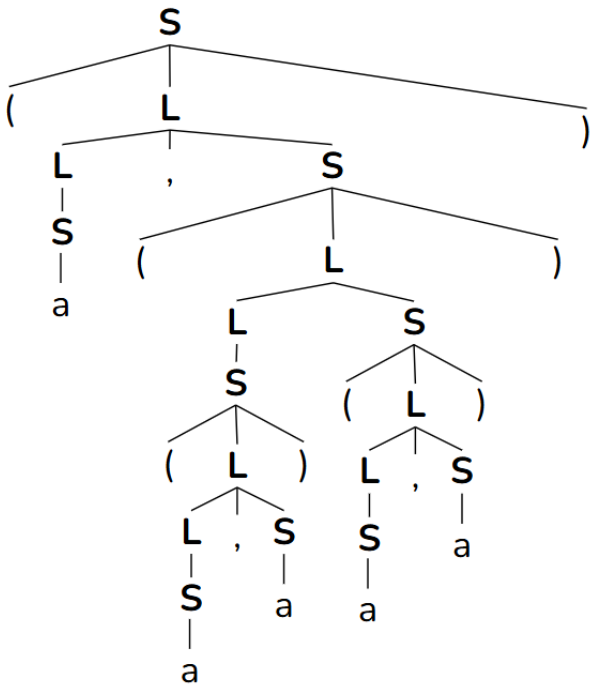

Figure 11: Árbol sintáctico generado

Ejercicio 8

8. Constrúyase un árbol sintáctico para la frase not (true or false) y la gramática:

bexpr → bexpr or bterm — bterm
bterm → bterm and bfactor — bfactor
bfactor → not bfactor — (bexpr) — true — false

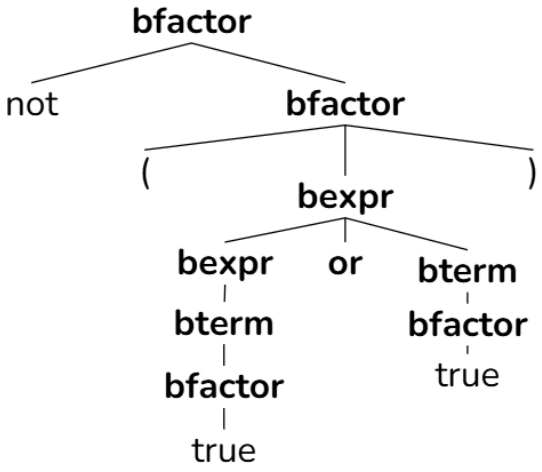

Figure 12: Árbol sintáctico generado

Ejercicio 9

9. Diseñe una gramática para el lenguaje del conjunto de todas las cadenas de símbolos 0 y 1 tales que todo 0 va inmediatamente seguido de al menos un 1.
S → 0U — U
U → 1S — 1

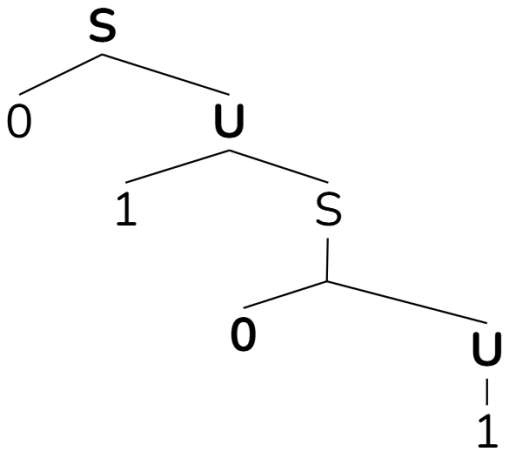

Figure 13: Árbol sintáctico generado

Ejercicio 10

10.Elimine la recursividad por la izquierda de la siguiente gramática:

S → (L) — a
L → L , S — S

Gramática sin recursividad

S → (L) — a
L → , S L´ — S

Ejercicio 11

11. Dada la gramática S → (S) — x, escriba un pseudocódigo para el análisis sintáctico de esta gramática mediante el método descendente recursivo.

Función S():
Si el símbolo actual es '(':
Avanzar en el buffer (mover al siguiente símbolo)
Llamar a la función S()
Si el símbolo actual es ')':
Avanzar en el buffer
Sino:
Mostrar error "Se esperaba ')""
Sino si el símbolo actual es 'x':
Avanzar en el buffer
Sino:
Mostrar error "Símbolo no válido"

INICIO
Llamar a S() si la entrada esta vacía: retornar éxito (cadena valida)
SINO retornar error (entrada incompleta)
FIN

Ejercicio 12

12. Qué movimientos realiza un analizador sintáctico predictivo con la entrada (id+id)*id, mediante el algoritmo 3.2, y utilizándose la tabla de análisis sintáctico de la tabla 3.1. (Tómese como ejemplo la Figura 3.13).

No terminal	Símbolo de entrada					
	id	+	*	(	)	$
E	E → TE ‘			E → TE ‘		
E ‘		E ‘→ +TE ‘			E ‘ → ε	E ‘ → ε
T	T → FT ‘			T → FT ‘		
T ‘		T ‘ → ε	T ‘→ *FT ‘		T ‘ → ε	T ‘ → ε
F	*F* → id			*F* → (*E*)		

Figure 14: Tabla de análisis sintáctico para la gramática 3.3

Pila	Entrada	Acción
$E	(id+id)*id$	E ->TE'
$E'T	(id+id)*id$	T->FT'
$E'T'F	(id+id)*id$	F->(E)
$E'T')E(	(id+id)*id$	concuerda (
$E'T')E	id+id)*id$	E ->TE'
$E'T')E'T	id+id)*id$	T->FT'
$E'T')E'T'F	id+id)*id$	F->id
$E'T')E'T'	id+id)*id$	concuerda ID
$E'T')E'T'id	+id)*id$	T->e
$E'T')E'T'	+id)*id$	E' ->TE'
$E'T')E'	+id)*id$	concuerda +
$E'T')E'T+	id)*id$	T->FT'
$E'T')E'T	id)*id$	F->ID
$E'T')E'T'F	id)*id$	concuerda ID
$E'T')E'T'id	id)*id$	T'->e
$E'T')E'T'	)*id$	E'->e
$E'T')E'	)*id$	concuerda)
$E'T')	*id$	T ->*FT
$E'TF*	id$	concuerda *
$E'T'F	*id$	F -> ID
$E'T'id	*id$	concuerda id
$E'T'	$	T-> e
$E'	$	E' ->e
$	$	aceptar

Figure 15: Movimientos que realiza un analizador sintáctico predictivo con la entrada (id + id) * id

Ejercicio 13

13. La gramática 3.2, sólo maneja las operaciones de suma y multiplicación, modifique esa gramática para que acepte, también, la resta y la división; Posteriormente, elimine la recursividad por la izquierda de la gramática completa y agregue la opción de que F, también pueda derivar en num, es decir, F → (E) — id — num.

Gramática Original: E → E + T — T
Gramática Modificada: E → E + T — E - T — T

Gramática Original: T → T * F — F
Gramática Modificada: T → T * F — T / F — F

Gramática Original: F → (E) — id
Gramática Modificada: F → (E) — id — num

Gramática Modificada: E → E + T — E + T — T
Gramática sin recursividad: E → + T E' — - T E' — e

E' → TE'
Gramática Modificada: T → T * F — T / F — F

Gramática sin recursividad: T → * F T' — / F T' — e

T → FT'

Ejercicio 14

14. Escriba un pseudocódigo (e implemente en Java) utilizando el método descendente recursivo para la gramática resultante del ejercicio anterior (ejercicio 13).

Procedimiento E():
Si T() entonces
Mientras token actual es '+' o '-' hacer:
Si token actual es '+' entonces
Consumir token '+' y analizar T()
Sino si token actual es '-' entonces
Consumir token '-' y analizar T()
Fin Mientras
Fin Si

Procedimiento T():
Si F() entonces
Mientras token actual es '*' o '/' hacer:
Si token actual es '*' entonces
Consumir token '*' y analizar F()
Sino si token actual es '/' entonces
Consumir token '/' y analizar F()
Fin Mientras
Fin Si

Procedimiento F():
Si token actual es '(' entonces
Consumir token '(' y analizar E()
Si token actual es ')' entonces
Consumir token ')'
Sino Error: Se esperaba ')'
Sino si token actual es 'id' entonces
Consumir token 'id'
Sino si token actual es 'num' entonces
Consumir token 'num'
Sino
Error: Se esperaba '(' o 'id' o 'num'

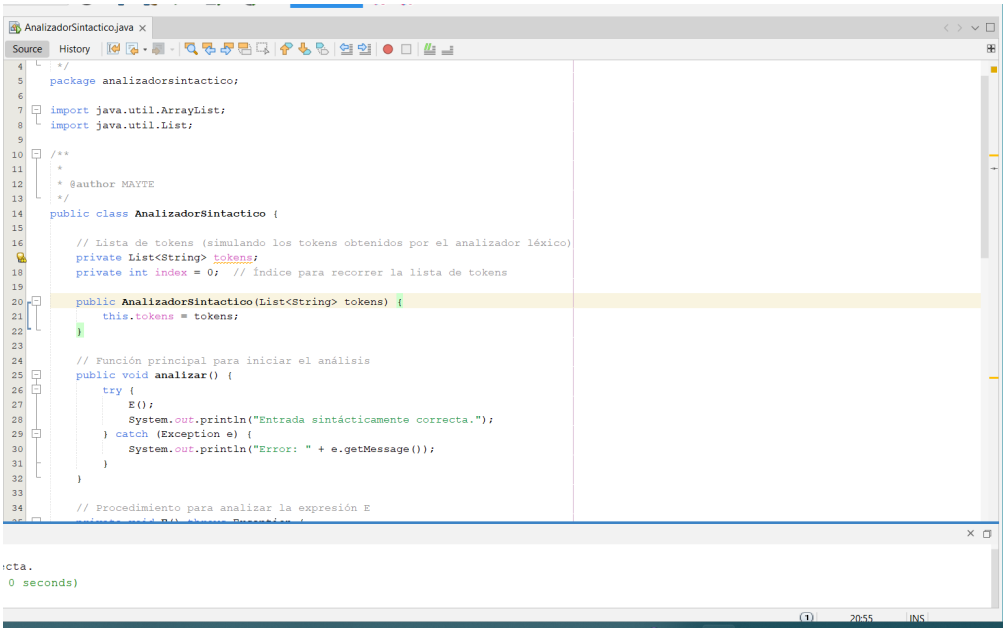

Figure 16: Código en Java

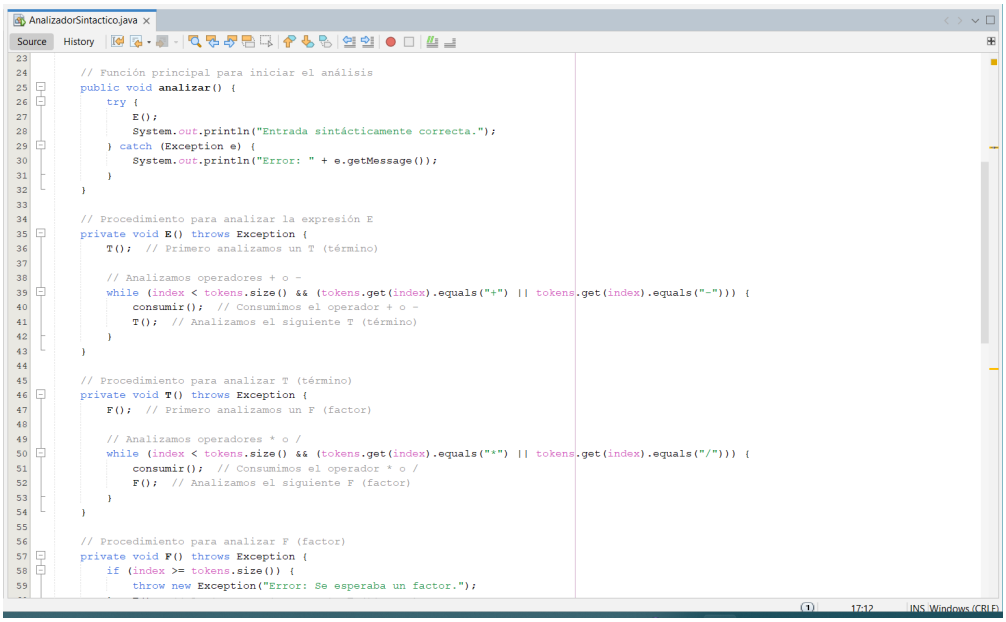

Figure 17: Código en Java

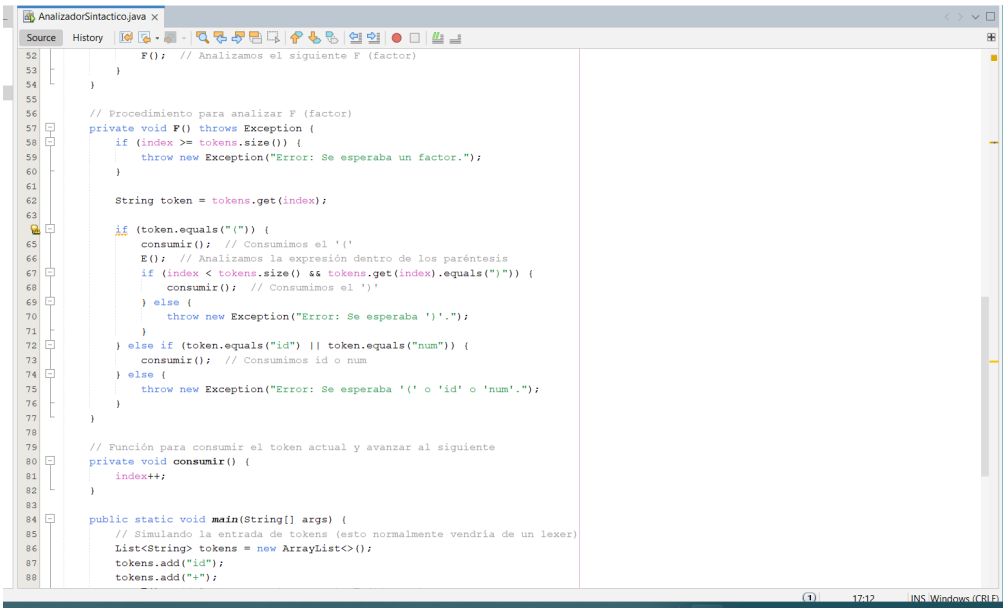

Figure 18: Código en Java

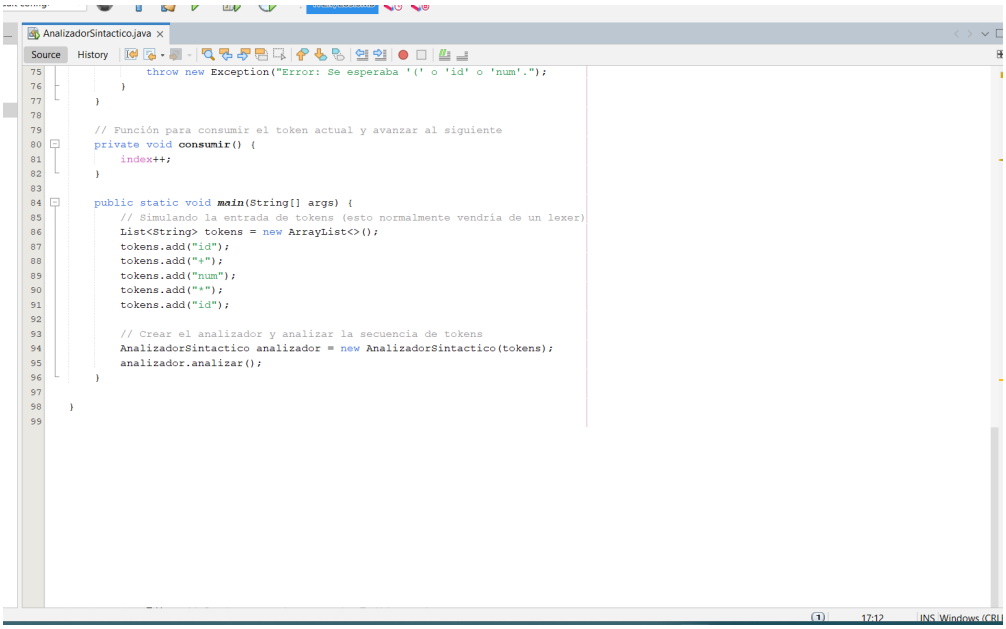

Figure 19: Código en Java

5. Conclusiones

Con el desarrollo de esta actividad se logro reafirmar el conocimiento obtenido y plasmado en el capitulo 3 del libro "Compiladores Fases de análisis".
A través de la elaboración de árboles de análisis para cadenas y la resolución de problemas relacionados con gramáticas y expresiones regulares, pude mejorar mi comprensión sobre cómo interpretar y validar secuencias de componentes léxicos. Además, los ejercicios me ayudaron a identificar soluciones a las gramáticas propuestas, lo que me permitió mejorar la comprensión de cómo se construyen y ajustan estas reglas para generar un análisis sintáctico preciso. De esta manera, pude aplicar de forma práctica los métodos de análisis utilizados en la construcción de compiladores, familiarizándome con los procesos necesarios para realizar un análisis adecuado y depurar errores en las gramáticas.

Referencias Bibliográficas

References

[1] Carranza Sahagún, D. U. (Coord.). (2024). Compiladores: fases de análisis. Editorial Transdigital. https://doi.org/10.56162/transdigitalb44 .