



## DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **SOLICITUD DE ACTIVIDADES**

Celaya, Guanajuato, 18 / septiembre / 202

### LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ  
**SEMESTRE AGOSTO-DICIEMBRE 2023**

#### **ACTIVIDAD 4 ( VALOR 44 PUNTOS )**

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

EL LECTOR DEBE TOMAR MUY EN CUENTA QUE ESTA ACTIVIDAD ES UN EXAMEN, Y NO UNA SIMPLE TAREA, PUES DEMANDA DEDICACIÓN PARA INVESTIGAR, LEER, ANALIZAR, REDACTAR, ILUSTRAR Y PROPOSER DE MANERA PROFESIONAL LOS TEMAS PROPUESTOS EN LA ESTRUCTURA TEMÁTICA DE ESTA ASIGNATURA.

#### **2. ANÁLISIS LÉXICO.**

INVESTIGUE, LEA, COMPREnda Y ELABORE UNA **MONOGRAFÍA TÉCNICA** COMPLETAMENTE APEGADA A LO SOLICITADO EN LA [GUÍA TUTORIAL](#) (PUNTO 3, INCISO a ) ACERCA DE LOS SIGUIENTES TEMAS :

- GENERADORES DE ANALIZADORES LÉXICOS
- APLICACIONES ( CASO DE ESTUDIO )

CONSIDERACIÓN :

DEBE USTED ENTENDER EL VALOR QUE TIENE ESTA ACTIVIDAD Y QUE LOS TEMAS ANTES REFERIDOS, PARA NADA DEBEN SER ABORDADOS COMO SIMPLES CONCEPTOS REDACTADOS CON LA LIGEREZA QUE YA SE HA OBSERVADO EN ACTIVIDADES PREVIAS.

ANALICE CADA TEMA, SUS CARACTERÍSTICAS, SU IMPORTANCIA, SUS CONCEPTOS, SUS EJEMPLOS, SUS ILUSTRACIONES, Y LOS TIPOS DE EVIDENCIAS QUE USARÁ PARA DEMOSTRAR QUE USTED HA ADQUIRIDO UN VERDADERO CONOCIMIENTO ACERCA DE ÉSTOS.



Av. Antonio García Cubas #600 esq. Av. Tecnológico, Colonia Alfredo V. Bonfil, C.P. 38010  
Celaya, Gto. Tel. 01 (461) 611 75 75 e-mail: [lince@celaya.tecnm.mx](mailto:lince@celaya.tecnm.mx) [tecnm.mx](http://tecnm.mx) | [celaya.tecnm.mx](http://celaya.tecnm.mx)





**IMPORTANTE :** SI LO REQUIERE PUEDE CONSULTAR EL [SIGUIENTE DOCUMENTO](#) PARA ORIENTAR SU TRABAJO EN CONOCER QUÉ ES Y CÓMO HACER UNA MONOGRAFÍA CON EL RIGOR ACADÉMICO REQUERIDO.

EN LO REFERENTE AL TEMA **APLICACIONES ( CASO DE ESTUDIO )**, ARRIBA REFERIDO Y A MANERA DE PRÁCTICA, ELABORE CON EL APOYO DEL SOFTWARE LIBRE DENOMINADO **ANALIZADOR LÉXICO FLEX**, UN EJERCICIO DEMOSTRATIVO SOBRE AL MENOS TRES DEFINICIONES LÉXICAS PROPIAS DEL LENGUAJE PROGRAMACIÓN C.

### 3. ANÁLISIS SINTÁCTICO.

INVESTIGUE, LEA, COMPREnda Y ELABORE UNA **MONOGRAFÍA TÉCNICA** COMPLETAMENTE APEGADA A LO SOLICITADO EN LA [GUÍA TUTORIAL](#) (PUNTO 3, INCISO a ) ACERCA DE LOS SIGUIENTES TEMAS :

- GRAMÁTICAS LIBRES DE CONTEXTO Y ÁRBOLES DE DERIVACIÓN
- DIAGRAMAS DE SINTAXIS

CONSIDERACIÓN :

DEBE USTED ENTENDER EL VALOR QUE TIENE ESTA ACTIVIDAD Y QUE LOS TEMAS ANTES REFERIDOS, PARA NADA DEBEN SER ABORDADOS COMO SIMPLES CONCEPTOS REDACTADOS CON LA LIGEREZA QUE YA SE HA OBSERVADO EN ACTIVIDADES PREVIAS.

ANALICE CADA TEMA, SUS CARACTERÍSTICAS, SU IMPORTANCIA, SUS CONCEPTOS, SUS EJEMPLOS, SUS ILUSTRACIONES, Y LOS TIPOS DE EVIDENCIAS QUE USARÁ PARA DEMOSTRAR QUE USTED HA ADQUIRIDO UN VERDADERO CONOCIMIENTO ACERCA DE ÉSTOS.

A MODO DE PRÁCTICA REALICE ESTE PUNTO Y ELABORE EJERCICIOS PRÁCTICOS CON LOS CUÁLES USTED DEMUESTRE

- ¿ QUÉ SON LAS GRAMÁTICAS LIBRES DE CONTEXTO ?
- ¿ QUÉ ES UN CONTEXTO ?, HAGA UNA MUY BUENA ILUSTRACIÓN DE ESTE CONCEPTO.
- ¿ SIRVE DICHO CONTEXTO A LOS PROPÓSITOS DE UNA GRAMÁTICA QUE DEFINA UN LENGUAJE FORMAL ?

ADEMÁS INCLUYA EJEMPLOS ( AL MENOS TRES ) DE ÁRBOLES DE DERIVACIÓN Y DEMUESTRE CÓMO AYUDAN A LOS PROCESOS DE ANÁLISIS SINTÁCTICOS.

**IMPORTANTE :** SI LO REQUIERE PUEDE CONSULTAR EL [SIGUIENTE DOCUMENTO](#) PARA ORIENTAR SU TRABAJO EN CONOCER QUÉ ES Y CÓMO HACER UNA MONOGRAFÍA CON EL RIGOR ACADÉMICO REQUERIDO.

POR ÚLTIMO, RECUERDE LEER LA [GUÍA TUTORIAL](#) PARA EL CORRECTO TRATAMIENTO DE ESTE INCISO.





### ¿QUÉ SE CALIFICARÁ ?

LA RÚBRICA PARA EVALUAR ESTA ACTIVIDAD ESTARÁ INTEGRADA POR LOS SIGUIENTES CRITERIOS.

- a. **LA OPORTUNIDAD.** SI EL TRABAJO FUE ENTREGADO OPORTUNAMENTE.
- b. **LA COMPRENSIÓN.** SE VALORARÁ EL GRADO DE COMPRENSIÓN DEL TEMAS ANALIZADOS.
- c. **LA CALIDAD.** SI LAS EVIDENCIAS ENVIADAS CORRESPONDEN A LA CALIDAD ESPERADA PARA ESTE NIVEL PROFESIONAL QUE SE CURSA.
- d. **LA CAPACIDAD DE SÍNTESIS.** SI LAS EVIDENCIAS ENTREGADAS TIENEN EL NIVEL DE DETALLE Y PROFUNDIDAD REQUERIDA, O EN BIEN SI SE OMITIERON CONCEPTOS CON EL AFÁN DE SIMPLIFICAR Y ENTREGAR UN MATERIAL ACADÉMICA Y TÉCNICAMENTE POBRE.
- e. **LA CREATIVIDAD.** LA MANERA EN QUE SE EXPRESAN LOS CONCEPTOS Y EL TRATAMIENTO QUE SE DA A LA INFORMACIÓN ANALIZADA PARA QUE ÉSTA SEA COMPRESIBLE EN SU ESENCIA.

**IMPORTANTE :** CUENTA CON EL TIEMPO SUFFICIENTE PARA REALIZAR ESTA ACTIVIDAD Y SUMAR PUNTOS IMPORTANTES A SU CALIFICACIÓN DE ESTA EVALUACIÓN.

**IMPORTANTE :** TODO EL MATERIAL ESCRITO DEBERÁ SER HECHO A MANO.





## **CONSIDERACIONES.**

**CADA UNO DE LOS PUNTOS ANTERIORES DEBE SER DESARROLLADO CON LA PROFUNDIDAD ACORDE A UN NIVEL PROFESIONAL, Y APEGÁNDOSE COMPLETAMENTE A LAS DIRETRICES DE LA GUÍA TUTORIAL.**

**NO CONCIBA ESTE TRABAJO, COMO UN SIMPLE RESUMEN O EJERCICIO DE TRANSCRIPCIÓN, PUES EL VALOR INDICADO AL INICIO DE ESTA ACTIVIDAD LE DARÁ A USTED UNA BUENA IDEA DE LO QUE SE ESPERA DE ELLA, EN CUANTO A CALIDAD Y EL APRENDIZAJE OBTENIDO, MISMO QUE SERÁ PUESTO A PRUEBA MEDIANTE UN EXAMEN ESCRITO O BIEN ORAL EN CLASE.**

**SI DECIDIÓ ELABORAR ESTA ACTIVIDAD EN EQUIPO, CADA INTEGRANTE DE ÉSTE DEBERÁ POSEER EL MISMO NIVEL DE CONOCIMIENTO, PUES TAN SOLO REPARTIR TEMAS ENTRE LOS INTEGRANTES DEL EQUIPO, SUPONDRIÁ UN GRAVE ERROR DE INTERPRETACIÓN A LA INTENCIÓN DIDÁCTICA REAL DE ESTA ACTIVIDAD.**

**POR ÚLTIMO, ESTA ACTIVIDAD SOLO SE PODRÁ DESARROLLAR EN EQUIPO, SI SE REGISTRÓ EN UNO PREVIAMENTE, UTILIZANDO EL FORMATO ENTREGADO EN LA ACTIVIDAD INICIAL. DE LO CONTRARIO DEBERÁ ELABORAR Y ENTREGAR LA ACTIVIDAD DE FORMA INDIVIDUAL.**

LA ENTREGA DE DICHO REGISTRO SE HARÁ VÍA CORREO ELECTRÓNICO ENVIANDO ÉSTE AL PROFESOR DESIGNADO, Y POSTERIORMENTE EN CLASE ENTREGANDO LA HOJA EN FÍSICO.

## **OBSERVACIONES:**

- CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO, INCLUIDA LA PROPIA SOLICITUD DE LA ACTIVIDAD.
- **INTEGRE TODO SU TRABAJO EN UN SOLO ARCHIVO DE TIPO .PDF, Y ASIGNE EL NOMBRE QUE A CONTINUACIÓN SE INDICA.**

NO OLVIDE ANEXAR LAS HOJAS DE ESTA ACTIVIDAD Y DE SU TRABAJO DESPUÉS DE SU PORTADA.

- UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA NOMENCLATURA QUE SE INDICA MÁS ADELANTE EN ESTE DOCUMENTO.
- SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- **POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA Y ASÍ EVITAR PROBLEMAS DE NULIDAD POR EXTEMPORANEIDAD.**





**LA NOMENCLATURA SOLICITADA PARA ENVIAR SU TRABAJO ES LA SIGUIENTE :**

AAAA-MM-DD\_TNM\_CELAYA\_MATERIA\_DOCUMENTO\_[EQUIPO]\_NOCTROL\_APELLIDOS\_NOMBRE\_SEM.PDF

**( NOTA : \*\*\* TODO DEBE SER ESCRITO USANDO LETRAS MAYÚSCULAS \*\*\* )**

**DONDE :**

TNM_CELAYA	: INSTITUCIÓN ACADÉMICA
AAAA	: AÑO
MM	: MES
DD	: DÍA
MATERIA	: <b>LAI</b> , LI MÁS EL GRUPO ( -A , -B, -C )
DOCUMENTO	: A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUTIVO POR EL QUE CORRESPONDA)
[EQUIPO]	: NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR. [ OPCIONAL ]
NOCTROL	: SU NÚMERO DE CONTROL
APELLIDOS	: SUS APELLIDOS
NOMBRE	: SU NOMBRE
SEM	: EL PERIODO SEMESTRAL EN CURSO: <b>AGO-DIC</b>

**EJEMPLO :**

SI EL TRABAJO SE SOLICITÓ EN EQUIPO.

2023-09-18\_TNM\_CELAYA\_LAI-A\_A4\_EQUIPO\_99\_9999999\_PEREZ\_PEREZ\_JUAN\_AGO-DIC23.PDF

DONDE EL NOMBRE DEBERÁ CORRESPONDER AL JEFE DE EQUIPO QUE HACE LA ENTREGA DEL TRABAJO.

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2023-09-18\_TNM\_CELAYA\_LAI-A\_A4\_9999999\_PEREZ\_PEREZ\_JUAN\_AGO-DIC23.PDF





**FECHA Y HORA DE ENTREGA:**

LA INDICADA EN LA PLATAFORMA VIRTUAL.

EN CASO DE QUE EL TRABAJO SE HAYA SOLICITADO EN EQUIPO, EL JEFE DEL MISMO SERÁ EL ÚNICO RESPONSABLE DE ENVIAR LA ACTIVIDAD EN LA PLATAFORMA VIRTUAL.

**MUY IMPORTANTE:**

1. DESPUÉS DE LA HORA INDICADA EN LA PLATAFORMA VIRTUAL ( AÚN CUANDO SOLO SEA UN MINUTO O VARIOS ), LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.

SE LE SUGIERE ENVIAR CON ANTICIPACIÓN SU ACTIVIDAD A FIN DE EVITAR CONFLICTOS POR NO ENTREGAR ÉSTA A TIEMPO.

BAJO NINGÚN PRETEXTO O JUSTIFICACIÓN SE ACEPTARÁN LOS TRABAJOS EXTEMPORÁNEOS, EVITE LA PENA DE RECORDAR A USTED QUE EL VALOR DE LA PUNTUALIDAD ES PARTE IMPORTANTE DE SUS EVIDENCIAS Y ES EL PRIMER PUNTO QUE SE HA DE EVALUAR.

2. NO OLVIDE ANEXAR A SU ARCHIVO .PDF DE EVIDENCIAS UNA PORTADA PROFESIONAL, Y ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.
3. POR ÚLTIMO, TODA EVIDENCIA GENERADA QUE CONTENGA AL MENOS UNA TRANSCRIPCIÓN DE CUALQUIER FUENTE Y DE CUALQUIER TIPO, ES DECIR CON MATERIAL PLAGIADO SERÁ ANULADA DE FORMA INCONTROVERTIBLE.



Av. Antonio García Cubas #600 esq. Av. Tecnológico, Colonia Alfredo V. Bonfil, C.P. 38010  
Celaya, Gto. Tel. 01 (461) 611 75 75 e-mail: lince@celaya.tecnm.mx tecnm.mx | celaya.tecnm.mx



# Tecnológico Nacional de México en Celaya

Asignatura: Lenguajes y Autómatas II

Docente: Ricardo González González

## Actividad 4

Equipo 4

Integrantes:

Cardoso Morales Cristian David  
González Rodríguez Francisco  
Núñez Servín Juan Ángel

INGENIERÍA EN SISTEMAS  
COMPUTACIONALES

27/09/2023

# Tecnológico Nacional de México en Celaya

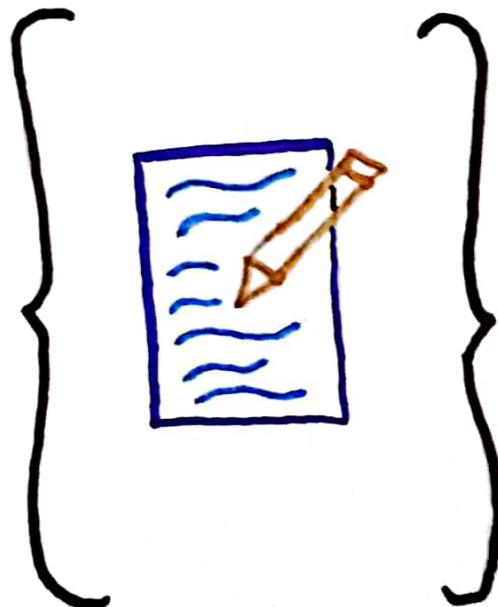
Asignatura : Lenguajes y Autómatas II

Docente : Ricardo González González

"Generadores de analizadores  
léxicos y aplicaciones (caso  
de estudio)"

Integrantes :

Cordoso Morales Cristian David  
González Rodríguez Francisco  
Núñez Servin Juan Ángel



27/09/2023

## TABLA DE CONTENIDO

Indicé de cuadros.....	2
Introducción .....	3
Definición y propósito .....	4
Funcionamiento .....	5
Especificación Formal .....	5
Generación de código .....	6
Integración en el compilador .....	6
Tokens, patrones y lexemas .....	7
Tipos de generadores de analizadores léxicos .....	9
LEX .....	9
FLEX .....	10
ANTLR .....	11
JFLEX .....	12
Sección de reglas .....	13
Comparación de generadores en términos de características y usos .....	14
Estructura de un analizador léxico generado .....	15
Concordancia de patrones .....	16
Aplicaciones y significado .....	17
Aplicaciones ( caso de estudio ) .....	18
DSL .....	18
Concepto y utilidad .....	18
Desarrollo de un DSL para matemáticas simbólicas .....	19
Rol de los analizadores léxicos en el DSL .....	20
Definición de la gramática .....	20
Implementación del analizador léxico .....	20
Escaneo del código fuente .....	21
Validación y generación de árboles sintácticos .....	21
Aplicaciones prácticas del DSL .....	22
Resolución de ecuaciones simbólicas .....	22
Investigación matemática .....	22
Educación .....	22
Desafíos y evolución .....	23
Conclusión .....	23

## INDICE DE CUADROS

Figura 1. Proceso general .....	4
Figura 2. Reconocimiento tokens .....	5
Tabla 1. Ejemplos de tokens .....	8
Figura 3. Proceso analizadores .....	9
Figura 4. Proceso de LEX .....	10
Figura 5. Proceso de FLEX .....	11
Figura 6. Proceso de ANTLR .....	12
Figura 7. Proceso de JFLEX .....	13
Tabla 2. Comparación de generadores .....	15
Figura 8. Estructura general .....	16
Figura 9. Ejemplo de un AFND .....	17
Figura 10. Introducción DSL .....	20
Figura 11. Comportamiento DSL .....	21
Figura 12. Proceso de aplicación DSL .....	22

# Monografía Generadores de analizadores léxicos

## Introducción

La creación de compiladores y analizadores léxicos es una disciplina fundamental en el campo de la informática y la programación. Estos analizadores léxicos, también conocidos como escáneres, desempeñan un papel esencial en el proceso de traducción de código fuente escrito por humanos a un formato que las máquinas pueden comprender. Los analizadores léxicos se encargan de descomponer el código en unidades léxicas o tokens, que son elementos básicos de programación, como palabras clave, identificadores, números y símbolos. Uno de los componentes que son críticos para desarrollar eficazmente estos analizadores léxicos es el generador de analizadores léxicos.

## Definición y Propósito

Un generador de analizadores léxicos es una herramienta de software que automática la creación de analizadores léxicos a partir de una especificación formal. Su principal objetivo es simplificar el proceso de desarrollo de un analizador léxico, permitiendo a los diseñadores y desarrolladores concentrarse en la lógica del lenguaje de programación en lugar de preocuparse por todos los detalles minuciosos del análisis léxico.

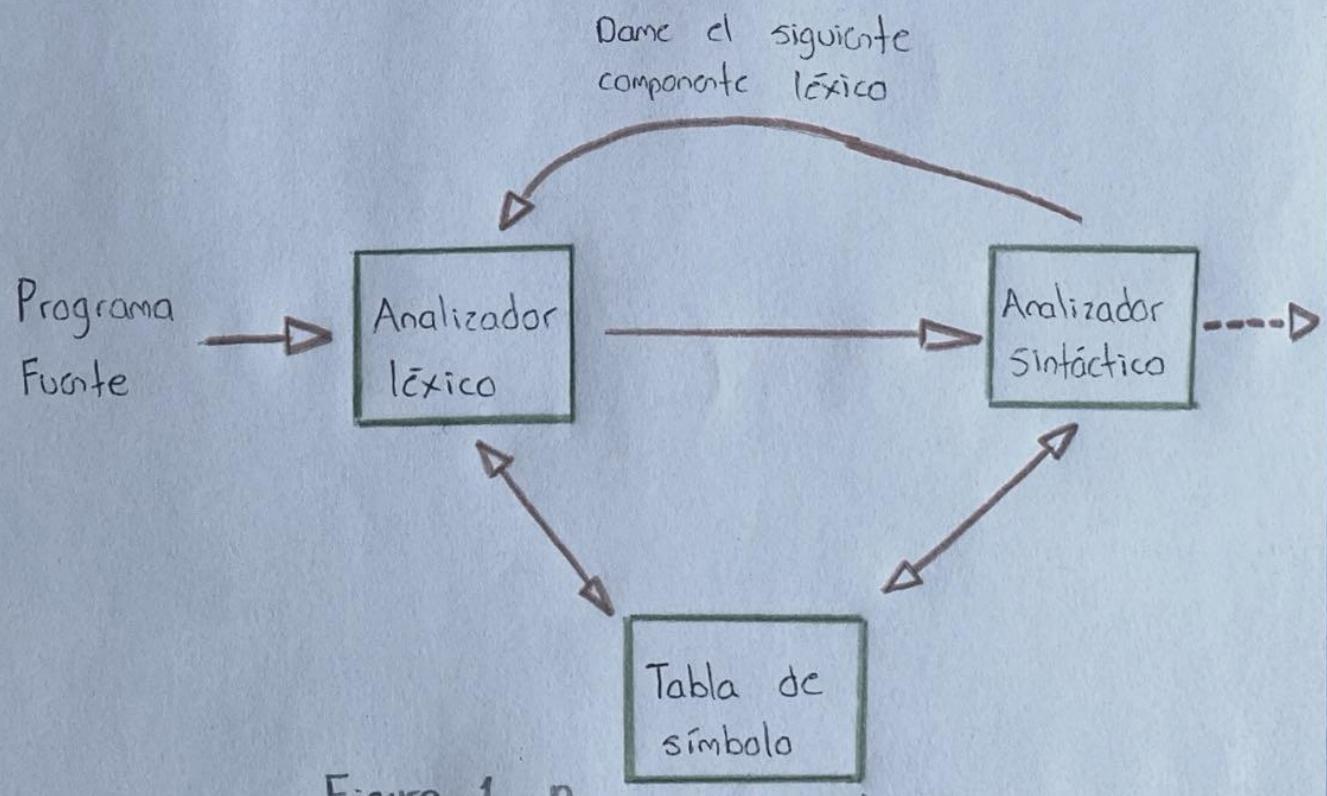


Figura 1. Proceso general

## Funcionamiento

### Especificación Formal

El proceso comienza con la definición formal del lenguaje de programación o el formato que se desea analizar. Esta de las especificaciones se realiza a menudo utilizando herramientas como los gramáticas regulares o gramáticas libres de contexto, que describen las reglas y patrones que determinan cómo se deben reconocer los tokens en el código fuente. Estas gramáticas proporcionan una descripción formal de las estructuras de lenguaje y la sintaxis, lo que facilita enormemente la comprensión de cómo se deben identificar los tokens.

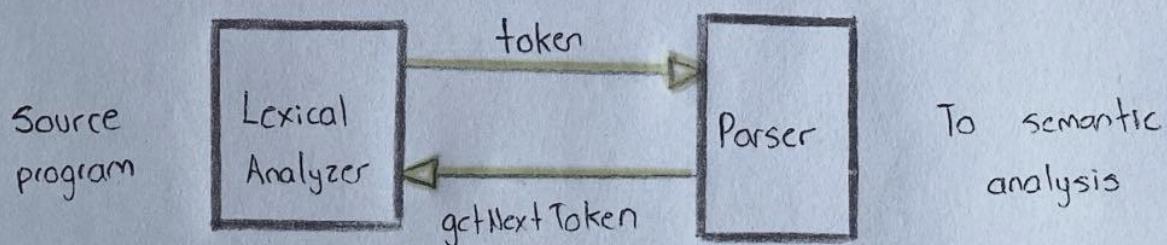


Figura 2. Reconocimiento tokens

## Generación de código

Una vez que se ha definido la especificación formal, el generador de analizadores léxicos toma toda esta información como entrada y genera automáticamente el código fuente del analizador léxico en el lenguaje de programación deseado. Esto ahorra tiempo y reduce significativamente el riesgo de errores humanos en la altamente optimizado y eficiente que puede manejar grandes volúmenes de código fuente de manera rápida y precisa.

## Integración en el compilador

Una vez generado el código del analizador léxico, se integra en el compilador junto con el analizador sintáctico y otros componentes necesarios para el proceso de la compilación. El analizador léxico es la primera etapa del proceso de análisis, y su función principal es escanear el código fuente de entrada y dividirlo en tokens. Cada token se clasifica según las reglas definidas en la especificación formal, y luego se pasa a todo el analizador sintáctico para su procesamiento posterior.

## Tokens, patrones y lexemas

Un "token" consiste en un nombre de token y, en algunas ocasiones, un valor de atributo opcional. El nombre de token representa un tipo particular de unidad léxica en el código fuente, como una palabra clave específica o un identificador.

Por otro lado, un "patrón" se refiere a la descripción de la apariencia que pueden tener los lexemas de un token. Para tokens simples, como palabras clave, el patrón se limita a la secuencia de caracteres que compone dicha palabra clave. Sin embargo, para tokens más complejos, como los identificadores, los patrones pueden ser estructuras más elaboradas que se relacionan con múltiples cadenas de caracteres.

Un "lexema" representa una serie de caracteres encontrados en el código fuente del programa que encaja con el patrón asociado a un token específico. Es el analizador léxico el encargado de reconocer estos lexemas como instancias de los tokens correspondientes.

A continuación, se presenta una tabla que muestra ejemplos de tokens comunes junto con sus patrones, descritos en términos informales, así como ejemplos concretos de lexemas.

Token	Descripción Formal	Lexemas de Ejemplo
if	caracteres i,f.	if
else	caracteres e,l,s,e	Else
comparación	$<0> \circ <=0> \circ$ $= = \circ ! =$	$\leq, !=$
id	letra seguida por letras y dígitos	p1, puntuación, D2
número	cualquier constante numérica	3.14159, 0, 6.02e23
literal	cualquier cosa, excepto "", rodeada por '''s	"core dumped"

Tabla 1. Ejemplos de tokens

Ejemplos

Token - While, Suma, Identificador

Lexema - While, +, a - valor - b

Patrón - While, +, [a-z-A-Z] +

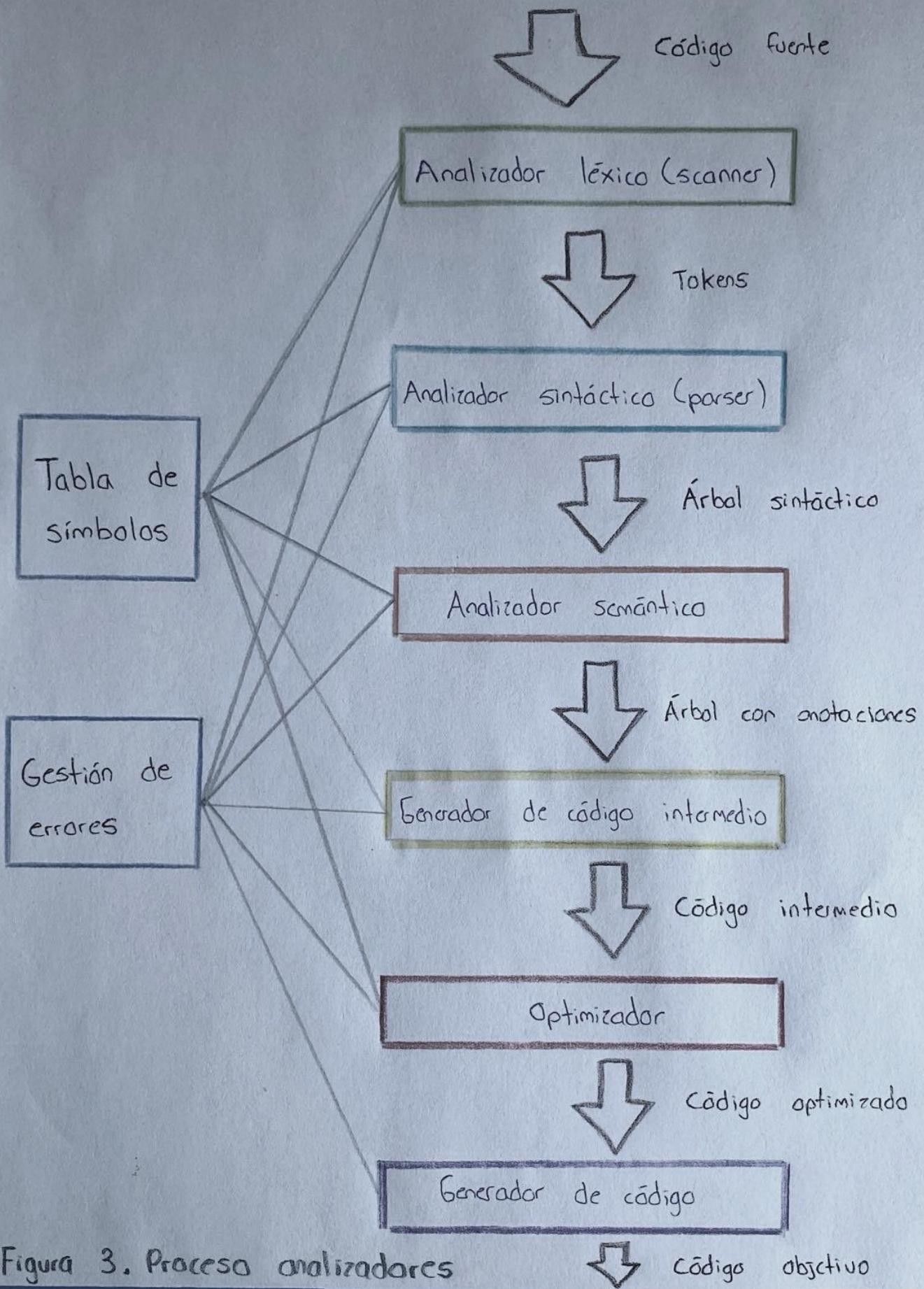


Figura 3. Proceso analizadores

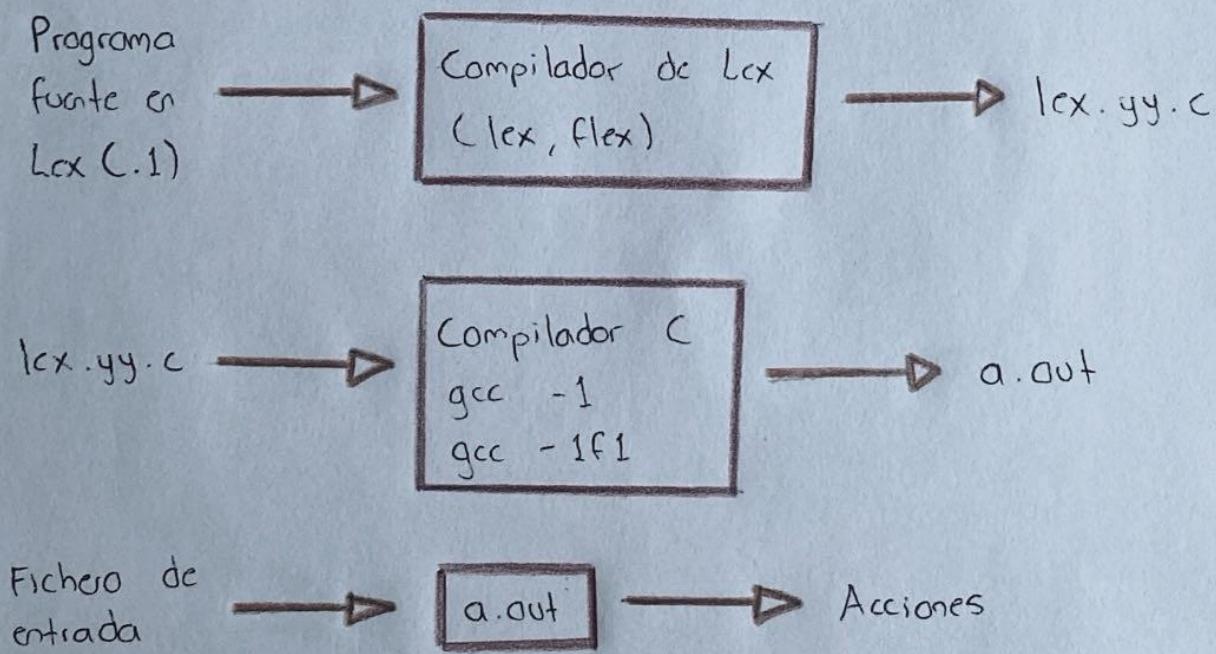
Código objetivo

## Tipos de generadores de analizadores léxicos

### Lex

Lex es uno de los generadores de analizadores léxicos más antiguos y ampliamente utilizados. Fue desarrollado en los laboratorios Bell en la década de 1970 y genera código en C. Lex permite a los desarrolladores especificar las reglas de análisis léxico utilizando expresiones regulares. Es ampliamente utilizado en sistemas Unix y ha servido como base para muchos otros generadores de analizadores léxicos.

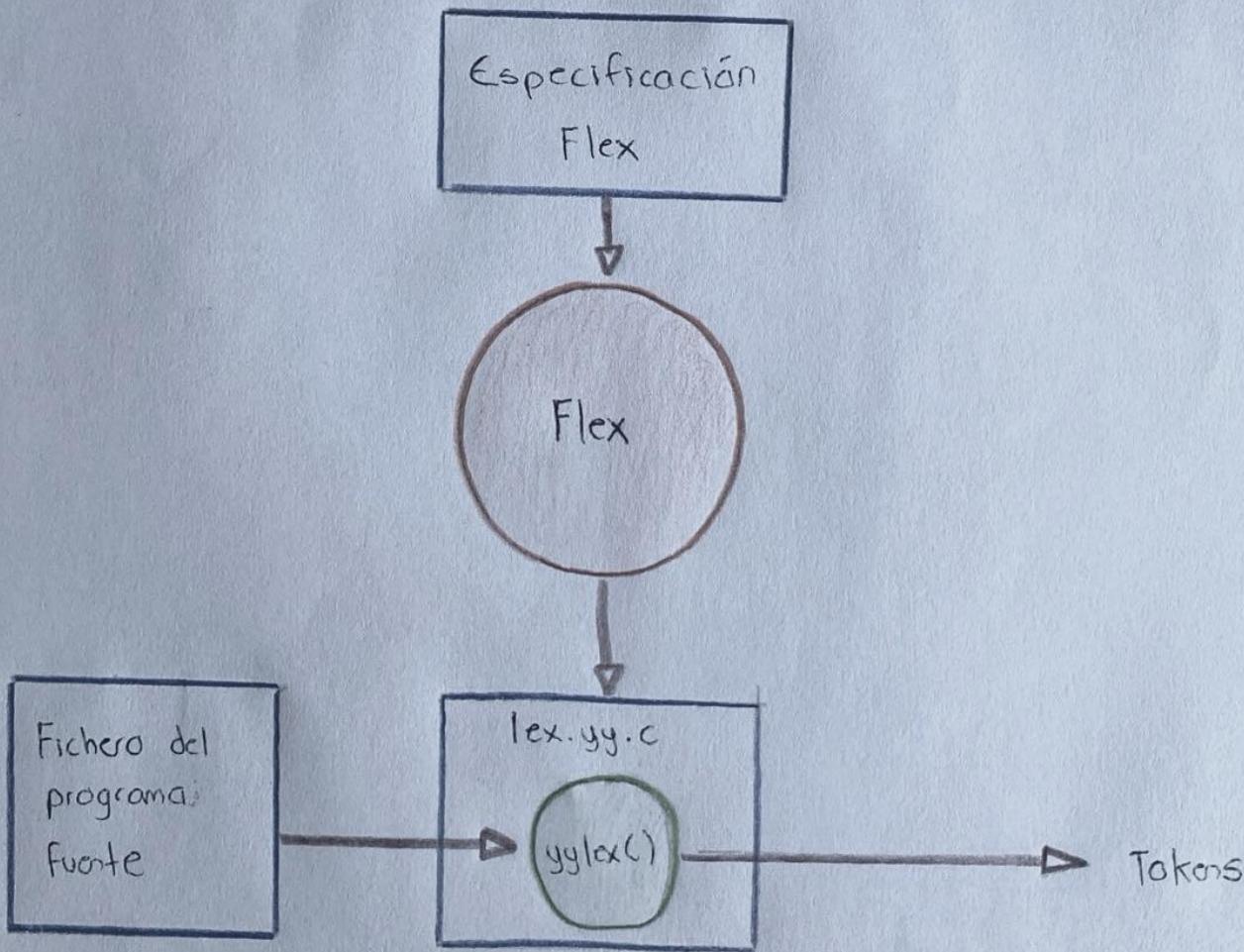
Figura 4. Proceso de Lex



# Flex

Flex es una versión mejorada y moderna de Lex y es una herramienta de código abierto. Aunque sigue siendo compatible con las especificaciones de Lex, ofrece una mayor flexibilidad y funcionalidad. Flex es especialmente popular en el desarrollo de sistemas Unix y proyectos de código abierto debido a su licencia de código abierto.

Figura 5. Proceso de Flex



## ANTLR

ANother Tool for Language Recognition es una herramienta de generación de analizadores léxicos y sintácticos extremadamente versátil. A diferencia de Lex y Flex, ANTLR permite generar analizadores léxicos y sintácticos en varios lenguajes de programación como Java, C#, Python y más. Además, ANTLR se utiliza más comúnmente en la creación de lenguajes de dominio específico (DSL) y sistemas de procesamiento de lenguaje natural.

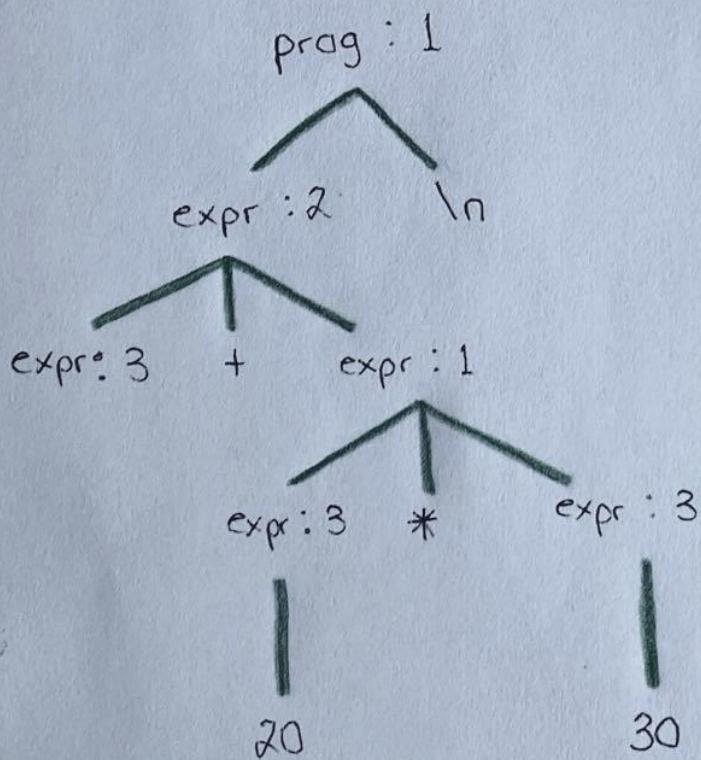


Figura 6. Proceso de ANTLR

## JFlex

Es una herramienta similar a Flex, pero está diseñada específicamente para generar analizadores léxicos en Java. Esto es especialmente útil para proyectos que utilizan Java como lenguaje principal de desarrollo. JFlex ofrece una integración perfecta con el ecosistema de Java y proporciona un alto rendimiento en aplicaciones Java.

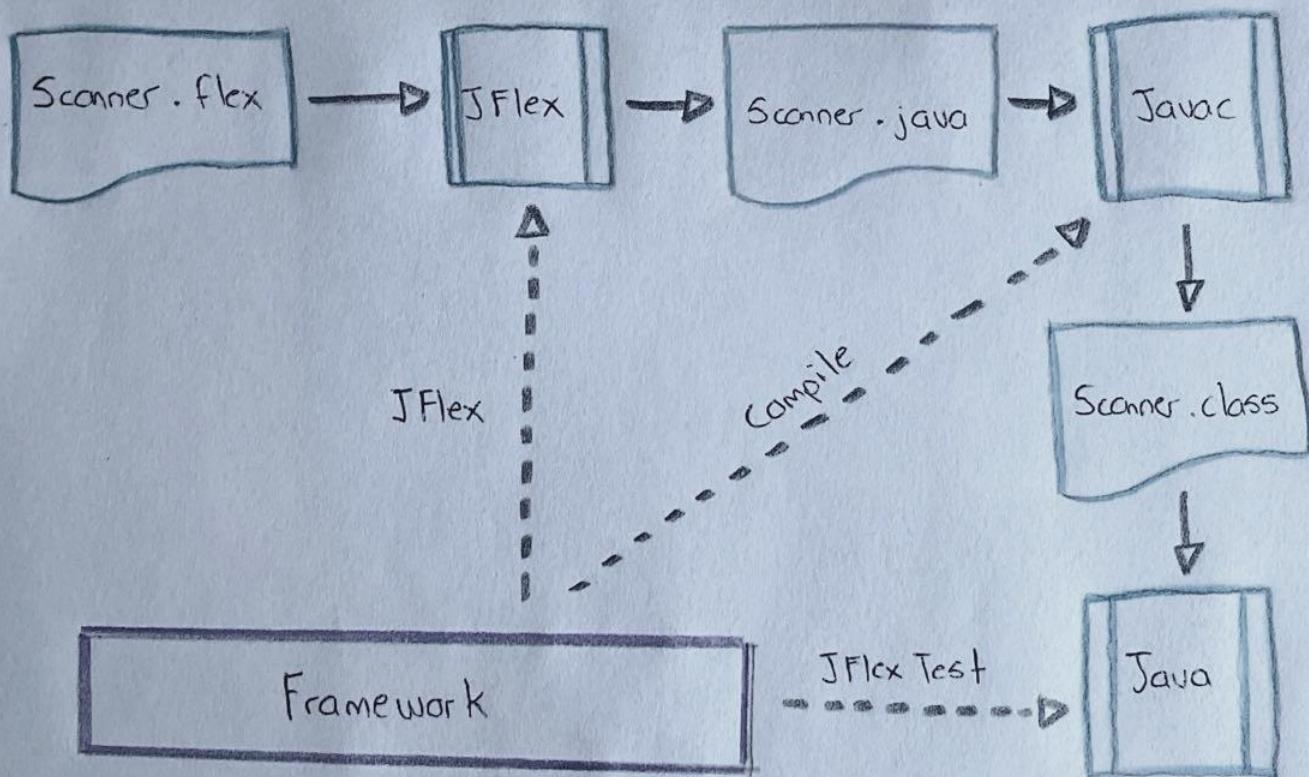


Figura 7. Proceso de JFlex

## Sección de reglas

En un programa Lex, una regla tiene una estructura que se asemeja a una sentencia patrón - acción de Awk. Esta estructura básica se compone de dos partes:

\* patrón - regular

\* acción - en - C

El patrón regular, que debe ubicarse al principio de la línea sin espacios antes del patrón. Lex admite las expresiones regulares en formato similar al de "egrep" con algunas extensiones adicionales detalladas en su documentación.

La acción en lenguaje C, que puede ser una única sentencia o un bloque de sentencias encerradas entre llaves {}.

Cuando hay varios patrones que coinciden con la entrada, Lex seleccionará la cadena más larga posible que se ajuste a un patrón. Si aún existen múltiples patrones que coinciden, Lex ejecutará la acción de la regla que aparece primero en el código fuente de Lex.

Comparación de generadores  
en términos de características  
y usos

Tabla 2. Comparación de generadores

Característica	Flex	ANTLR	Lex
Tipo de herramienta	Generador de analizador	Generador de analizador	Generador de analizador
Lenguaje de programación soportado	C/C++	Java	C
Gramática soportada	Expresiones regulares	Gramáticas de contexto libre	Expresiones regulares
Tipo de analizador	Analizador Léxico	Analizador Léxico y sintáctico	Analizador Léxico
Nivel de abstracción	Bajo	Alto	Bajo
Soporte para gramáticas	Limitado	Amplio	Limitado
Características avanzadas	Limitadas	Si (Herencia, Árboles de sintaxis)	Limitadas

## Estructura de un analizador léxico generado

Un analizador léxico que fue generado por Lex tiene la siguiente estructura:

El programa que sirve como analizador léxico incluye un programa fijo que simula un autómata, en este punto dejamos abierto si ese autómata es determinista o no determinista. El resto del analizador léxico consta de componentes creados por el propio Lex a partir del programa Lex.

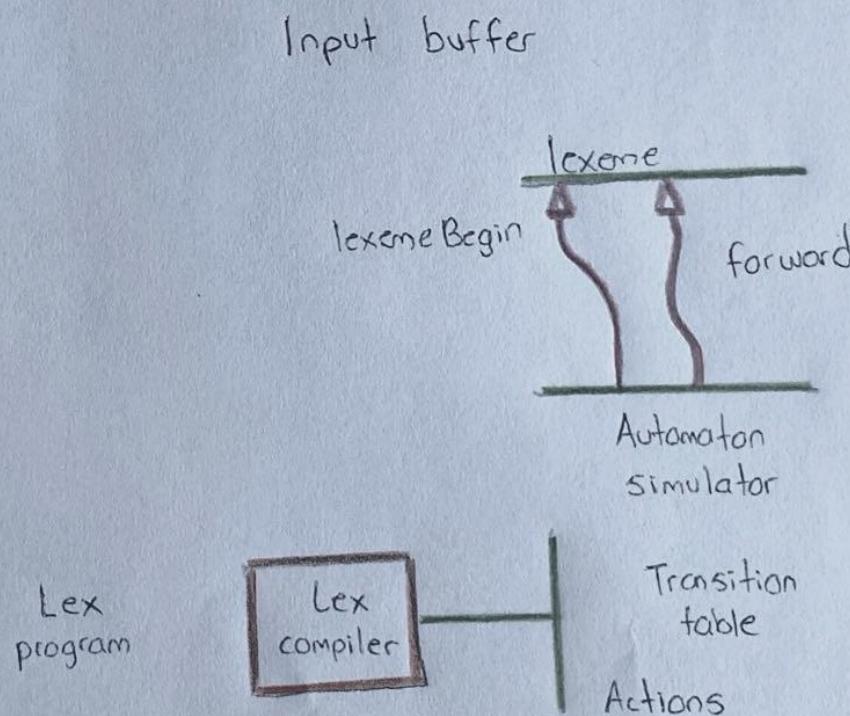


Figura 8. Estructura general

Para construir el autómata, comenzamos tomando cada patrón de expresión regular en el programa Lex y convierte en un AFND.

## Concordancia de patrones basados en AFND

Si el analizador léxico simula un NFA, entonces debe leer la entrada comenzando en el punto de su entrada al que nos hemos referido como lexemaBegin. A medida que mueve el puntero llamado hacia adelante en la entrada, calcula el conjunto de estados en el que se encuentra cada punto.

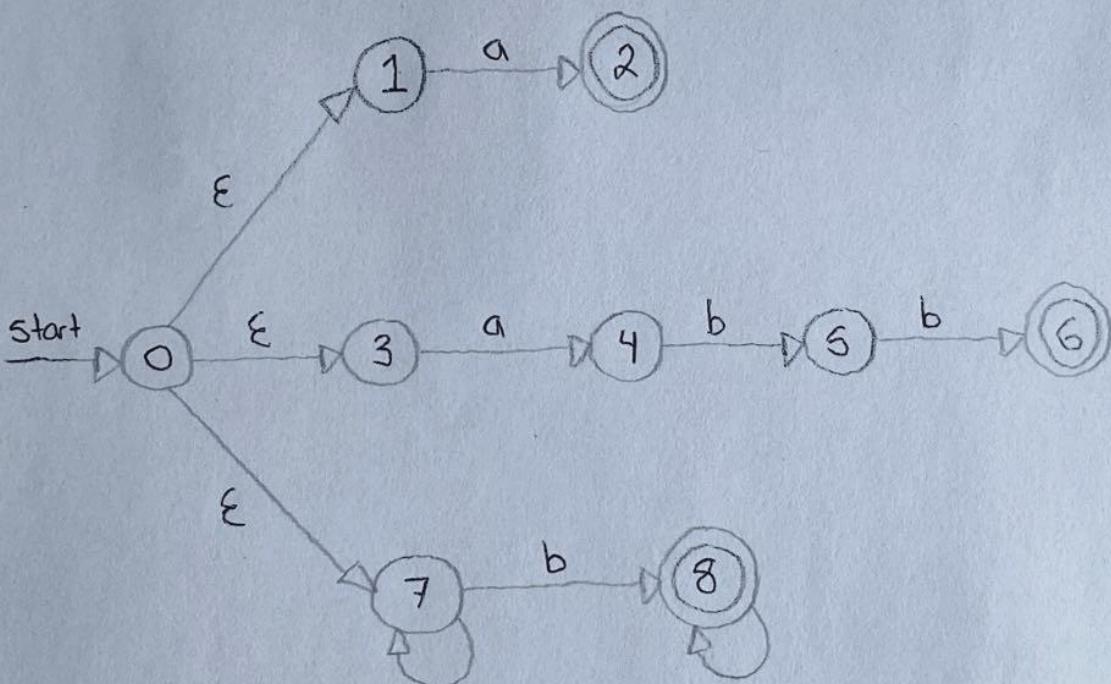


Figura 9. Ejemplo de un AFND

La simulación NFA llega a un punto en la entrada donde no hay estados siguientes. Por lo tanto, estamos listos para decidir cuál es el prefijo más largo que es un lexema que coincide con algún patrón.

## Aplicaciones y Significado

Los generadores de analizadores léxicos tienen una amplia gama de aplicaciones en el desarrollo de software. Se utilizan no solo en la creación de compiladores para lenguajes de programación, sino que también en sistemas de procesamiento de lenguaje natural, procesadores de texto avanzado, motores de búsqueda y cualquier aplicación que requiera la identificación y clasificación eficiente de patrones en datos de entrada.

Un área en la que los generadores de analizadores léxicos han tenido un impacto significativo es en la creación de lenguajes de programación personalizados y lenguajes específicos del dominio (DSL). Estos tipos de lenguajes personalizados o monudo se utilizan para resolver problemas específicos en campos como la ingeniería, la matemática, la bioinformática y otros, y los generadores de analizadores léxicos facilitan enormemente su implementación.

## Aplicaciones (caso de estudio)

DSL

Los analizadores léxicos son una parte esencial del desarrollo de software y desempeñan un papel muy crítico en la creación de lenguajes de programación específicos del dominio (DSL). Las aplicaciones de los analizadores léxicos se pueden hacer a través de un caso de estudio detallado centrado en el desarrollo de un DSL para matemáticas simbólicas. Este caso de estudio ilustrará como los analizadores léxicos facilitan la creación de lenguajes de la programación personalizados y su aplicación en varios campos.

DSL : Concepto y Utilidad

Un lenguaje de programación específico del dominio (DSL) es un lenguaje diseñado para resolver problemas específicos en un dominio particular. A diferencia de los lenguajes de programación generales como

Java, C++ o Python, que son versátiles pero a menudo requieren mucho código para resolver problemas específicos, un DSL se adapta estrechamente a un dominio en particular y permite a los desarrolladores expresar soluciones de manera más concisa y más comprensible. Los DSL son herramientas valiosas para simplificar la programación en dominios especializados.

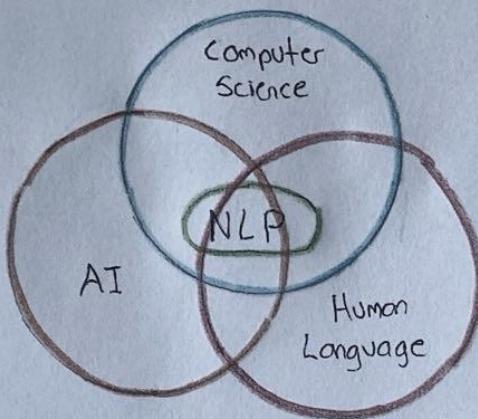


Figura 10. Interacción DSL

Caso de estudio: Desarrollo de un DSL para matemáticas simbólicas

En nuestro caso de estudio, consideraremos el desarrollo de un DSL para matemáticas simbólicas. Este DSL permitirá a los usuarios realizar cálculos matemáticos con símbolos algebraicos en lugar de valores numéricos. Ejemplos de aplicaciones incluyen resolución de ecuaciones, simplificación y manipulaciones simbólicas avanzadas.

## Rol de los analizadores léxicos en el DSL

### Definición de la gramática

La creación de un DSL comienza con la definición de su gramática, que describe las reglas sintácticas y léxicas que rigen la estructura del lenguaje. En nuestro caso de estudio, definiremos reglas para los operadores matemáticos, variables, constantes, funciones y más.

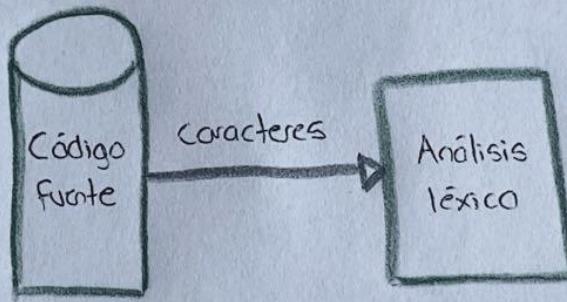


Figura 11. Comportamiento DSL

### Implementación del analizador léxico

Una vez definida la gramática, es necesario implementar el analizador léxico. Aquí es donde los generadores de analizadores léxicos, como Flex o ANTLR, desempeñan un papel crucial. Utilizando estas herramientas, generamos automáticamente el código del analizador léxico basado en las reglas definidas previamente.

## Escaneo del código fuente

Cuando un usuario escribe una expresión matemática en nuestro DSL, el analizador léxico escanea el código fuente y divide la expresión en tokens. Por ejemplo, si el usuario escribe "x + y", el analizador léxico podría generar los tokens : "variable (x)", "operador (+)", "variable (y)".

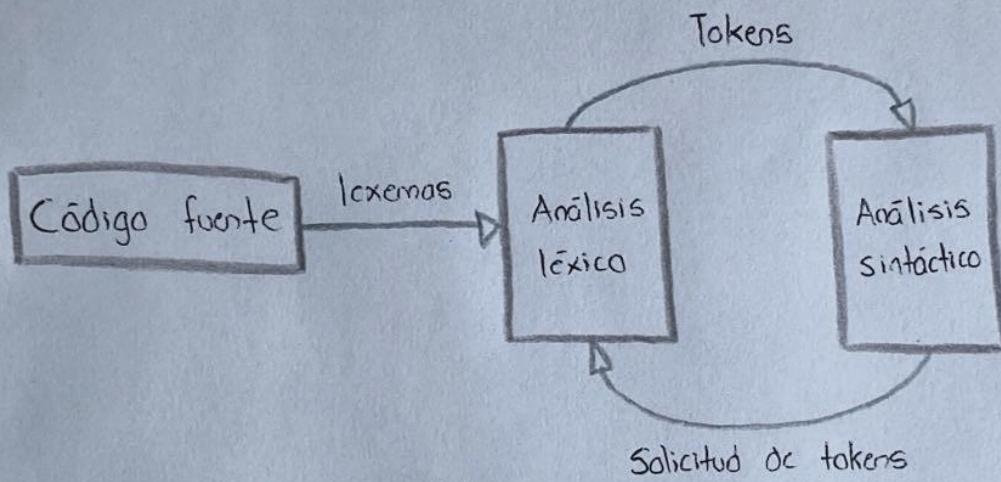


Figura 12. Proceso de aplicación DSL

## Validación y generación de árboles sintácticos

Una vez generados los tokens, el analizador léxico verifica que la expresión cumpla con las reglas gramaticales del DSL. Si es así, pasa los tokens al analizador sintáctico, que crea un árbol sintáctico para representar la estructura de la expresión.

## Aplicaciones prácticas del DSL

### Resolución de ecuaciones simbólicas

Los usuarios pueden utilizar nuestro DSL para resolver ecuaciones simbólicas de manera más natural y concisa. Por ejemplo, podrían escribir una ecuación como " $2x^2 + 3x + 1 = 0$ " y obtener la solución simbólica.

### Investigación matemática

El DSL podría ser utilizado por matemáticos y por los investigadores para explorar y comprender mejor las relaciones matemáticas en sus estudios. Permitiría una notación más natural y simplificada para así expresar fórmulas y ecuaciones complejas.

### Educación

Nuestro DSL podría ser una herramienta valiosa en los entornos educativos para enseñar matemáticas simbólicas a estudiantes. Proporcionaría una forma más accesible y visual de aprender conceptos matemáticos que son más avanzados.

## Desafíos y Evolución

A pesar de su importancia, el desarrollo de analizadores léxicos no está exento de desafíos. La creación de gramáticas precisas y la generación de código eficiente son tareas complejas. Además, la adaptación a nuevos paradigmas de programación, como en el desarrollo web y móvil, plantea desafíos adicionales para los generadores de analizadores léxicos.

La evolución continua de estas herramientas es crucial para mantenerse al día con los avances en el campo de la programación. Los desarrolladores de generadores de analizadores léxicos están constantemente buscando formas de mejorar la eficiencia, la flexibilidad y la capacidad de manejo de nuevos lenguajes y de los paradigmas de programación.

## Conclusión

En conclusión, los generadores de análisis léxico avanzan como un primer paso en el proceso de traducción.

de lenguaje humano a lenguaje de máquina. Al dividir el texto en tokens, permiten a las computadoras identificar palabras clave, operadores y símbolos que son esenciales para la ejecución de programas de manera precisa. Esto es fundamental en la compilación de código fuente y en la interpretación de lenguajes de programación.

Estos generadores de análisis léxicos desempeñan un papel fundamental en la construcción de software y en el procesamiento de lenguaje natural, ya que facilitan la comprensión y la manipulación de texto y código, lo que a su vez mejora la eficiencia y toda la confiabilidad de las aplicaciones informáticas.

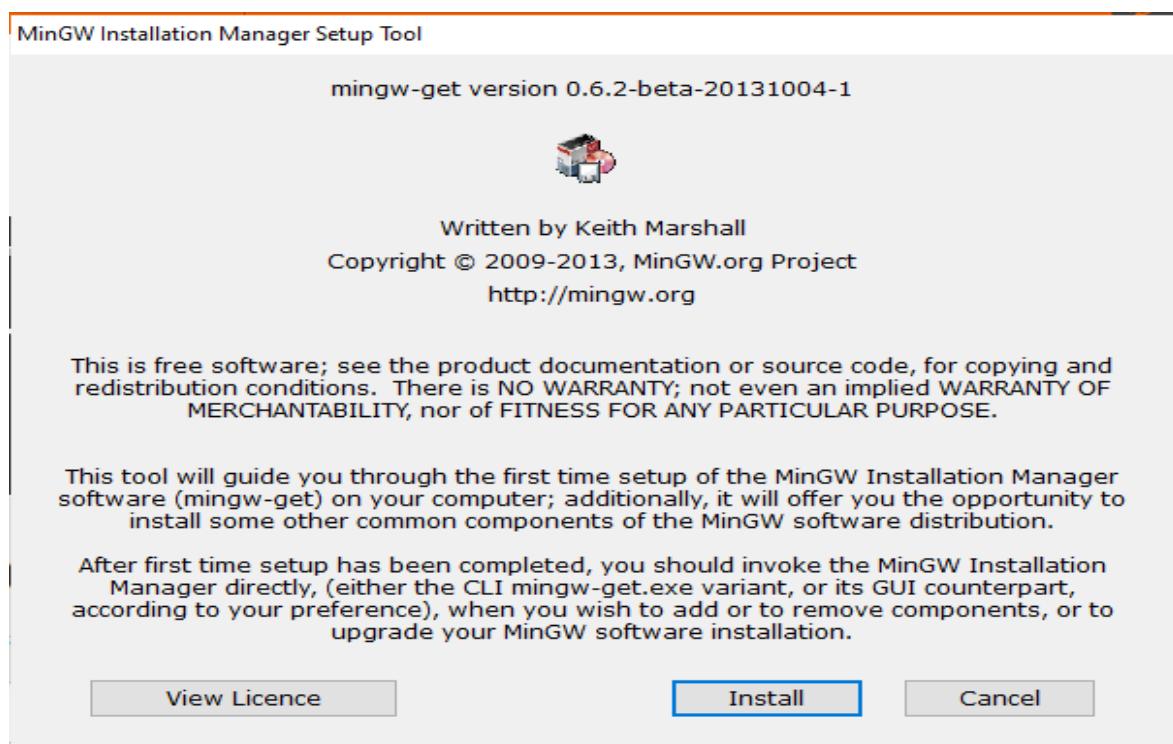
# Instalación Completa

## Instalar MinGW para Windows

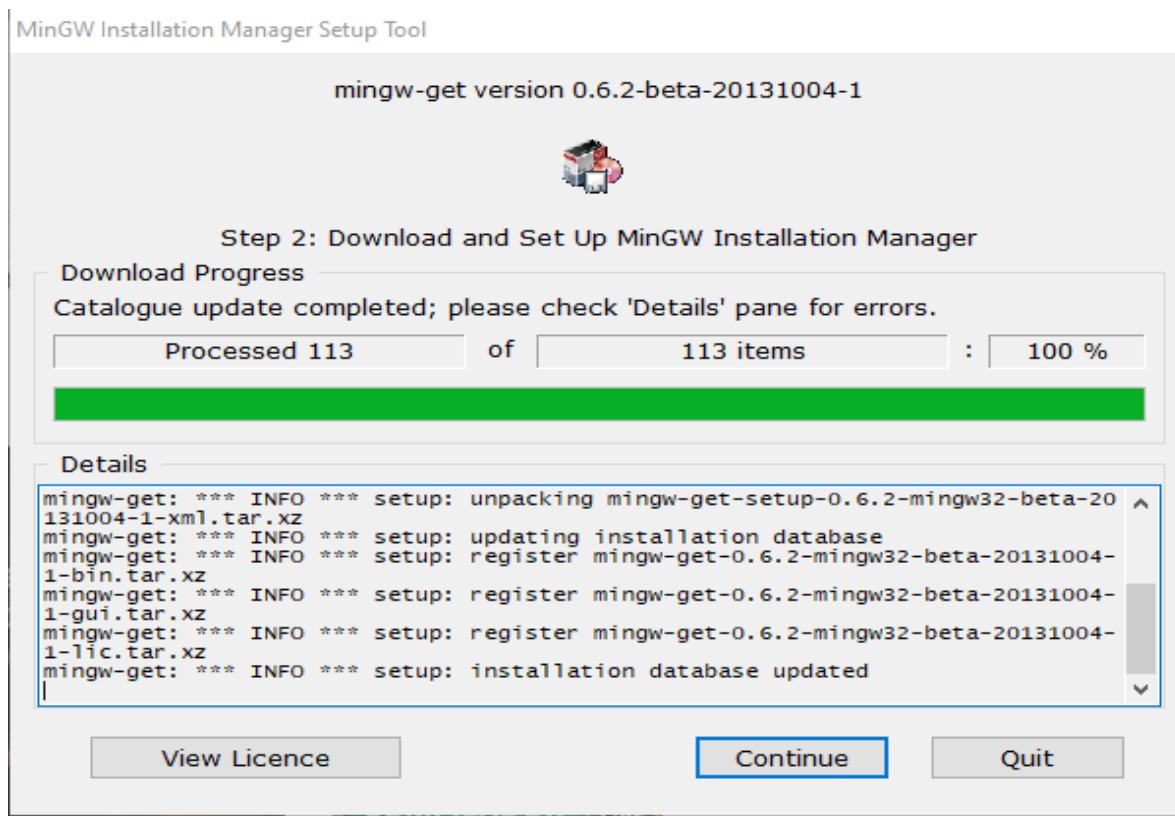
Para poder instalar este primer programa puedes visitar esta liga que se muestra a continuación: <https://sourceforge.net/projects/mingw/>, una vez estando ahí solo daras click en Download.

The screenshot shows the SourceForge website for the MinGW project. At the top, there's a navigation bar with links for 'Open Source Software', 'Business Software', and 'Resources'. Below the navigation is a banner for the TV show 'Yellowstone'. The main content area features the project title 'MinGW - Minimalist GNU for Windows' with a brief description: 'A native Windows port of the GNU Compiler Collection (GCC)' and credits to 'Brought to you by: cstraus, earnie, gressett, keithmarshall'. It shows a rating of 4.5 stars from 176 reviews, weekly downloads of 6,466,423, and a last update date of 2021-09-05. There are buttons for 'Download', 'Get Updates', and 'Share This'. A 'Windows' tag is visible at the bottom left.

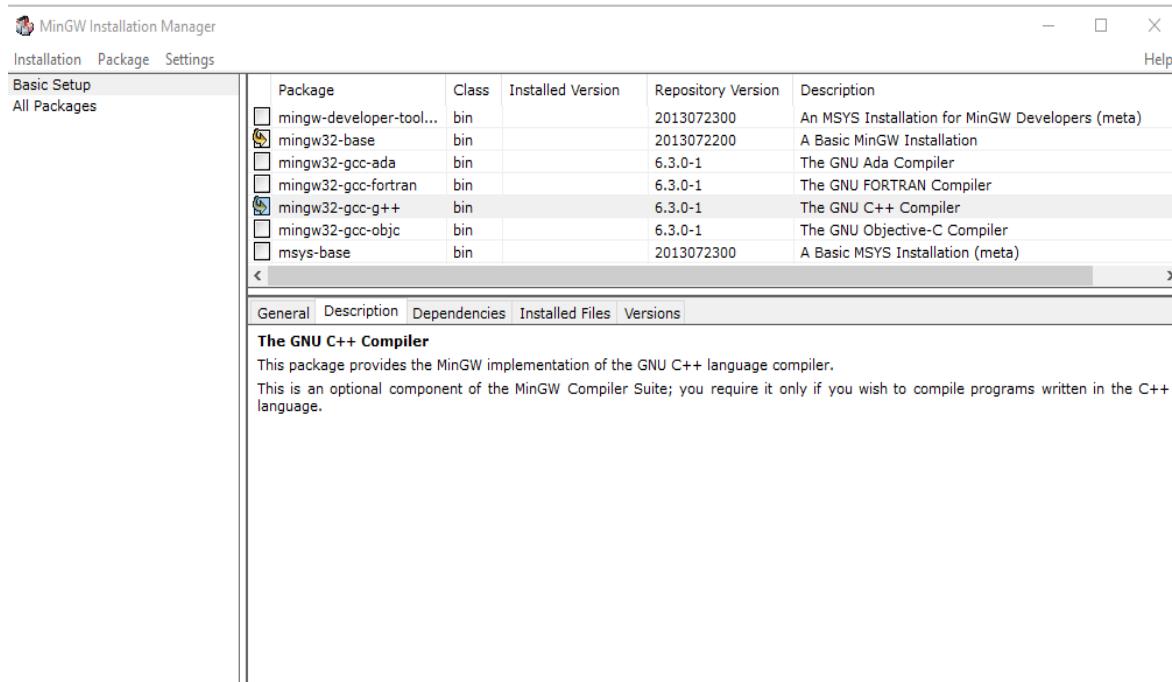
Abriremos el programa y solo daremos click en el botón de Install.



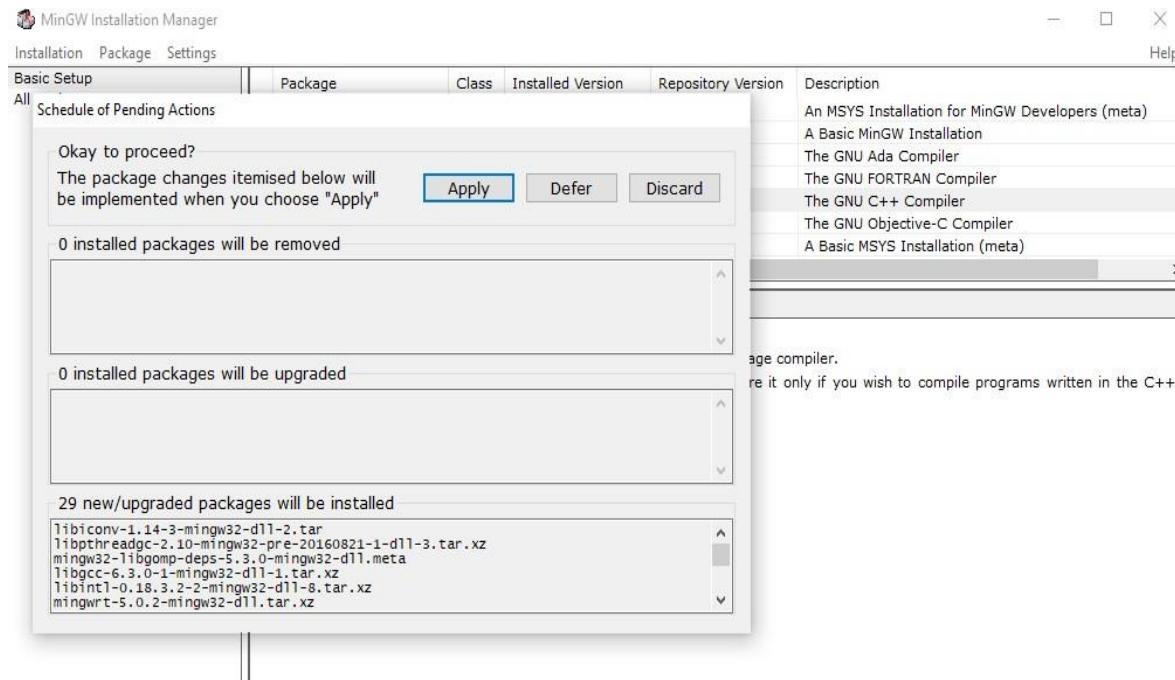
Esperaremos en lo que se termina de hacer la instalación y una vez terminada, daremos click en el botón de Continuar.



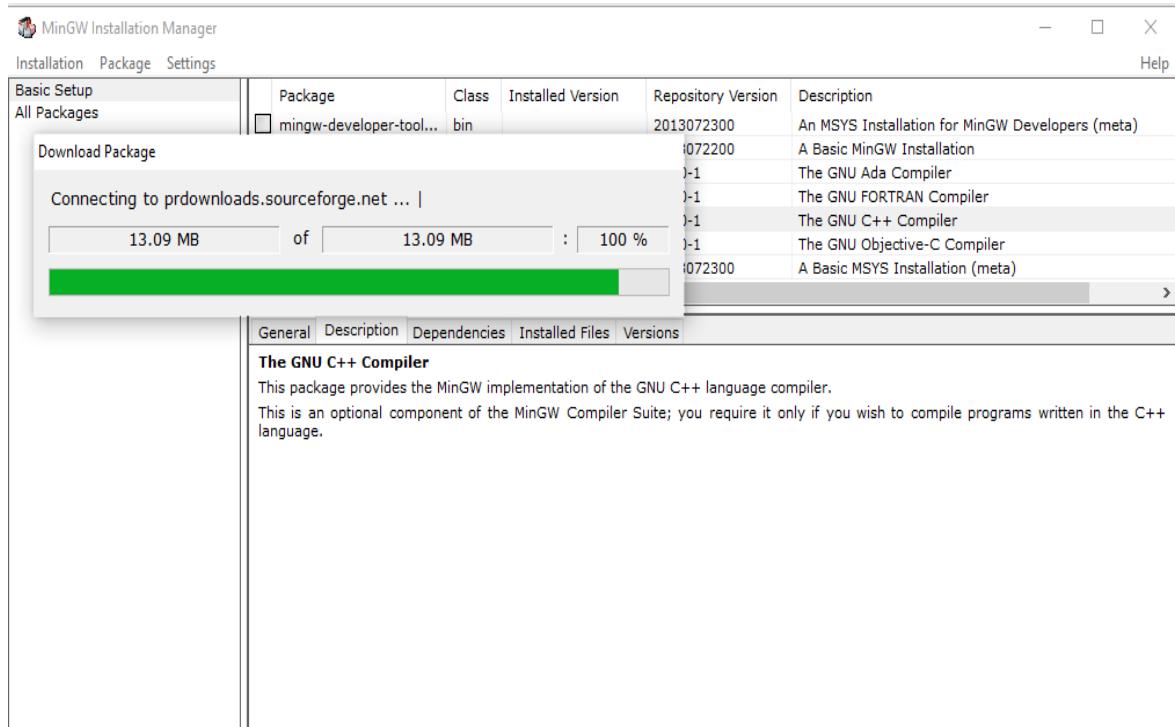
Nos aparecerá la siguiente pantalla y marcaremos la segunda opción y la quinta opción como se muestra en la imagen inferior.



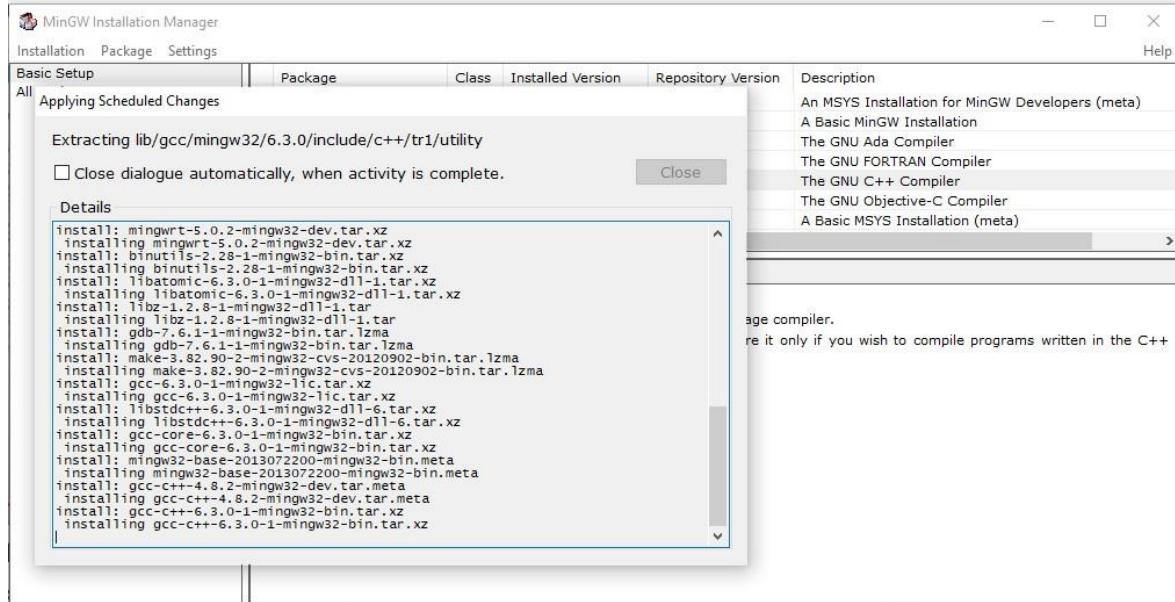
Una vez marcada las opciones, daremos en la opción de Installation y nos aparecerá una ventana, donde solo daremos click en el botón de Apply.



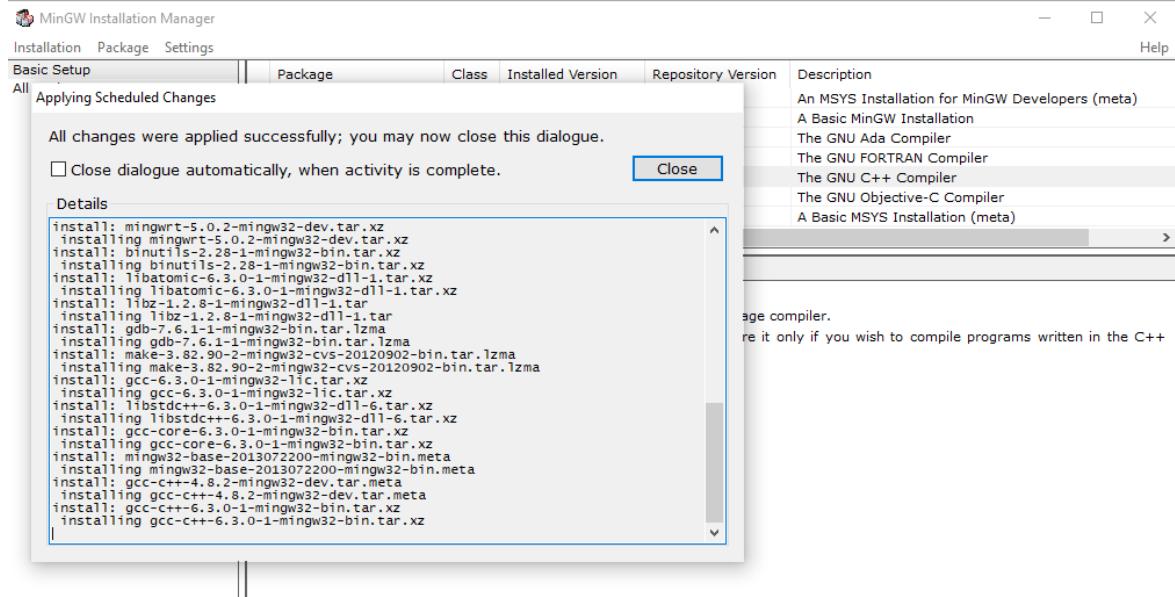
Esperaremos el proceso mientras se instalan todos los paquetes que marcamos en las opciones.



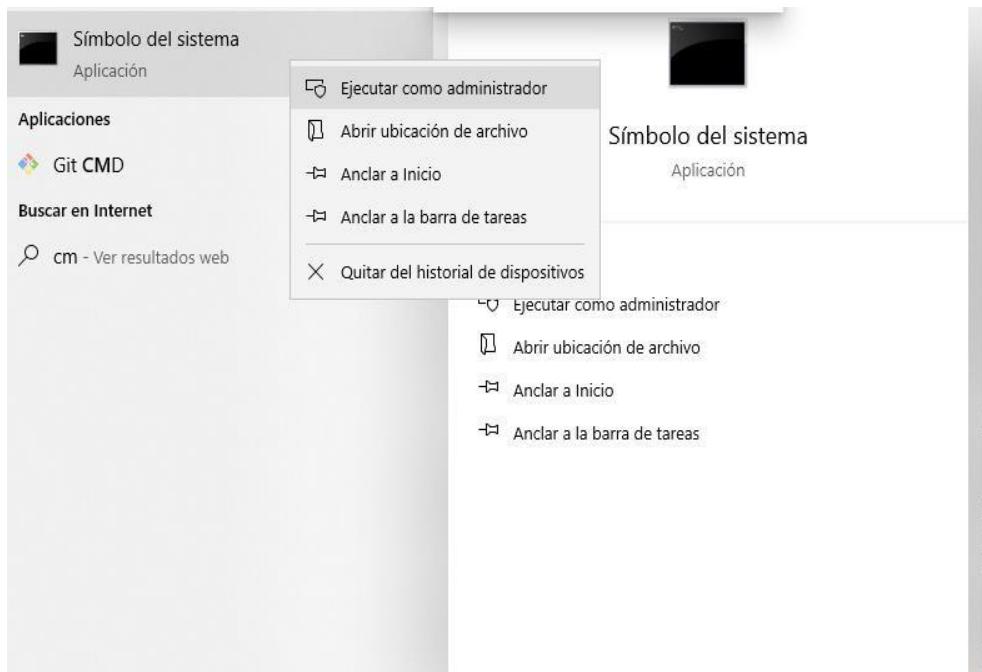
Esperaremos a que ahora se instalen las demás librerías para que podamos correr cualquier programa.



Una vez que se instalen todas las librerías y los paquetes, solo daremos click en el botón de Close y cerraremos todas las ventanas.



Abriremos nuestra terminal, pero para esto la ejecutaremos como administrador.



Una vez estando en la terminal, entraremos a la carpeta de MinGW, luego a carpeta bin y estando en esa carpeta solo ejecutaremos el comando “dir” para saber que todos los directorios se han instalado y aparecen creados correctamente.

```
C:\>cd MinGW
C:\MinGW>cd bin
C:\MinGW\bin>dir
El volumen de la unidad C es OS
El número de serie del volumen es: 1695-848D

Directorio de C:\MinGW\bin

25/09/2023 10:41 p. m.    <DIR>      .
25/09/2023 10:41 p. m.    <DIR>      ..
22/05/2017 04:18 a. m.      966,670 addr2line.exe
22/05/2017 04:18 a. m.      992,782 ar.exe
22/05/2017 04:18 a. m.      1,734,158 as.exe
29/05/2017 02:59 p. m.      997,902 c++.exe
22/05/2017 04:18 a. m.      955,406 c++filt.exe
24/07/2017 11:03 a. m.      996,366 cpp.exe
22/05/2017 04:18 a. m.      1,024,526 dlltool.exe
22/05/2017 04:18 a. m.      163,342 dllwrap.exe
22/05/2017 04:18 a. m.      151,054 elfedit.exe
29/05/2017 02:59 p. m.      997,902 g++.exe
24/07/2017 11:03 a. m.      70,670 gcc-ar.exe
24/07/2017 11:03 a. m.      70,670 gcc-nm.exe
24/07/2017 11:03 a. m.      70,670 gcc-ranlib.exe
24/07/2017 11:03 a. m.      995,342 gcc.exe
24/07/2017 11:03 a. m.      596,494 gcov-tool.exe
```

Para checar que versión tenemos falta con ejecutar el comando de “gcc –version” y con eso será suficiente para conocer la versión.

```
C:\MinGW\bin>gcc --version
gcc (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Otra forma para checar que versión tenemos falta con ejecutar el comando de “g++ –version” y con eso será suficiente para conocer la versión.

```
C:\MinGW\bin>g++ --version
g++ (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

## Descargar Flex para Windows

Para poder instalar este primer programa puedes visitar esta liga que se muestra a continuación: <https://gnuwin32.sourceforge.net/packages/flex.htm>, una vez estando ahí solo daras click en la parte de Download Setup.

The screenshot shows a web browser window with the URL "gnuwin32.sourceforge.net/packages/flex.htm" in the address bar. The page content is as follows:

**Flex for Windows**

Flex: fast lexical analyzer generator

**Version**  
2.5.4a

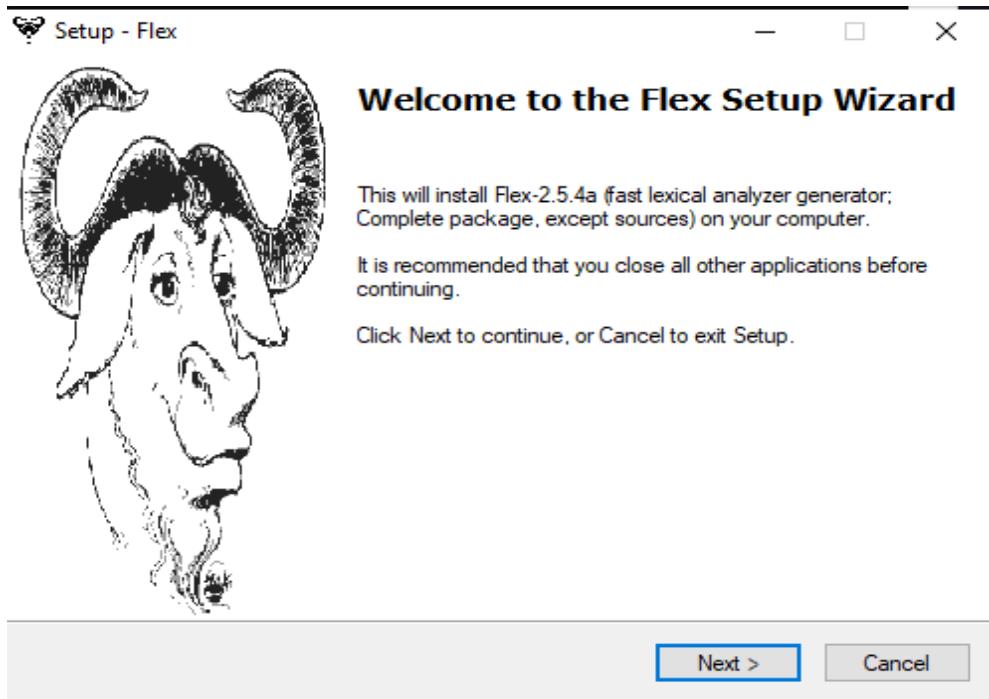
**Description**  
Flex is a fast lexical analyser generator. It is a tool for generating programs that perform pattern-matching on text. There are many applications for Flex, including writing compilers in conjunction with GNU Bison. Flex is a free implementation of the well known Lex program. It features a Lex compatibility mode, and also provides several new features such as exclusive start conditions.

**Homepage**  
<http://www.gnu.org/software/flex/flex.html>

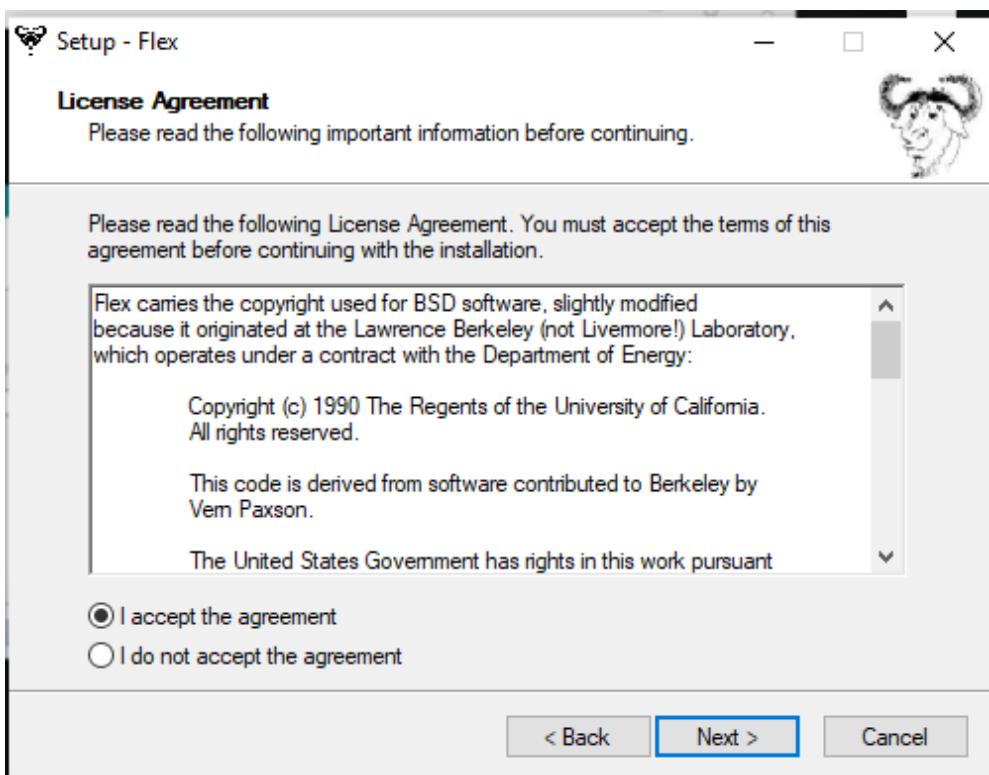
**Download**  
If you download the Setup program of the package, the dependencies, as listed below under Requirements, are already included, apart from msrv.dll. If you download the package as Zip files, then you must download and install the [dependencies zip file](#).

Description	Download	Size	Last change
* Complete package, except sources	<a href="#">Setup</a>	1226215	7 April 2004

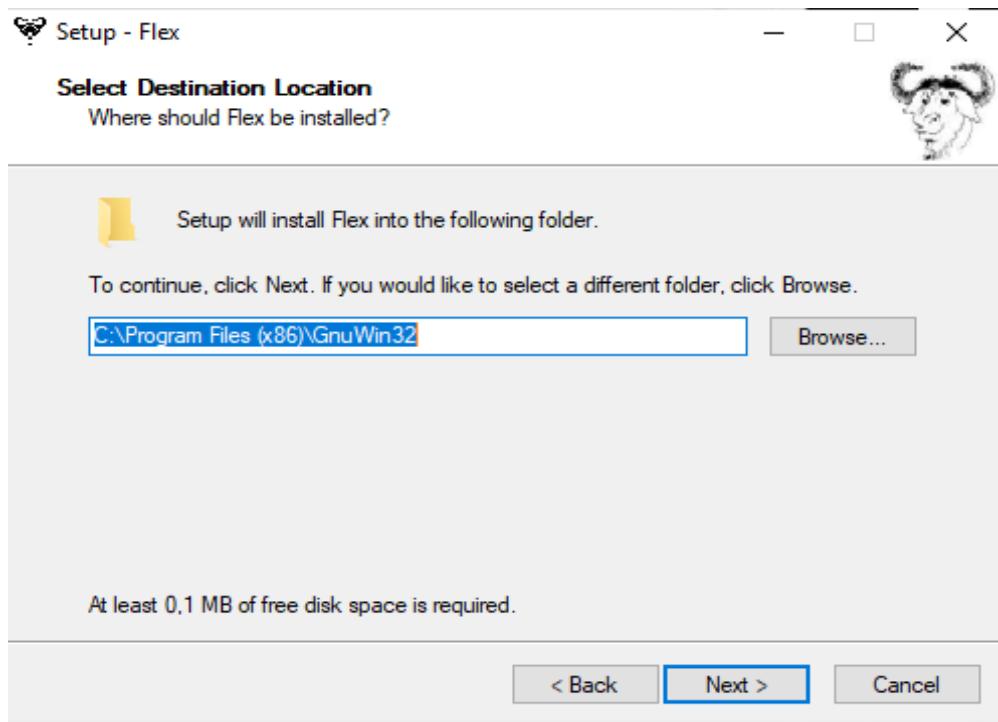
Una vez descargado el programa nos aparecerá esta primer ventana donde solo bastara con darle click en el botón de Next.



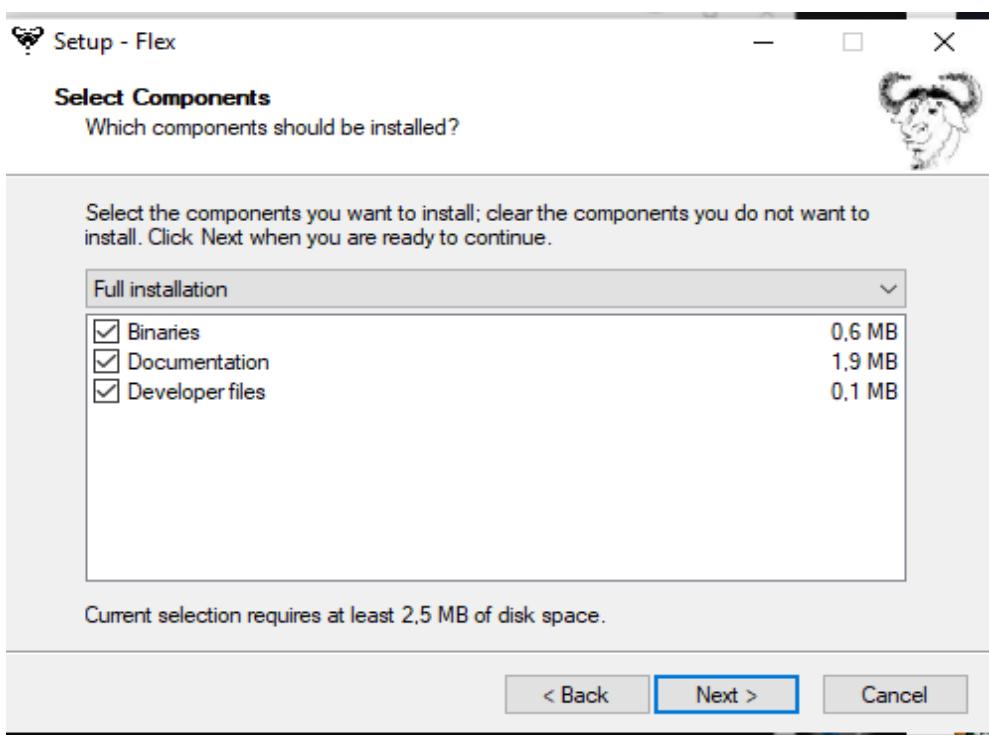
En la siguiente ventana marcaremos la opción de “I accept the agreement” y daremos click en el botón de Next.



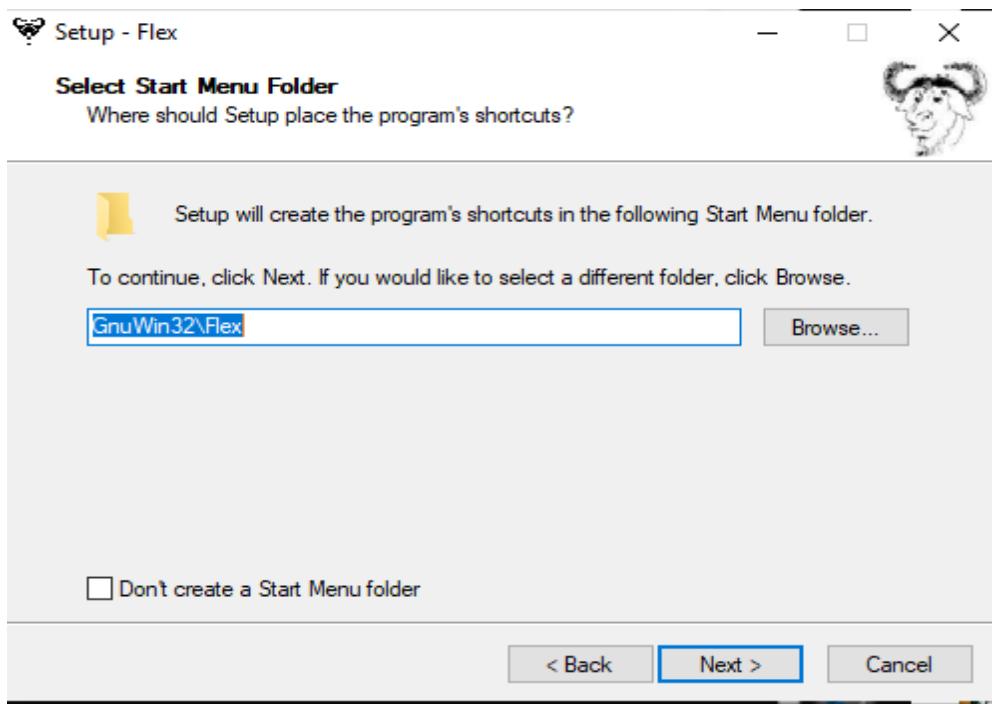
Solo basta en la siguiente pantalla dejar la carpeta por default donde se instalara y se guardaran los programas, asi que es recomendable solo dar click en el botón de Next.



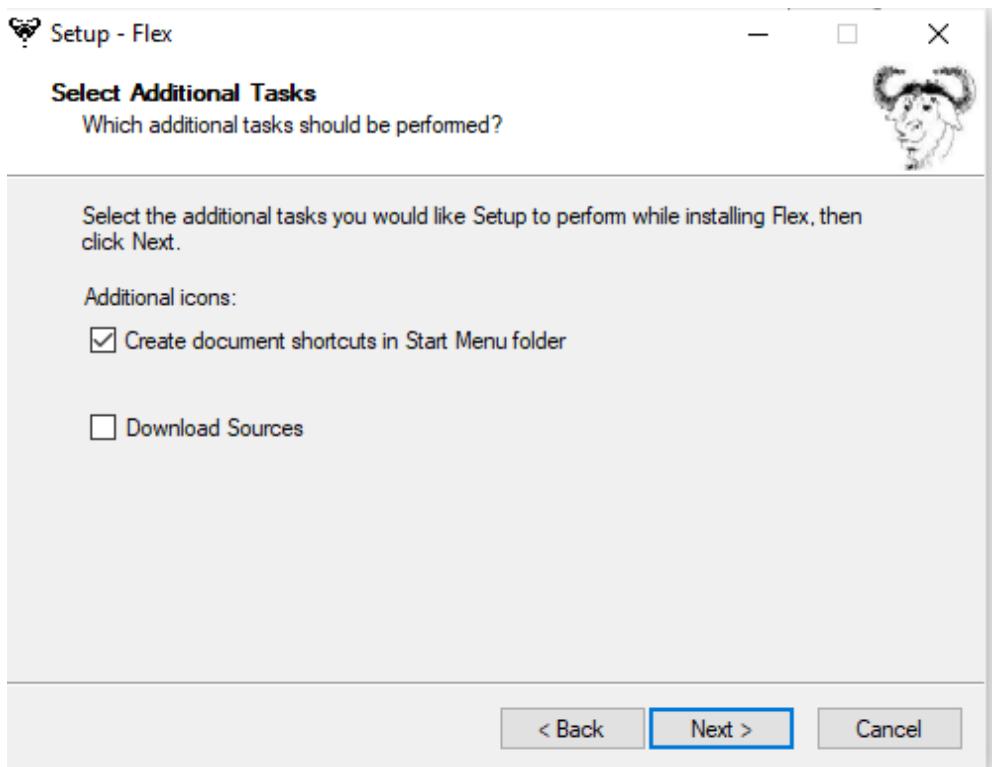
En la siguiente pantalla dejaremos las opciones marcadas por default y solo daremos click en el botón de Next.



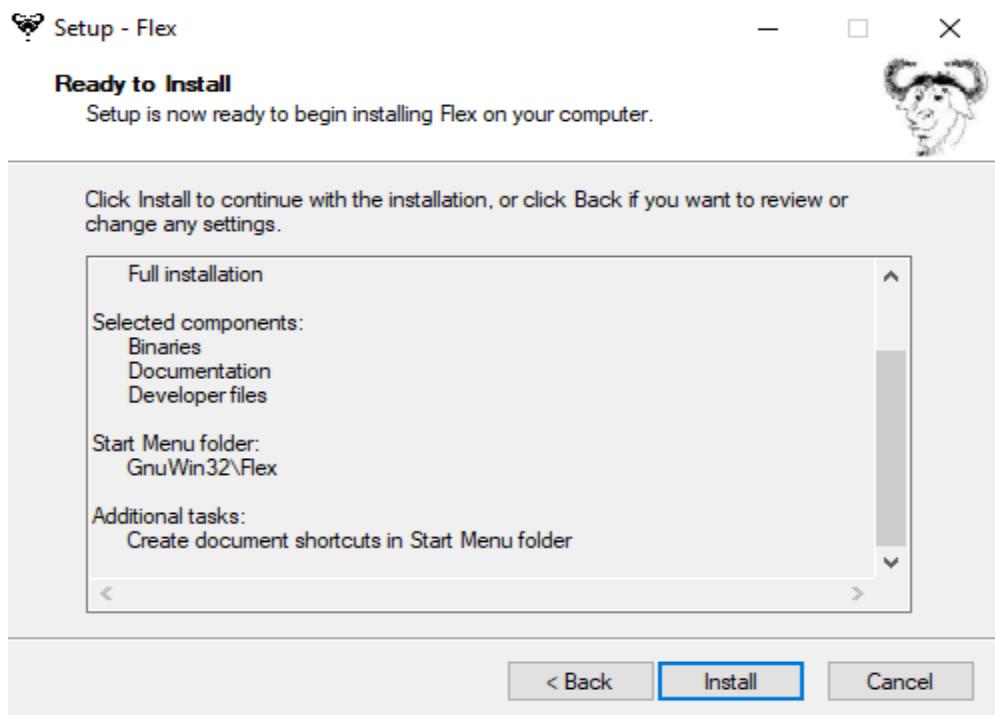
Nos aparecerá la siguiente carpeta y es recomendable no mover nada y solo dar click en el botón de Next.



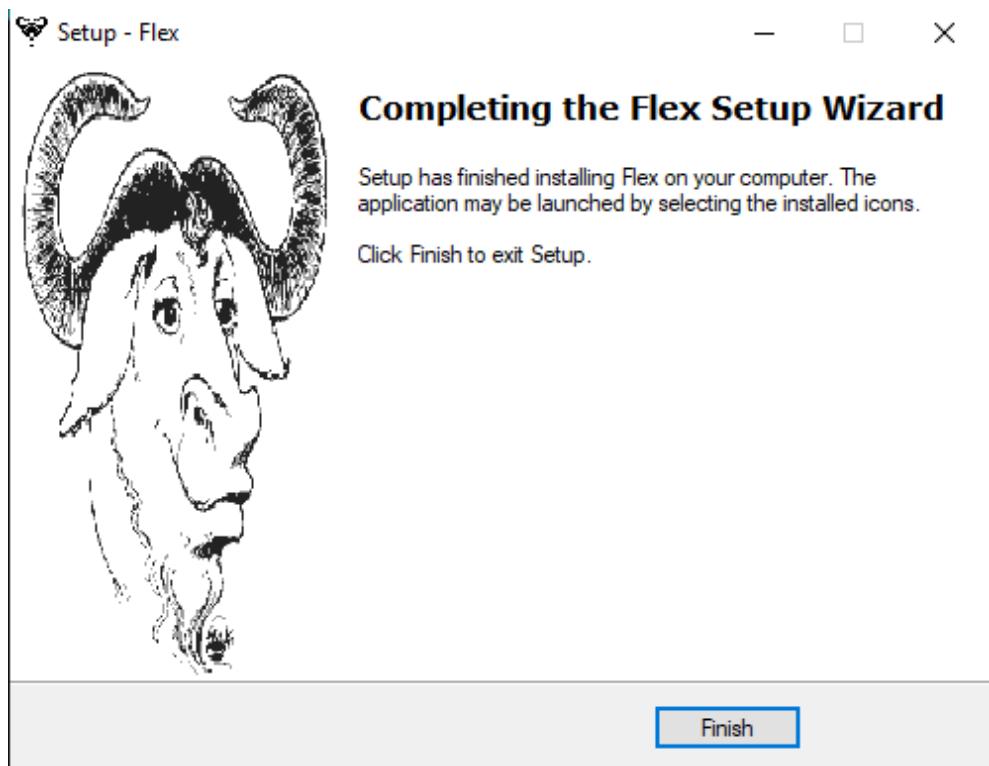
En la siguiente ventana marcaremos la opción de “Create document shortcuts in Start Menu Folder” y daremos click en el botón de Next.



Una vez seguido estos pasos daremos click en el botón de Install para que se pueda o termine de completar la instalación.



Una vez terminado todo el proceso solo daremos click en el botón de Finish para terminar con este proceso de instalación de Flex.

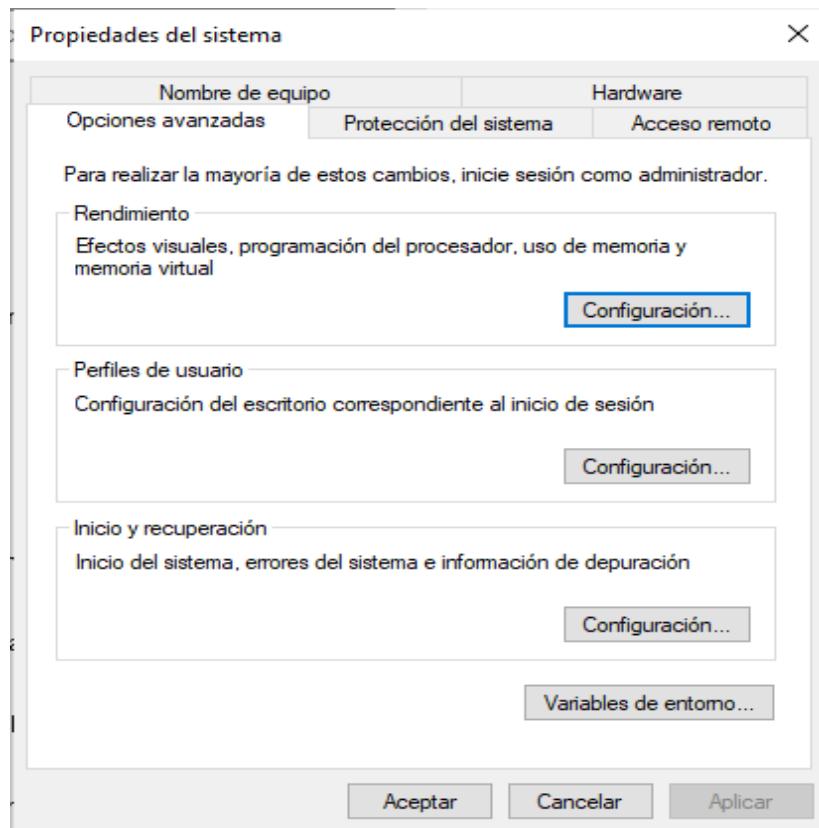


## Configurar rutas

Accederemos al sistema desde la pantalla principal e iremos a la opción de “Configuración avanzada del sistema”.



Una vez dando click en esa opción nos aparecerá la pantalla siguiente y daremos click en el botón de “Variables de entorno”.



Nos aparecerá la siguiente ventana donde seleccionaremos la opción de “Path” y una vez seleccionada daremos click en el botón de Editar.

Ajor  
Pach  
Daff

The screenshot shows the Windows Environment Variables dialog box. It has two main sections: 'Variables de usuario para Tesoro' (User variables for Tesoro) and 'Variables del sistema' (System variables). In the 'Variables de usuario' section, the 'Path' variable is selected and highlighted in blue. In the 'Variables del sistema' section, the 'Path' variable is also listed. At the bottom of each section are buttons for 'Nueva...' (New), 'Editar...' (Edit), and 'Eliminar' (Delete). Below the system variables section are buttons for 'Aceptar' (Accept) and 'Cancelar' (Cancel).

Variable	Valor
CABAL_DIR	C:\cabal
GHCUP_INSTALL_BASE_PREF...	C:\
OneDrive	C:\Users\Tesoro\OneDrive
<b>Path</b>	C:\Users\Tesoro\AppData\Local\Programs\Python\Python311\Scripts\;C:\Users\Tesoro\AppData\Local\Temp;C:\Users\Tesoro\AppData\Local\Temp
TEMP	C:\Users\Tesoro\AppData\Local\Temp
TMP	C:\Users\Tesoro\AppData\Local\Temp

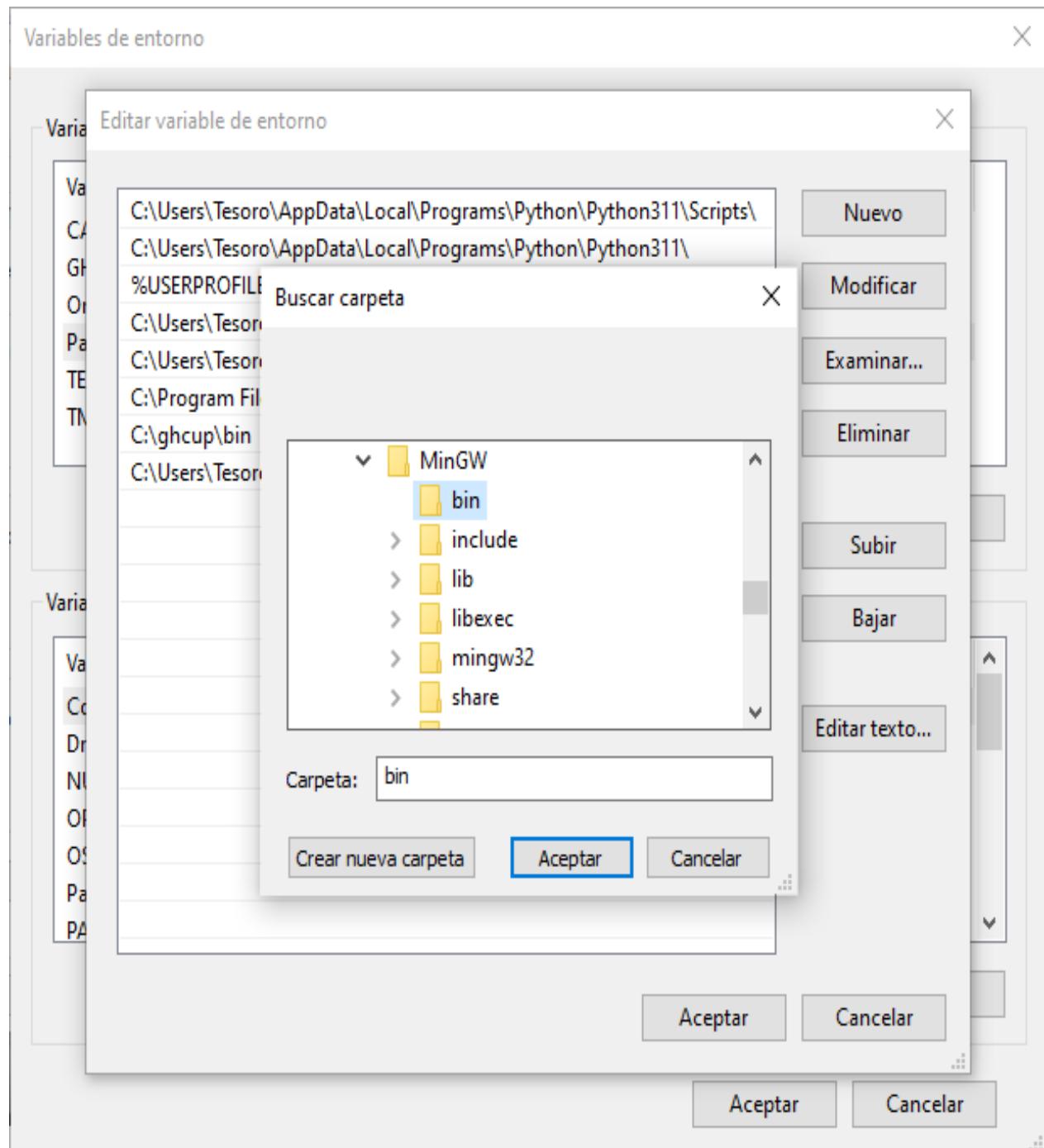
Variable	Valor
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OPENSSL_CONF	C:\Program Files\PostgreSQL\psqlODBC\etc\openssl.cnf
OS	Windows_NT
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program ...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

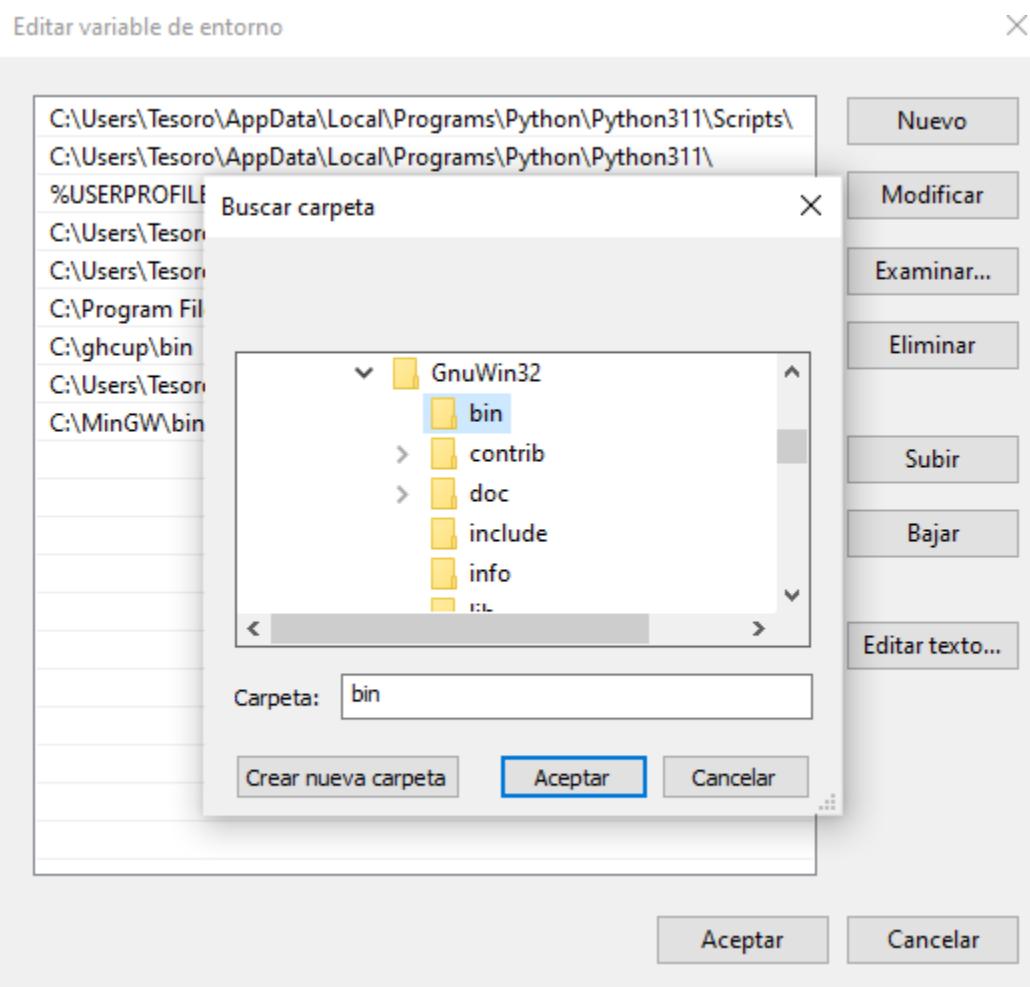
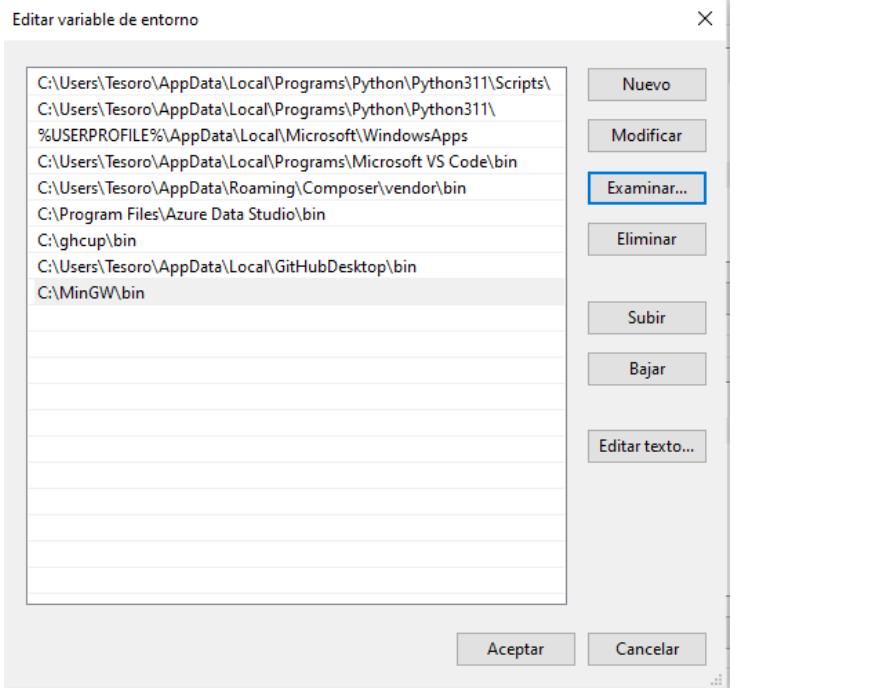
Nueva...    Editar...    Eliminar

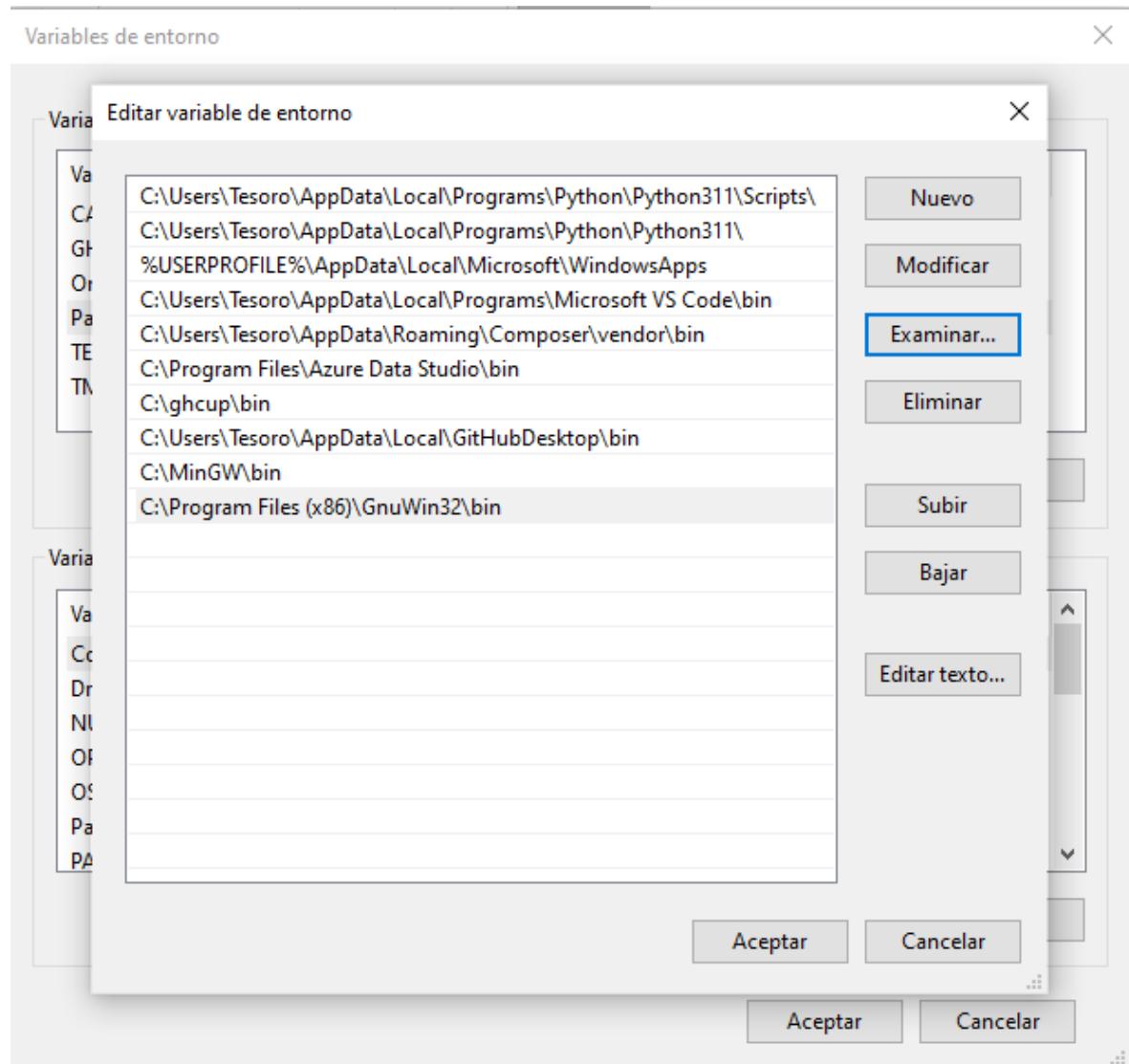
Nueva...    Editar...    Eliminar

Aceptar    Cancelar

En la siguiente ventana da daremos click en el botón de Nuevo y agregaremos la ruta de bin de MinGW y la ruta de bin de GnuWin32 y veremos que se agregan de manera exitosa como se pueden ver en las imágenes inferiores.







Una vez terminado todo este proceso, iremos a nuestra terminal para checar que nuestro lenguaje “C++” este instalado en una version completa y asi poder empezar a desarrollar los programas. Y de esta manera terminamos con la instalación de todo.

```
C:\ Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.3448]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>c++ --version
c++ (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\WINDOWS\system32>g++ --version
g++ (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

TECNOLÓGICO  
NACIONAL DE MÉXICO®**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4***[Handwritten signatures]*

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
1	Programa utilizando Flex	27 Septiembre 2023

1	INTRODUCCION
La gran mayoría de los programas necesita interactuar con archivos debido a que son en estos donde guardan la información con la que trabajarán, bien puede ser datos que necesitan tanto leer como escribir sobre información que está relacionada con el programa. Es por esto que resulta de gran importancia conocer la forma en que los archivos son creados y el cómo se puede implementar sus diferentes funciones en los lenguajes de programación.	
La creación de archivos en Flex se utiliza en una amplia variedad de aplicaciones, desde sistemas operativos y controladores de dispositivo, hasta aplicaciones de sistemas embebidos y firmware. La optimización del código es una preocupación importante en la creación de archivos, ya que la eficiencia del código puede tener un impacto significativo en el rendimiento y la velocidad de ejecución del programa.	
En esta práctica se explorarán las instrucciones necesarias en lenguaje Flex para poder reconocer ciertas palabras, como números y algunas asignaciones.	

2	OBJETIVO
Con esta práctica se busca que mediante la programación en Flex reconozca tanto como palabras, como números y algunas asignaciones como “=, +=, etc”, y así mismo que mande un error cuando algún carácter no sea válido.	

3	MATERIALES NECESARIOS
<ul style="list-style-type: none"><li>• 1 computadora que cumpla con los requisitos mínimos solicitados por Flex.<ol style="list-style-type: none"><li>1. Sistema operativo: Windows 2000 / 98 / XP / Vista / 7 / 8 / 10 / 11</li><li>2. Memoria (RAM): 256 MB</li><li>3. Espacio disco duro: 100 MB</li><li>4. Procesador: Intel de 750 Mhz</li></ol></li><li>• Flex.</li><li>• Visual Studio Code.</li><li>• Manuales o videos didácticos referentes al funcionamiento del uso de Flex.</li></ul>	

*[Handwritten signatures]*

TECNOLÓGICO  
NACIONAL DE MÉXICO®**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II****AUTOR: EQUIPO 4**

5

**DESARROLLO****Programa 1**

En esta práctica se busca realizar un pequeño programa donde va a reconocer ciertas palabras conformadas, como números y además de ciertas asignaciones dadas.

Para lograr esto, es necesario establecer un conjunto de métodos en donde se realizarán el proceso correspondiente a la opción seleccionada, entonces se siguió los siguientes pasos.

1. Se establece la estructura básica de la gran mayoría de los programas en Flex.

```
%{  
#include <stdio.h>  
%}
```

2. Se crean en este caso, los parametros que va a reconocer cuando se esté ejecutando el programa.

```
DIGITO [0-9]  
LETRA [a-zA-Z]  
IDENTIFICADOR ({LETRA}|_)({LETRA}|{DIGITO})*  
NUMERO {DIGITO}+  
ASIGNACION =|\\+=|-=
```

El significado u objetivo de cada una de las etiquetas son las siguientes.

**DIGITO [0-9]:** Esta definición define el patrón de un solo dígito decimal. Se utiliza para reconocer cualquier dígito del 0 al 9 en el código fuente. Por ejemplo, se usaría para reconocer números enteros.

**LETRA [a-zA-Z]:** Esta definición define el patrón de una sola letra en mayúscula o minúscula. Se utiliza para reconocer caracteres alfabéticos en el código fuente. Por ejemplo, se usaría para reconocer identificadores de variables o nombres de funciones.

**IDENTIFICADOR ({LETRA}|\_)({LETRA}|{DIGITO})\*:** Esta definición define el patrón de un identificador válido en un lenguaje de programación. Comienza con una letra o un guion bajo (\_) y puede seguirse de letras, dígitos o guiones bajos. Se utiliza para reconocer nombres de variables, funciones y otros identificadores en el código fuente.

**NUMERO{DIGITO}+:** Esta definición define el patrón de un número entero válido. Consiste en uno o más dígitos. Se utiliza para reconocer números enteros en el código fuente.

**ASIGNACION =|\\+=|-=:** Esta definición define el patrón de los operadores de asignación. Reconoce el operador de asignación simple (=) y los operadores de asignación compuesta (+= y -=). Se utiliza para reconocer operaciones de asignación en el código fuente.

TECNOLÓGICO  
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II****AUTOR: EQUIPO 4**

Las siguientes etiquetas corresponden a los mensajes que se imprimirán en pantalla, siendo de estos los establecidos en la salida solicitada.

```
%%
{ IDENTIFICADOR} { printf("IDENTIFICADOR: %s\n", yytext); }
{ NUMERO} { printf("NUMERO: %s\n", yytext); }
{ ASIGNACION} { printf("ASIGNACION: %s\n", yytext); }
[^a-zA-Z0-9_+=-] { printf("ERROR: Carácter no valido en asignación:
%s\n", yytext); }
. /* Ignorar otros caracteres */
%%
```

**{IDENTIFICADOR} { printf("IDENTIFICADOR: %s\n", yytext); }:** Esta regla define un patrón llamado {IDENTIFICADOR} que coincide con secuencias de caracteres que representan identificadores (por ejemplo, nombres de variables o funciones). Cuando se encuentra un identificador en el archivo de entrada, se ejecuta la acción asociada, que imprime "IDENTIFICADOR:" seguido del texto coincidente (el identificador) utilizando printf.

**{NUMERO} { printf("NUMERO: %s\n", yytext); }:** Similar al caso anterior, esta regla define un patrón llamado {NUMERO} que coincide con secuencias de caracteres que representan números. Cuando se encuentra un número en el archivo de entrada, se ejecuta la acción asociada, que imprime "NUMERO:" seguido del texto coinciente (el número) utilizando printf.

**{ASIGNACION} { printf("ASIGNACION: %s\n", yytext); }:** Esta regla define un patrón llamado {ASIGNACION} que coincide con secuencias de caracteres que representan un operador de asignación. Cuando se encuentra este operador en el archivo de entrada, se ejecuta la acción asociada, que imprime "ASIGNACION:" seguido del texto coinciente (el operador de asignación).

**[^a-zA-Z0-9\_+=-] { printf("ERROR: Carácter no valido en asignación: %s\n", yytext); }:** Esta regla define un patrón que coincide con cualquier carácter que no sea una letra, un número, ni los caracteres especiales '\_', '+', '=', o '!'. Cuando se encuentra un carácter no válido en una asignación, se ejecuta la acción asociada, que imprime un mensaje de error que indica que el carácter no es válido.

.. Este patrón coincide con cualquier otro carácter que no haya sido especificado en las reglas anteriores. La acción asociada es ignorar estos caracteres.

Después imprimirá el valor y a que pertenece, ya sea si es una palabra reconocida, un número o alguna asignación.

```
int main() {
yylex();
return 0;
}
```

Al momento de ejecutar el código con la anterior parte mencionada, se ve de la siguiente manera una vez que empezamos a hacer prueba por prueba.



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

Como primera instancia haremos que reconozca una palabra cualquiera.

```
C:\C++>a
Primera Prueba
IDENTIFICADOR: Primera
ERROR: Caracter no valido en asignacion:
IDENTIFICADOR: Prueba
ERROR: Caracter no valido en asignacion:
```

Como segunda parte, haremos que reconozca algún número.

```
12345
NUMERO: 12345
ERROR: Caracter no valido en asignacion:
```

Ahora haremos que reconozca una asignación, como por ejemplo un igual (=).

```
=
ASIGNACION: =
ERROR: Caracter no valido en asignacion:
```

Por ultimo haremos que se provoque errores distintos, en este caso agregaremos un @ y dira que no reconoce el carácter.

```
Primer@ Prueb@
IDENTIFICADOR: Primer
ERROR: Caracter no valido en asignacion: @@
ERROR: Caracter no valido en asignacion:
IDENTIFICADOR: Prueb
ERROR: Caracter no valido en asignacion: @@
ERROR: Caracter no valido en asignacion:
```

De esta manera podemos observar que no paso la primera prueba con la primera palabra ya que no reconoció el @, y en la segunda palabra de esta misma forma no volvió a reconocer el @.

TECNOLÓGICO  
NACIONAL DE MÉXICO®**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II****AUTOR: EQUIPO 4****6****BITÁCORA DE INCIDENCIAS**

Fecha	Problema encontrado	Solución
22/09/2023	No podía compilarse el programa, ya que faltaba agregar una ruta a nuestra computadora para que reconociera la programación de Flex.	Se consultó un video en YouTube, donde explicaban que rutas debían ponerse adecuadamente, para la compilación de Flex.

**7****OBSERVACIONES**

Durante el desarrollo de este programa, a pesar de que era un programa sencillo y no tan largo en líneas de código, se presentaron algunos problemas que fueron solucionados poco a poco, además de que nos empezamos a enrolar por primera vez al utilizar Flex, ya que era para nosotros la primera vez en desarrollar un programa relacionado con esto.

Fue importante ir reconociendo cada archivo y cada sentencia que se generaba, para de esta forma cumplir con lo solicitado.

**8****CONCLUSIÓN**

Al terminar esta práctica se pudieron comprender los diferentes puntos que se tenían que instalar y que pasos se deberían de seguir para poder completar con el programa y lograr con lo solicitado en esta práctica.

Además, que se dijo optar por hacer la programación en el entorno de Visual Studio Code, ya que la mayoría del equipo o incluso todos hemos desarrollados diferentes programa en este entorno y es fácil agregar las extensiones y así mismo lograr con el objetivo.

**9****REFERENCIAS**

- Gadgetas [@Gadgetas]. (2019, septiembre 24). Instalación Flex, minGW, c++ para Windows - Compiladores analizador léxico. Youtube.  
[https://www.youtube.com/watch?v=nFGcPIW\\_rOw](https://www.youtube.com/watch?v=nFGcPIW_rOw)
- MinGW - Minimalist GNU for Windows. (2023, septiembre 23). SourceForge.  
<https://sourceforge.net/projects/mingw/>
- Flex. (s/f). Sourceforge.net. Recuperado el 26 de septiembre de 2023, de  
<https://gnuwin32.sourceforge.net/packages/flex.htm>

# INGENIERÍA EN SISTEMAS COMPUTACIONALES

"Gramáticas Libres de Contexto y Árboles de Derivación"

"Diagramas de Sintaxis"

## Alumnos

Cardoso Morales Cristian David  
González Rodríguez Francisco  
Núñez Servín Juan Ángel

## Maestro

ISC. Ricardo González González

## TABLA DE CONTENIDO

Indicé de cuadros.....	2
Introducción .....	3
Capítulo 1: Fundamentos de Gramáticas Libres de Contexto: .....	6
1.1. Definición de Gramáticas Libres de Contexto (GLC) .....	6
1.2 Ejemplos de GLC simples.....	10
1.3. Equivalencia entre GLC y lenguajes regulares.....	11
Capítulo 2: Capítulo 2: Árboles de Derivación: .....	13
1.1. Definición de Gramáticas Libres de Contexto (GLC) .....	6
1.2 Ejemplos de GLC simples.....	10
1.3. Equivalencia entre GLC y lenguajes regulares.....	11
Sección de reglas .....	13
Comparación de generadores en términos de características y usos .....	14
Estructura de un analizador léxico generado .....	15
Concordancia de patrones .....	16
Aplicaciones y significado .....	17
Aplicaciones ( caso de estudio ) .....	18
DSL .....	18
Concepto y utilidad .....	18
Desarrollo de un DSL para matemáticas simbólicas .....	19
Rol de los analizadores léxicos en el DSL .....	20
Definición de la gramática .....	20
Implementación del analizador léxico .....	20
Escaneo del código fuente .....	21
Validación y generación de árboles sintácticos .....	21
Aplicaciones prácticas del DSL .....	22
Resolución de ecuaciones simbólicas .....	22
Investigación matemática .....	22
Educación .....	22
Desafíos y evolución .....	23
Conclusión .....	23

## INDICE DE CUADROS

Figura 1. Componentes de GLC .....	8
------------------------------------	---

Figura 2. Jerarquía Chomsky .....	12
Tabla 1. Ejemplos de tokens .....	8
Figura 3. Proceso analizadores .....	9
Figura 4. Proceso de LEX .....	10
Figura 5. Proceso de FLEX .....	11
Figura 6. Proceso de ANTLR .....	12
Figura 7. Proceso de JFLEX .....	13
Tabla 2. Comparación de generadores .....	15
Figura 8. Estructura general .....	16
Figura 9. Ejemplo de un AFND .....	17
Figura 10. Introducción DSL .....	20
Figura 11. Comportamiento DSL .....	21
Figura 12. Proceso de aplicación DSL .....	22

# INTRODUCCIÓN

En la introducción de este estudio plantearemos el problema que nos ha llevado a explorar lo fascinante de los diagramas de sintaxis. Este tema se fundamenta en la creciente importancia de la representación visual del lenguaje en diversas disciplinas, desde la lingüística computacional hasta la enseñanza de idiomas y la comunicación efectiva.

Se centrará en comprender y analizar cómo los diagramas de sintaxis se utilizan como herramientas poderosas para visualizar y comprender la estructura gramatical de los lenguajes, lo que a su vez puede mejorar la comprensión y generación de texto en sistemas de inteligencia artificial. Este tema responde a la necesidad de abordar un área específica y profundizar en su estudio.

Así que como tal las gramáticas son una herramienta fundamental, ya que desempeñan un papel esencial en la descripción y el análisis de los lenguajes formales.

Desde el inicio de la computación, las gramáticas han sido una piedra angular para modelar la sintaxis de los lenguajes y ayudar a las máquinas a comprender y procesar información textual.

En este contexto, es esencial comprender las gramáticas formales, que son conjuntos de reglas precisas que definen cómo se generan las cadenas de símbolos en un lenguaje.

Estas gramáticas formales, se utilizan para describir la estructura de lenguajes y se dividen en varias categorías según su capacidad expresiva y reglas de derivación.

El objetivo de esta monografía es explorar a fondo las gramáticas libres de contexto y su relación con los árboles de derivación, que estos últimos proporcionan una representación visual clave de la estructura de las cadenas generadas por las gramáticas.

Así como también se verá como el contexto influye en la interpretación y comprensión de máxima profundidad de estos conceptos fundamentales en la teoría de la computación y la lingüística formal.

# Capítulo 1: Fundamentos de Gramáticas Libres de Contexto

En la teoría de la computación, las GLC (Gramáticas Libres de Contexto) se utilizan para definir la sintaxis de los lenguajes de programación, lo que permite a los compiladores y analizadores sintáticos verificar la corrección de software.

Cabe mencionar que en el procesamiento de lenguaje natural, las GLC son esenciales para analizar y generar texto en diversos idiomas.

Así que en este primer capítulo, estableceremos las bases fundamentales de las Gramáticas Libres de Contexto.

## 1.1 Definición de Gramáticas Libres de Contexto (GLC):

Dentro de las GLC's están formalmente estructuradas por lo siguiente y son un tipo específico de gramática formal utilizadas para la descripción de dicha estructura sintáctica de los lenguajes formales.

Y se define en una cuádrupla  $G = (N, \Sigma, P, S)$ , donde:

- **Símbolos no terminales ( $N$ ):** Conjunto finito que contienen categorías gramáticas abstractas que representan elementos dentro del lenguaje que estamos describiendo.

Estos símbolos no terminales son como etiquetas que se utilizan para estructurar y dar sentido a las cadenas generadas por la gramática.

Por ejemplo: en una gramática para el lenguaje de las expresiones matemáticas, los símbolos no terminales podrían representar elementos como:

Expresión  
Operador  
Número

• **Símbolos terminales ( $\Sigma$ ):** Conjunto finito que son los elementos reales del lenguaje que se describe.

Son las palabras, números, operadores o símbolos concretos que forman las cadenas válidas en el lenguaje.

En la gramática para expresiones matemáticas, los símbolos terminales podrían ser números (1, 2, 3, etc.), operadores (\*, +), y parentesis [( )].

• **Reglas de producción ( $P$ ):** Conjunto finito y son las reglas que indican cómo se pueden combinar los símbolos no terminales y terminales para generar cadenas en el lenguaje.

Estas reglas establecen las reglas sintácticas dentro del lenguaje y definen la estructura gramatical.

Por ejemplo, una regla de producción podría indicar que una "expresión" puede ser una "expresión + expresión", lo que define la forma en que se pueden sumar dos expresiones.

**Símbolo inicial ( $S$ ):** El símbolo inicial es el punto de partida para generar cadenas válidas en el lenguaje.

Todas las derivaciones en la gramática comienzan con el símbolo inicial y avanzan a través de las reglas de producción para construir la cadena deseada.

Y es el punto de partida que define qué tipo de cadenas se pueden generar con la gramática.

$$G = (N, \Sigma, P, S)$$

Lenguaje  
Libre de  
Contexto

No Terminales

Terminales

Reglas de Producción

Símbolo inicial

Figura 1. Componentes de GLC

Estos componentes definen una GLC como una herramienta formal.

Por lo cual comprender estos componentes es esencial para el análisis sintáctico y la comprensión de la sintaxis de un lenguaje formal.

## 1.2 Ejemplo de GLC simple

Gramática para el lenguaje de paréntesis bien balanceado

$$S \rightarrow (S) \mid SS \mid \epsilon$$

Analizando sus componentes:

Símbolo inicial:  $S$ , lo que significa que cualquier cadena generada comienza con  $S$ .

Símbolos terminales: Son aquellos que no pueden reemplazarse por otros símbolos en el proceso de derivación y que forman parte de las cadenas válidas en el lenguaje.

En este caso, son

- $= "(" \rightarrow$  paréntesis abierto
- $= ")" \rightarrow$  paréntesis cerrado

Símbolos no terminales: símbolos abstractos representando categorías gramaticales.

Siendo en este caso  $S$

*Ayer*  
*Hoy*  
*Mañana*

Reglas de producción: Se tienen 3 reglas y son las siguientes

- $S \rightarrow "(S)"$ , que un paréntesis abierto, seguido de  $S$ , seguido de un paréntesis cerrado es una cadena válida.
- $S \rightarrow SS$ , lo que indica que dos cadenas  $S$  se pueden concatenar para formar una cadena válida.
- $S \rightarrow \epsilon$  (cadena vacía), una cadena vacía también es válida.

### 1.3 Equivalencia entre GLC y Lenguajes Regulares

- **GLC y su capacidad expresiva:** Las Gramáticas Libres de Contexto están y admiten una capacidad expresiva significativamente mayor que los Lenguajes Regulares.

Las GLC pueden describir estructuras más complejas y gramáticas más ricas que los Lenguajes Regulares. Pueden representar lenguajes con anidamiento, como paréntesis balanceados, así como lenguajes que requieren un contexto más amplio para su descripción, como el reconocimiento de expresiones aritméticas.

- **Lenguajes Regulares y su simplicidad:** Los Lenguajes Regulares son el tipo más simple de lenguaje formal.

Se pueden describir mediante gramáticas regulares y son adecuados para describir patrones simples, como cadenas de texto que cumplen patrones fijos.

Por ejemplo cadenas como "ababab" o "ab"

- **La Jerarquía de Chomsky:** La relación entre GLC y LR se enmarca en esta jerarquía, propuesta por el lingüista Noam Chomsky.

Dicha jerarquía establece cuatro niveles de gramáticas, llenando desde la más simple (Tipo 3 - Lenguajes Regulares) hasta la más compleja (Tipo 0 - Lenguajes sin Restricciones)

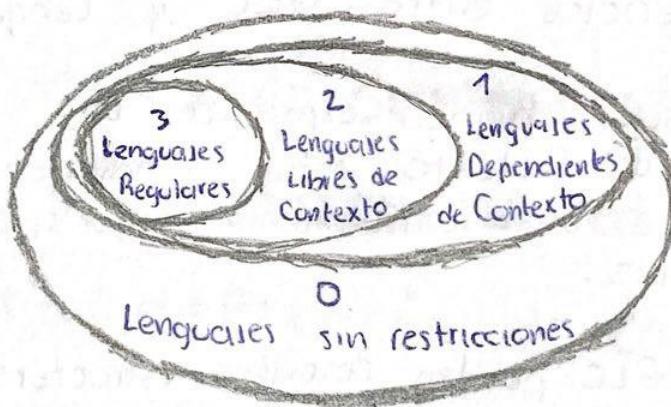


Figura 2. Jerarquía Chomsky

- **Equivalencia y Contenidos de Lenguajes:** Todos los Lenguajes Regulares pueden ser generados por gramáticas regulares, que son un tipo específico de GLC más simple. Sin embargo, no todos los lenguajes Libres de Contexto pueden ser generados por gramáticas regulares.

Lo que significa que los L.R. son un subconjunto de L.L.C

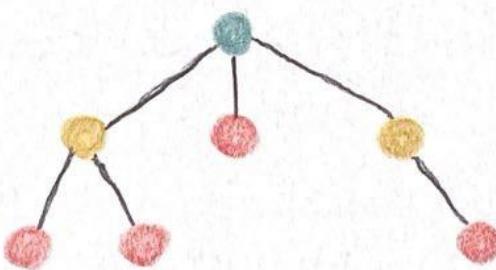
# Capítulo 2: Árboles de Derivación

La importancia de estos Árboles como una representación gráfica clave en el proceso de generación de cadenas a partir de una GLC.

## 2.1 Definición y estructura de los árboles de derivación

Son estructuras jerárquicas en forma de árbol que ilustran cómo una cadena de símbolos se deriva a partir del símbolo, ya sea terminal o no terminal, y las ramas del árbol indican cómo se aplican las reglas de producción de la gramática.

En la parte superior del arbol se encuentra el símbolo inicial, y en las hojas del árbol se encuentra la cadena derivada, leída de izquierda a derecha.



- Nodo raíz
- Nodo interior
- Hojas

Figura 3. Estructura de Árbol

## 2.2 Cómo construir un árbol de derivación

Para construir un árbol de derivación a partir de una cadena, se sigue un proceso iterativo basado en las reglas de producción de la GLC:

- Comienza con el símbolo inicial en la raíz del árbol
- Se aplican reglas de producción para remplazar un símbolo no terminal en un nodo por su derivación correspondiente.
- Se continua el proceso hasta que todos los nodos en el árbol sean símbolos terminales y la cadena completa se haya derivado.

## 2.3 Ejemplos de árboles de derivación para gramáticas y cadenas específicas

Los siguientes ejemplos ayudarán a ilustrar como los árboles de derivación se utilizan en la práctica y cómo representan visualmente el proceso de derivación de cadenas.

**Ejemplo 1:** Gramática para el lenguaje de paréntesis bien balanceados

$$S \rightarrow (S) \mid SS \mid E$$

$$\text{Cadena: } =(())"$$

Árbol de Derivación:

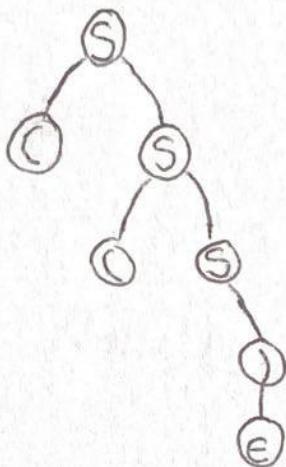


Figura 4. Árbol E.1

Ejemplo 2: Gramática para expresiones aritméticas

$$\text{Expr} \rightarrow \text{Expr} + \text{Expr} \mid \text{Expr} * \text{Expr} \mid (\text{Expr}) \mid \text{num}$$

Expresión  
ó  
Cadena

$$= 2 + 3 * 4$$

Árbol de Derivación:

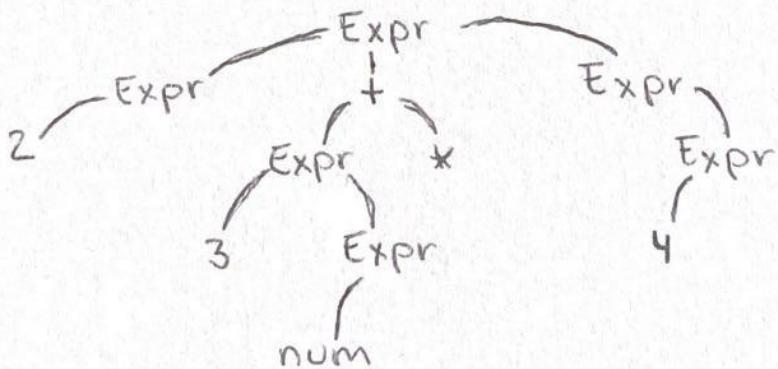


Figura 5. Árbol E.2

### Ejemplo 3: Gramática para oraciones en inglés

Sentence → Subject Verb Object  
Subject → I | You | He | She  
Verb → play | eat | study  
Object → chess | pizza | math

Oración: "I play chess"

Árbol de Derivación:

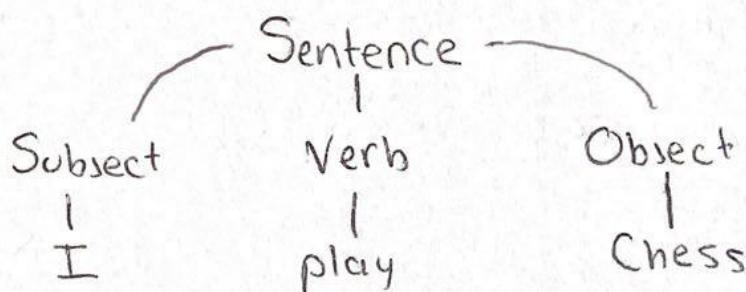


Figura 6. Árbol Ej. 3

# Análisis sintáctico

Los lenguajes de Programación tienen una serie de reglas que nos ayudan a describir la estructura sintáctica de los programas bien estructurados que acepta:

En Pascal:

El Programa

Bloques

Bloques de sentencias

una Sentencia de expresiones

una expresión de comp. léxicos

:  
:  
;

Caracteres básicos

Figura 7 Estructura sintáctica de un Programa en Pascal

Se puede describir la sintaxis de las construcciones de los lenguajes de programación por medio de gramáticas libres de contexto o notación BNF (Backus-Naur-Form).

La fase de análisis sintáctico tiene por objetivo solicitar tokens al analizador léxico y construir una representación al texto de entrada en base a una gramática dada. El proceso se encuentra dirigido por la gramática. En caso de que la entrada sea válida, suministra al árbol sintáctico.

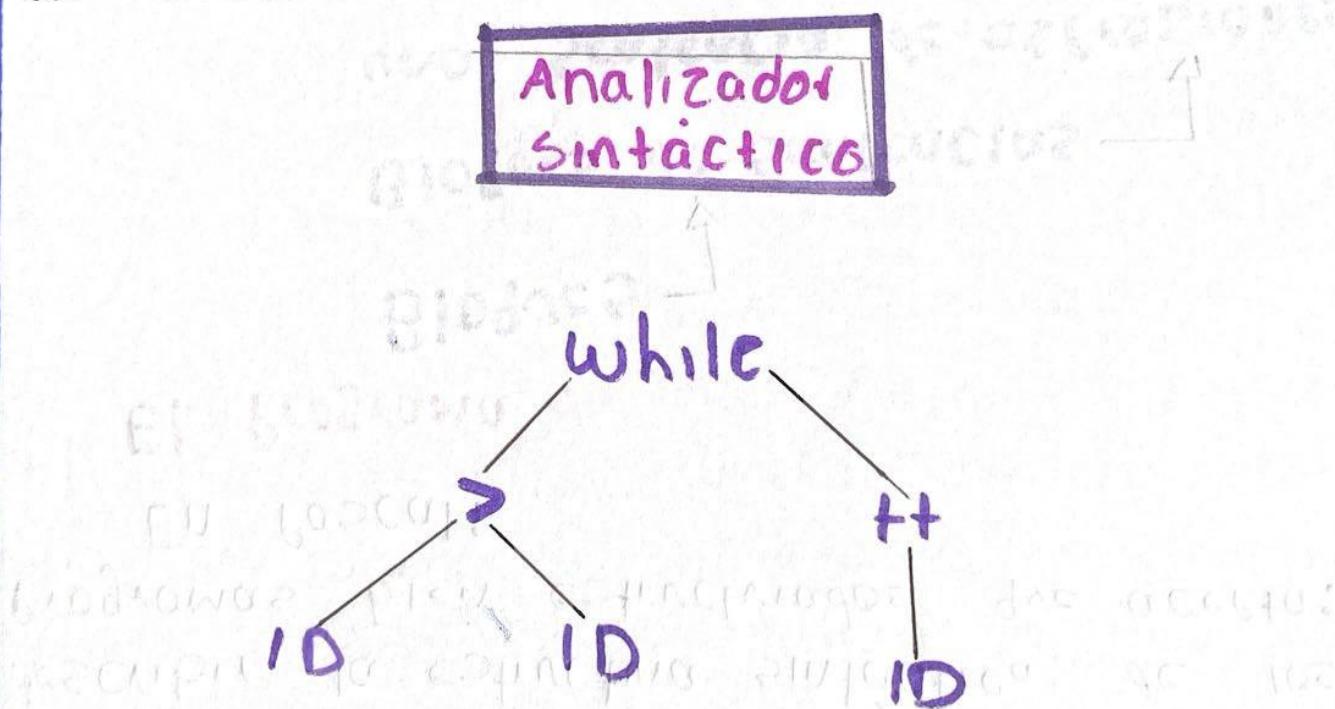


Figura 8 ejemplo árbol sintáctico

En la práctica, el analizador sintáctico dirige el proceso de compilación, a menudo el árbol ni siquiera se genera realmente.

# CONCEPTO DIAGRAMA DE SINTAXIS

Una gramática libre de contexto es un mecanismo para expresar un lenguaje formal. Pero se debe entender que no es el único, ya que existen varias alternativas como la notación BNF y los diagramas de sintaxis los cuales tienen la misma potencia. En el campo de la compilación, representaciones textuales como el ya mencionado BNF o sus variantes son más preferidas generalmente. BNF es bien entendido por los autores de compiladores y compiladores, pero no es bien entendido por la mayoría de los usuarios de un lenguaje.

Es aquí donde aparecen los diagramas sintáticos o de sintaxis, que representa una gran alternativa gráfica para la forma BNF (Backus-Naur Form).

Los diagramas sintácticos son formas de representar una gramática libre de contexto y sus restricciones sintácticas.

Cada diagrama de sintaxis está etiquetado con el nombre de un no terminal al que se representa como rectángulos y los terminales como círculos o ellipses unidas por flechas que indican secuencia y selecciones.

Ejemplo:

Dada la gramática

$$E ::= E + T \mid E - T \mid T$$

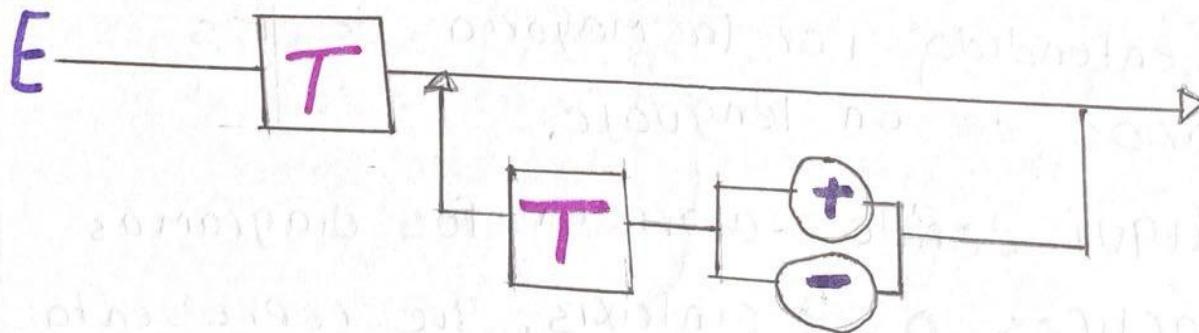


Figura 9 Ejemplo diagrama de sintaxis para una gramática. La equivalencia entre gramáticas y diagramas de sintaxis resulta evidente.

# Características de diagramas de sintaxis

Los diagramas de sintaxis para representar gramáticas libres de contexto (GLC) tienen ciertas características específicas que los hacen adecuados para modelar y visualizar la estructura gramatical de los lenguajes. Las más importantes son:

- **Jerarquía y recursividad:** Reflejan la jerarquía en la estructura gramatical del lenguaje, lo que significa que puede presentar recursividad, donde las mismas reglas gramaticales se aplican repetidamente para construir estructuras más complejas.
- **Símbolos no terminales y terminales:** Se distinguen entre símbolos no terminales y terminales. Los símbolos no terminales representan categorías gramaticales generales, mientras que los símbolos terminales representan palabras específicas del vocabulario.

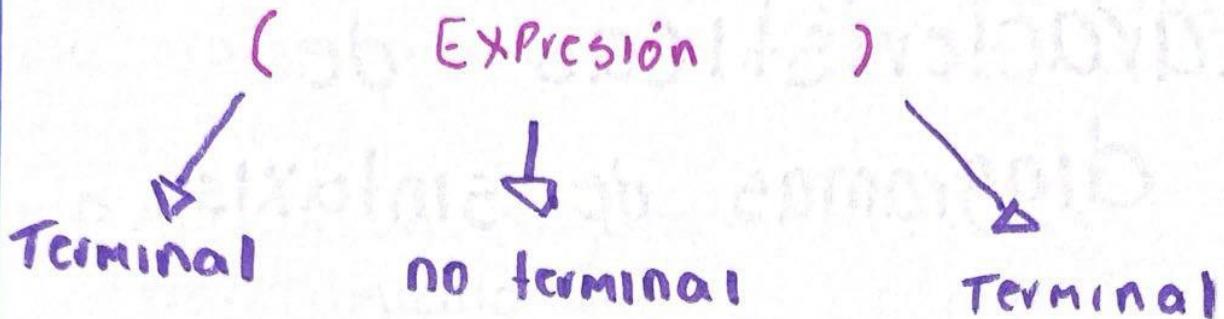


Figura los símbolos terminales y no terminales

- Permite ver derivaciones al instante en que ocurren.
- Producciones de reglas gramaticales: Los nodos no terminales representan símbolos no terminales. Cada nodo representa una producción de una regla gramatical.
- Gramáticas formales: Los diagramas de sintaxis para GLC son una representación visual de gramáticas formales. Estas gramáticas se basan en reglas precisas y estructuras gramaticales bien definidas, lo que permite un análisis más preciso.
- Gramáticas contextuales: Aunque los diagramas de sintaxis son ideales para GLC, también pueden adaptarse para representar gramáticas

contextuales más complejas que permiten un mayor contexto y reglas más flexibles

→ Aplicación en análisis sintáctico y procesamiento del lenguaje natural:

Los diagramas de sintaxis se utilizan en análisis sintáctico para descomponer y analizar la estructura de las oraciones y un procesamiento del lenguaje natural para comprender y generar texto de manera automática.

## Poder de los diagramas de sintaxis

La representación de una gramática que se haga de un conjunto de diagramas de sintaxis. Cada diagrama define a un no terminal. Hay un diagrama principal que define el idioma de la siguiente manera: Para pertenecer a la lengua, una palabra debe describir una trayectoria en el diagrama principal.

Los diagramas de sintaxis pueden representar las mismas gramáticas que la notación BNF, por inducción las operaciones básicas de BNF. Demostmando con la siguiente tabla:

Operación	BNF	Diagrama de sintaxis
XuxtaPosición	$AB$	$\rightarrow A \rightarrow B \rightarrow$
OPCIÓN	$A   B$	$\begin{array}{c} \rightarrow A \\ \parallel \\ \rightarrow B \end{array} \rightarrow$
	$E   B$	$\begin{array}{c} \rightarrow E \\ \parallel \\ \rightarrow B \end{array} \rightarrow$
Repetición	1 o más veces $[B]$	$\begin{array}{c} \rightarrow B \\ \nearrow \searrow \\ \rightarrow \end{array}$
	0 o más veces $[B]$	$\begin{array}{c} \rightarrow \\ \parallel \\ \rightarrow B \end{array} \rightarrow$

Tabla 01 COMPARACIÓN BNF CON DIAGRAMA DE SINTAXIS  
 en base a la tabla anterior, es posible convertir cualquier expresión BNF en su correspondiente diagrama de sintaxis, ya que A y B pueden ser, a su vez, expresiones o diagramas complejos.

# Reglas del diagrama de sintaxis

## • Rutas de acceso

La ruta de acceso principal comienza a la izquierda con una Palabra clave y continúa, de izquierda a derecha, hasta la barra vertical, que marca el final del diagrama. Las rutas de acceso que no continúen una flecha o una barra vertical solo muestra parte de la sintaxis.

## • Enlaces de continuación

Las rutas de acceso que son muy largas para una línea continua utilizan enlaces de continuación. Estos están denotados por letras dentro de un círculo que es el inicio y el final de un enlace:

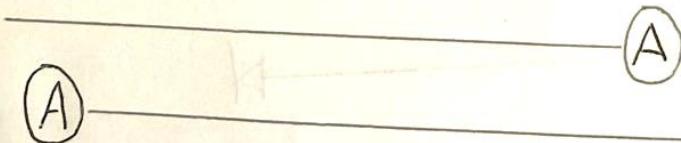


Figura 11 EJEMPLO enlaces de continuación

## • Entradas necesarias

Las entradas necesarias aparecen en la ruta principal. Si se pueden elegir más de una entrada, las opciones aparecen en una pila, la primera aparece en la ruta principal.

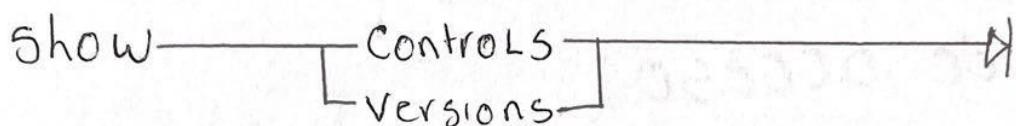


Figura 12 ejemplo de entradas necesarias

## • Entradasopcionales

Se Pueden elegir o ignorar estas entradas. Estas entradas aparecen debajo de la línea principal.

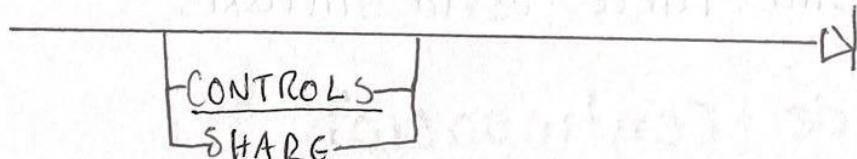


Figura 13 ejemplo de entradasopcionales

Algunos Comandos Pueden tener un valor opcional como Predeterminado. Este valor aparece subrayado.

## • Cadenas

Aparecen entre apóstrofes

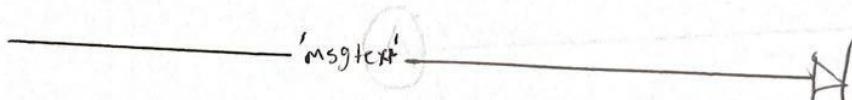


Figura 14 ejemplo de cadenas

## • Abreviaturas

Si una palabra clave o una palabra reservada tiene una abreviatura válida, la forma completa aparece en la vía principal y la otra debajo.

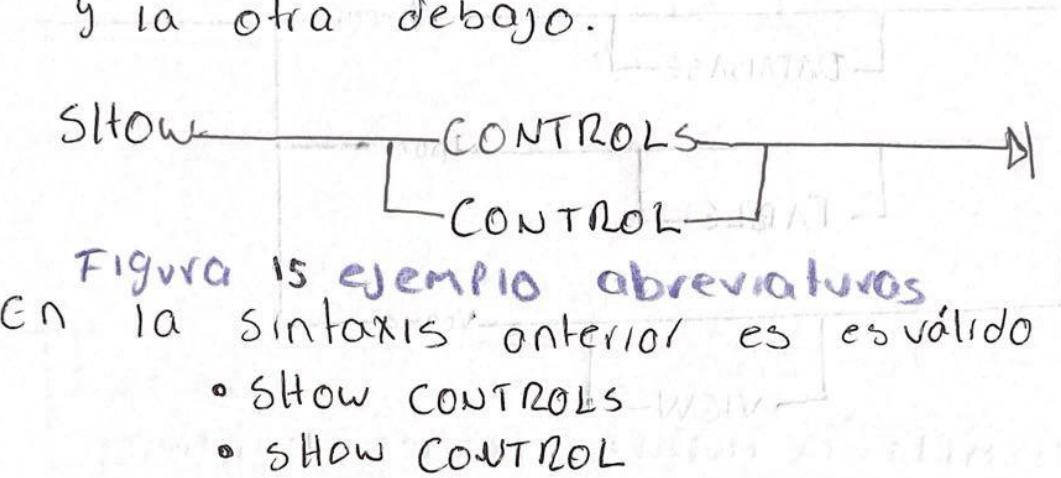


Figura 15 ejemplo abreviaturas

En la sintaxis anterior es válido

- SHOW CONTROLS
- SHOW CONTROL

## • Extractos

Cuando un fragmento de sintaxis es muy grande. Este se indica como una interrupción en la ruta de acceso y se marca con (1) a cada lado de la interrupción.

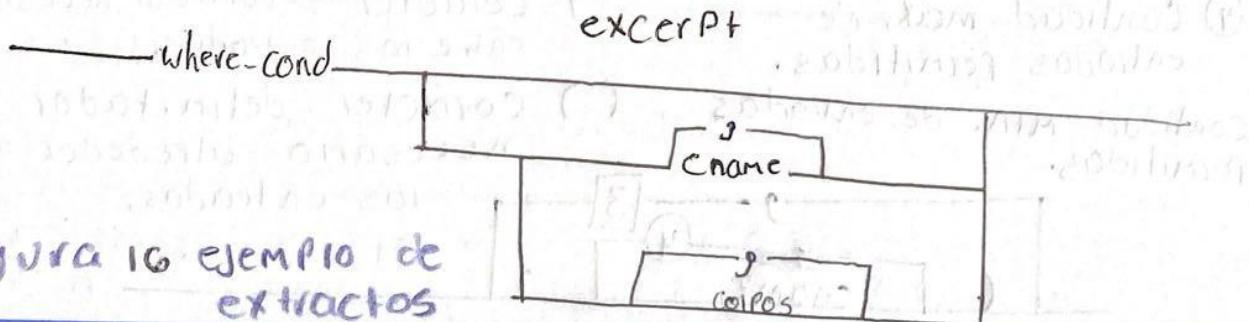
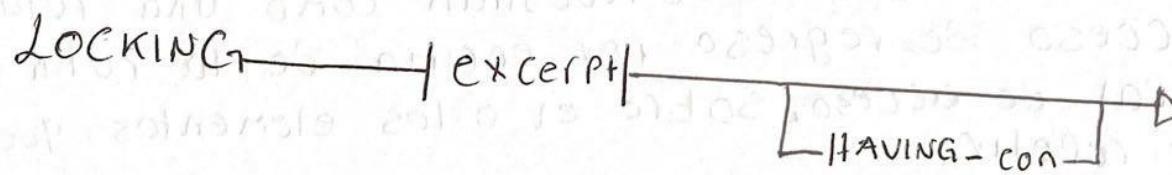


Figura 16 ejemplo de extractos

## • Múltiples frases legítimas

En un diagrama de sintaxis, cualquier número de frases puede ser legítimo.

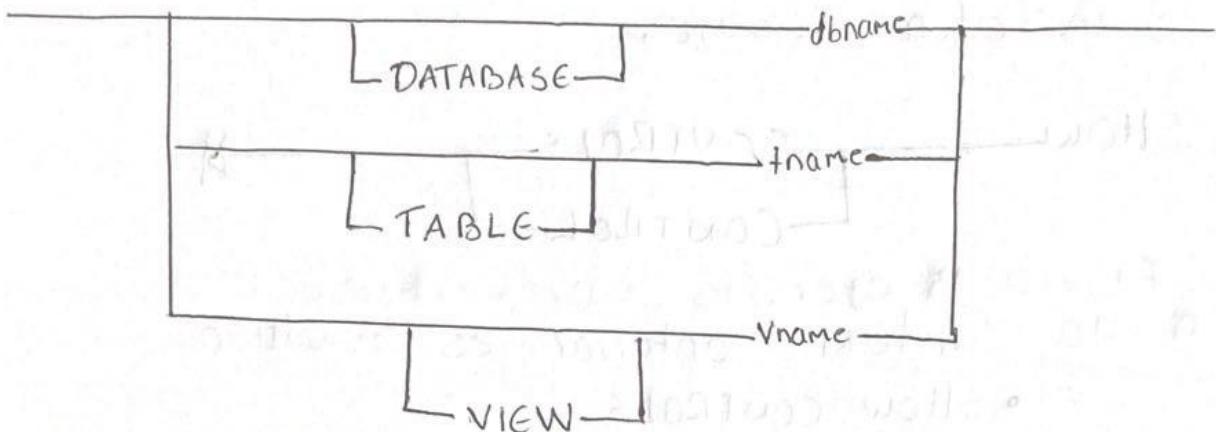


Figura 17 ejemplo de múltiples frases legítimas  
En el ejemplo anterior Cualquiera de las siguientes frases son legítimas:

- dbname
- DATABASE dbname
- TABLE fname
- VIEW Vname
- fname
- TABLE fname
- VIEW Vname

## • Bucles

Entrada o grupo de entradas que pueden repetir una o más veces. Se muestran como una ruta de acceso de regreso por encima de la ruta principal de acceso, sobre el o los elementos que puede repetir.

- ④ Cantidad máx. de entradas permitidas.
- ③ Cantidad mín. de entradas permitidas.
- ⑤ carácter separador necesario entre las entradas.
- ⑥ carácter delimitador necesario alrededor de las entradas.

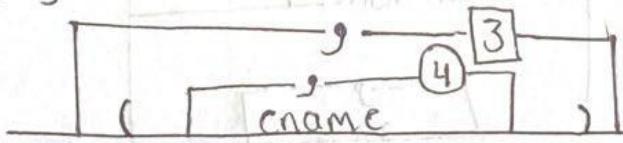


Figura 18 ejemplo de Bucles

# DIAGRAMA DE SINTAXIS DE LA CREACIÓN DE UNA VISTA EN SQL.

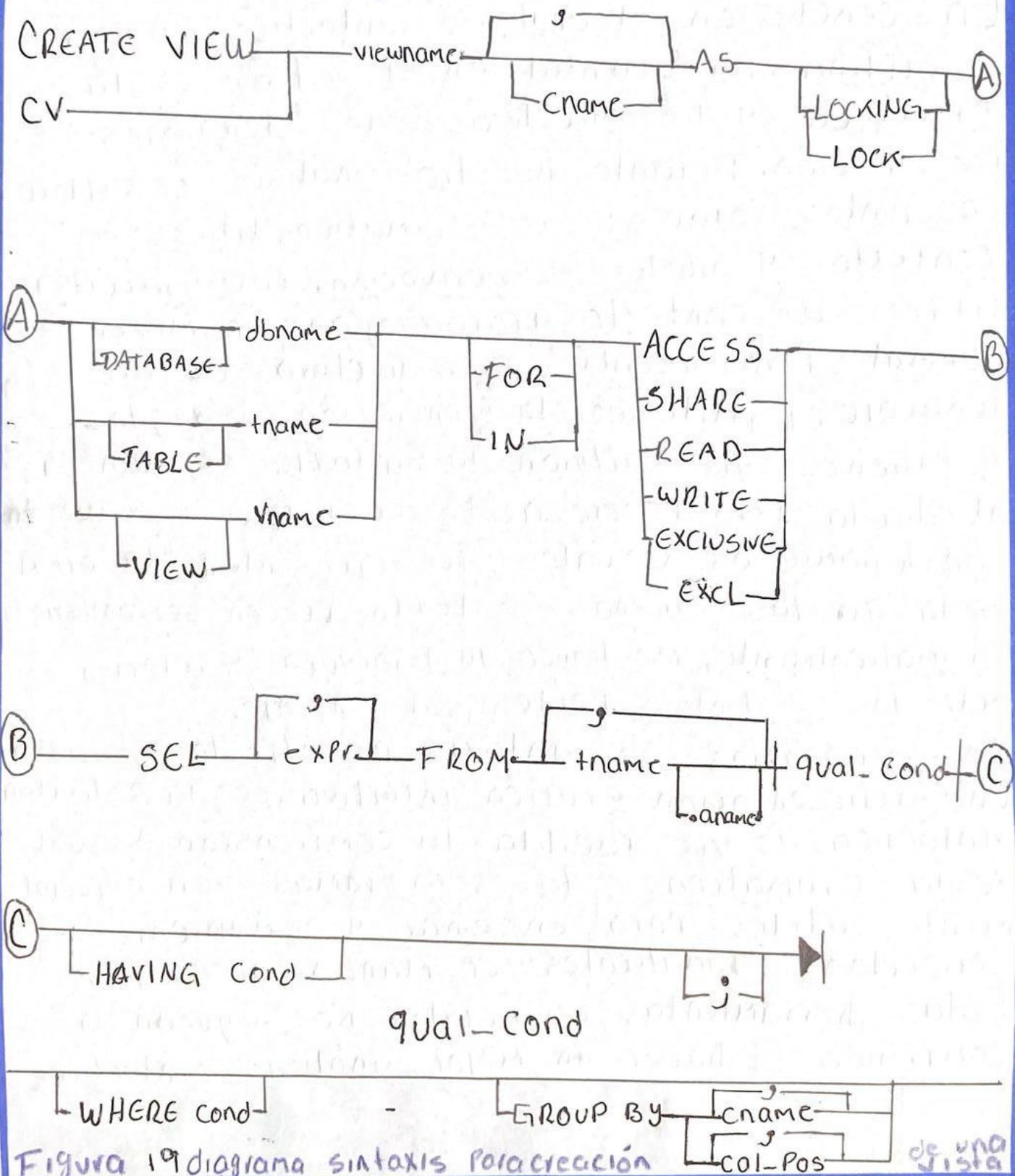


Figura 1º diagrama sintaxis para creación de vista

# CONCLUSIÓN

En conclusión, el análisis sintáctico, es una disciplina fundamental en el estudio de la gramática y la estructura de los lenguajes de programación. Durante nuestro análisis, se exploró dos puntos clave: las gramáticas libres de contexto y árboles de derivación. Las gramáticas libres de contexto proporcionan un marco formal para describir la estructura de un lenguaje, permitiendo la generación de reglas y patrones que capturan la sintaxis de manera abstracta pero poderosa. Los árboles de derivación son herramientas visuales que representan la forma en la que las cadenas de texto pueden ser analizadas gramaticalmente, mostrando la jerarquía y relación entre las distintas partes del lenguaje.

Los diagramas de sintaxis, por otro lado, ofrecen una representación gráfica intuitiva de la estructura sintáctica, lo que facilita la comprensión visual de la gramática. Estos diagramas son especialmente útiles para enseñar y comunicar conceptos gramaticales de manera accesible. Estas herramientas esenciales nos ayudan a comprender y hacer un mejor análisis sintáctico.

TECNOLÓGICO  
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II****AUTOR: EQUIPO 4**

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

PRACTICA No.	NOMBRE DE LA PRACTICA	FECHA DE ENTREGA
2	Ejercicios Prácticos de Análisis Sintáctico	27 Septiembre 2023

1	INTRODUCCION
Anteriormente en la Monografía, exploramos conceptos acerca de: las Gramáticas Libres de Contexto y todo lo que abarca para la definición de lenguajes formales, entendiendo sus características, propiedades y su impacto en la teoría de la computación y la lingüística.	
En esta práctica, a través de ejercicios prácticos, buscaremos internalizar estos conceptos y comprender cómo los árboles de derivación desempeñan un papel crucial en el análisis sintáctico.	
A su vez en esta práctica consolidaremos nuestro conocimiento, acerca del análisis sintáctico y lo que hay detrás de dicho análisis de cadenas en lenguajes formales.	

2	OBJETIVO
Entender a fondo el concepto de Gramáticas Libres de Contexto (GLC), explorando sus propiedades, reglas de producción y su papel en la descripción de lenguajes formales, con el fin de aplicar este conocimiento en la construcción de árboles de derivación y en el análisis sintáctico de cadenas en un contexto práctico.	

3	MATERIALES NECESARIOS
<ul style="list-style-type: none"><li>• 1 computadora que cumpla con los requisitos mínimos de la paquetería Office.<ol style="list-style-type: none"><li>1. Sistema operativo: Windows 2000 / 98 / XP / Vista / 7 / 8 / 10 / 11</li><li>2. Memoria (RAM): 256 MB</li><li>3. Espacio disco duro: 100 MB</li><li>4. Procesador: Intel de 750 Mhz</li></ol></li><li>• Navegador Web para consulta de información</li><li>• Libros o videos didácticos referentes al funcionamiento del Análisis Sintáctico.</li></ul>	

TECNOLÓGICO  
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II****AUTOR: EQUIPO 4**

5

**DESARROLLO****¿QUÉ SON LAS GRAMÁTICAS LIBRES DE CONTEXTO ?**

Las GLC son un tipo específico de gramática formal que se utilizan para describir la estructura sintáctica de los lenguajes formales. Una GLC se define como una cuádrupla  $G=(N,\Sigma,P,S)$ , donde:

- $N$  es un conjunto finito de símbolos no terminales.
- $\Sigma$  es un conjunto finito de símbolos terminales (disjunto de  $N$ ).
- $P$  es un conjunto finito de reglas de producción, donde cada regla tiene la forma  $A \rightarrow \alpha$ , donde  $A$  es un símbolo no terminal y  $\alpha$  es una cadena de símbolos terminales y no terminales.
- $S$  es el símbolo inicial, que pertenece a  $N$ .

**1.2. Componentes de una GLC: símbolos terminales, no terminales, reglas de producción:**

- Los símbolos terminales son aquellos que forman las cadenas en el lenguaje que describe la gramática. Por ejemplo, en una GLC que define expresiones aritméticas, los números y los operadores serían símbolos terminales.
- Los símbolos no terminales son aquellos que se utilizan en las reglas de producción para definir la estructura sintáctica de las cadenas en el lenguaje. Estos símbolos representan categorías gramaticales abstractas. Por ejemplo, en una GLC para expresiones aritméticas, "Expr" podría ser un símbolo no terminal que representa una expresión.
- Las reglas de producción establecen cómo se pueden formar las cadenas válidas en el lenguaje. Estas reglas indican cómo los símbolos no terminales pueden reemplazarse por cadenas de símbolos terminales y no terminales.

**1.3. Ejemplos de GLC simples:** Ejemplo 1: Gramática para el lenguaje de paréntesis bien balanceados.

$$S \rightarrow (S) \mid SS \mid \epsilon$$

- En esta GLC,  $S$  es el símbolo inicial.
- Los símbolos terminales son "(" y ")".
- Los símbolos no terminales son  $S$ .
- Las reglas de producción indican que una cadena puede ser una cadena vacía ( $\epsilon$ ), un paréntesis abierto seguido de una cadena  $S$  seguido de un paréntesis cerrado, o dos cadenas  $S$  concatenadas.

Las GLC son más expresivas que los lenguajes regulares, lo que significa que pueden describir lenguajes más complejos. Sin embargo, existe una relación interesante entre las GLC y los lenguajes regulares. Los lenguajes generados por GLC son subconjuntos de los lenguajes regulares. Un lenguaje es regular si y solo si puede ser generado por una gramática regular, que es un tipo más simple de gramática formal. Esto establece una jerarquía en la capacidad expresiva de las gramáticas y los lenguajes.

En resumen, sienta las bases para comprender las Gramáticas Libres de Contexto (GLC) al definirlas, explorar sus componentes clave, proporcionar ejemplos simples y discutir su relación con los lenguajes regulares.



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

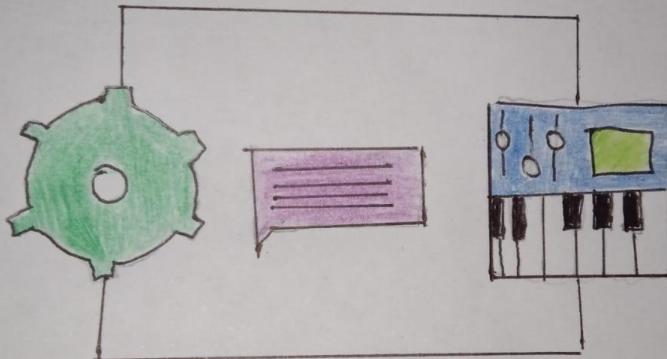
AUTOR: EQUIPO 4

### ¿QUÉ ES UN CONTEXTO?

El término "contexto" se refiere a la información que rodea a una palabra, frase o estructura lingüística en un enunciado o discurso y que influye en su significado y función gramatical. El contexto es fundamental para comprender cómo funcionan las palabras y las estructuras en una oración, ya que una misma palabra puede tener significados diferentes dependiendo de su contexto.

El contexto sintáctico se refiere específicamente a la relación de una palabra o frase con otras palabras o frases en una oración y cómo esta relación determina la función gramatical y el significado de las palabras involucradas.

Contexto



Acciones      Palabras      Cosas

### ¿SIRVE DICHO CONTEXTO A LOS PROPÓSITOS DE UNA GRAMÁTICA QUE DEFINA UN LENGUAJE FORMAL?

El contexto es un elemento fundamental para el análisis y la descripción de un lenguaje formal, ya que permite establecer las reglas sintácticas, semánticas y pragmáticas que rigen la construcción y la interpretación de las expresiones lingüísticas. Una gramática que defina un lenguaje formal debe tener en cuenta el contexto en el que se produce y se usa el lenguaje, así como los propósitos comunicativos y cognitivos de los hablantes.



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

El contexto puede ser de varios tipos: físico, social, cultural, histórico, etc., y puede influir en el significado, la función y la forma de las expresiones del lenguaje formal. Por ejemplo, el contexto puede determinar el uso de variables, constantes, símbolos, operadores, conectores, etc., así como las convenciones de escritura, lectura y razonamiento que se aplican al lenguaje formal. El contexto también puede facilitar la comprensión y la inferencia de las propiedades, los teoremas y las demostraciones que se derivan del lenguaje formal.

Por lo tanto, el contexto sirve a los propósitos de una gramática que defina un lenguaje formal, ya que le proporciona los criterios necesarios para describir y explicar el funcionamiento y la estructura del lenguaje.

En el contexto de la teoría de lenguajes formales y gramáticas formales, el contexto se refiere a la información que rodea a una cadena de símbolos y que determina su estructura y validez dentro del lenguaje definido por la gramática. El contexto puede influir en la generación y el análisis de cadenas en el lenguaje formal de diversas maneras:

**Contexto sintáctico:** En una gramática formal, como una gramática libre de contexto (GLC) o una gramática de contexto libre (GCL), el contexto sintáctico es crucial para definir las reglas de producción que generan las cadenas válidas del lenguaje. Las reglas gramaticales especifican cómo las unidades lingüísticas se combinan en función de su contexto sintáctico para formar oraciones válidas.

**Contexto semántico:** Además del contexto sintáctico, el contexto semántico puede ser importante en lenguajes formales que tienen un componente semántico, como los lenguajes de programación. Aquí, el significado de una construcción lingüística puede depender de su contexto semántico, como el valor de las variables o la interpretación de las expresiones.

**Contexto de análisis:** Cuando se realiza el análisis sintáctico de una cadena en un lenguaje formal, como en un parser de un lenguaje de programación, el contexto se utiliza para determinar la estructura sintáctica de la cadena. El análisis del contexto puede ayudar a decidir cómo se deben aplicar las reglas gramaticales y si la cadena es sintácticamente válida.

**Contexto de generación:** En la generación de cadenas en un lenguaje formal, el contexto puede influir en qué cadenas se generan y en qué orden. Por ejemplo, en un generador de código, el contexto puede determinar qué instrucciones se generan en función de la lógica del programa.

*Ajua*

*Fern*

*Dan*



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



### PRÁCTICA DE LABORATORIO LENGUAJES Y AUTÓMATAS II

AUTOR: EQUIPO 4

#### Ejercicios

- 1) Crear un lenguaje que pueda representar expresiones aritméticas simples que incluyan sumas y multiplicaciones de números enteros positivos.

La gramática sería de forma

#### Símbolos no terminales

- S (Símbolo inicial)
- E (Expresión)
- T (Término)
- F (Factor)

#### Símbolos terminales

- Números enteros positivos (por ejemplo, 0, 1, 2, 3, ...)
- + (Operador de suma)
- \* (Operador de multiplicación)
- ( (Paréntesis de apertura)
- ) (Paréntesis de cierre)

#### Reglas de producción

1.  $S \rightarrow E$
2.  $E \rightarrow E + T \mid T$
3.  $T \rightarrow T * F \mid F$
4.  $F \rightarrow (E) \mid N$

Derivar la cadena "2 \* (3 + 4)" utilizando la gramática dada.

#### Pasos:

Comenzamos con el símbolo inicial S.

Usamos la regla 1 ( $S \rightarrow E$ ) para derivar E.

Usamos la regla 3 ( $T \rightarrow T * F$ ) para derivar  $T * F$ .

Usamos la regla 3 ( $T \rightarrow F$ ) para derivar F.

Usamos la regla 4 ( $F \rightarrow (E)$ ) para derivar (E).

Usamos la regla 1 ( $S \rightarrow E$ ) para derivar E.

Usamos la regla 2 ( $E \rightarrow E + T$ ) para derivar  $E + T$ .

Usamos la regla 3 ( $T \rightarrow T * F$ ) para derivar  $T * F$ .

Usamos la regla 3 ( $T \rightarrow F$ ) para derivar F.

Usamos la regla 4 ( $F \rightarrow N$ ) para derivar N (en este caso, el número 2).

Usamos la regla 4 ( $F \rightarrow (E)$ ) para derivar (E).

Usamos la regla 1 ( $S \rightarrow E$ ) para derivar E.

Usamos la regla 2 ( $E \rightarrow T$ ) para derivar T.

Usamos la regla 3 ( $T \rightarrow T * F$ ) para derivar  $T * F$ .

Usamos la regla 4 ( $F \rightarrow N$ ) para derivar N (en este caso, el número 3).

Finalmente, usamos la regla 4 ( $F \rightarrow N$ ) para derivar N (en este caso, el número 4).



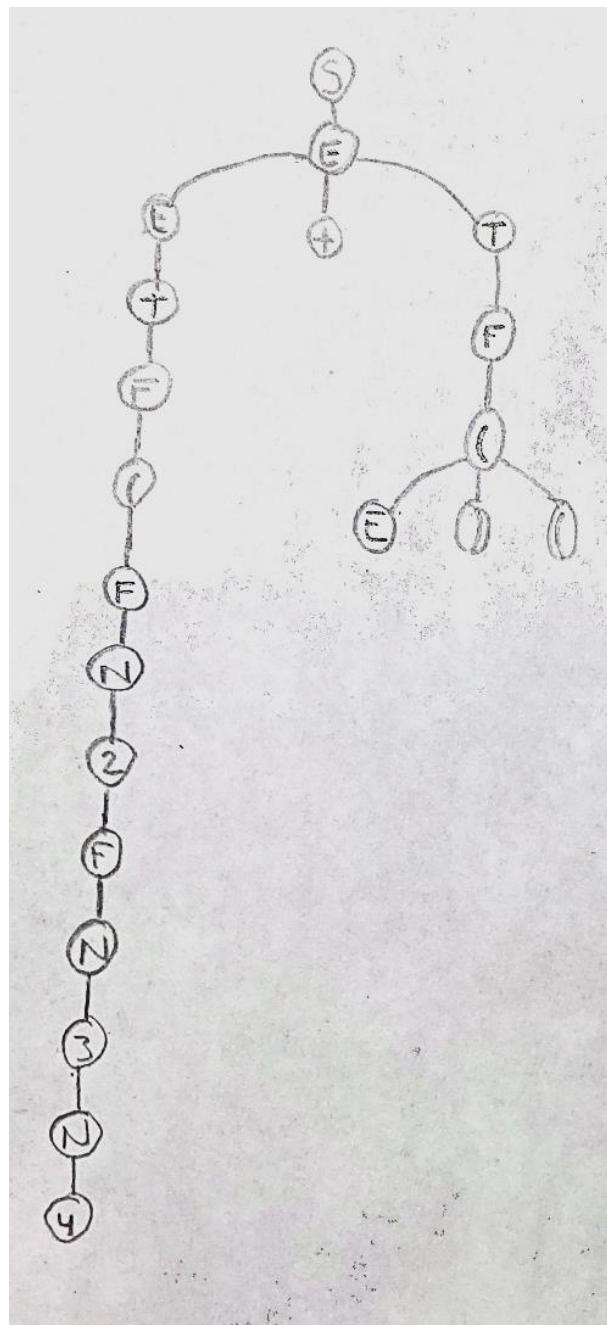
TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4



Alvaro  
Pach  
Díaz



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

- 2) Crear un lenguaje que pueda representar la declaración de una variable en java int x = 5;

Gramática para Declaraciones de Funciones

### Símbolos no terminales

Declaracion, Tipo, Identificador, Asignacion, Expresion, Literal.

### Símbolos terminales

int, x, =, 5.

### Reglas de producción

Declaracion -> Tipo

Tipo -> int | float | char | boolean | ...

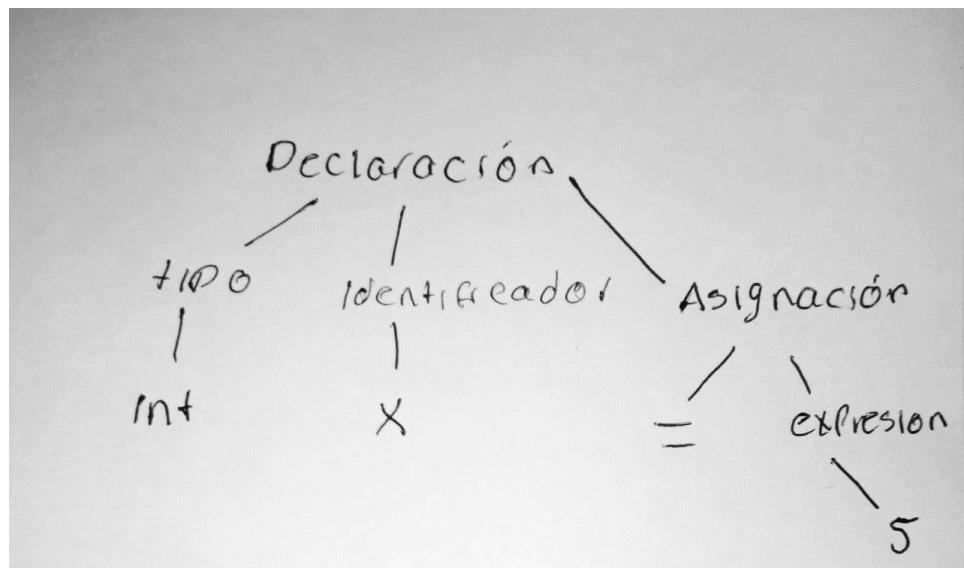
Identificador -> x | y | z | ...

Asignacion -> = Expresion

Expresion -> Literal | Identificador | Expresion Operador Expresion | ...

Literal -> 5 | 3.14 | 'a' | true | ...

Operador -> + | - | \* | / | ...





TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II

AUTOR: EQUIPO 4

- 3) Supongamos que estamos trabajando en el diseño de un lenguaje de marcado simple, similar a HTML.

La gramática sería de forma

### Símbolos no terminales

- S (Símbolo inicial)
- Etiqueta (Representa una etiqueta HTML)
- Contenido (Contenido dentro de una etiqueta)
- Texto (Texto libre)

### Símbolos terminales

- < (Símbolo de apertura de etiqueta)
- > (Símbolo de cierre de etiqueta)
- / (Símbolo de cierre de etiqueta)
- Palabras clave específicas (por ejemplo, "html", "head", "body", "p", etc.)

### Reglas de producción

1. S -> Etiqueta | Contenido
2. Etiqueta -> < PalabraClave > Contenido < / PalabraClave >
3. Contenido -> Etiqueta | Texto
4. Texto -> CualquierTexto

Ejemplo de Uso:

```
<html>
  <head>
    <title>Ejercicio 3</title>
  </head>
  <body>
    <p>¡Hola, RRG!</p>
  </body>
</html>
```

Pasos:

Comenzamos con el símbolo inicial S.

Utilizamos la regla 1 (S -> Etiqueta) para derivar Etiqueta.

Utilizamos la regla 2 (Etiqueta -> < PalabraClave > Contenido < / PalabraClave >) para derivar <html> Contenido </html>.

Dividimos el contenido en dos partes: <head> Contenido </head> y <body> Contenido </body>.

Procedemos con <head> Contenido </head>:

- Utilizamos la regla 1 (S -> Etiqueta) para derivar Etiqueta.
- Utilizamos la regla 2 (Etiqueta -> < PalabraClave > Contenido < / PalabraClave >) para derivar



TECNOLÓGICO  
NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

<head> Contenido </head>

- Dividimos el contenido en dos partes: <title> Texto </title>.
- Procedemos con <title> Ejercicio 3 </title>:
  - Utilizamos la regla 1 ( $S \rightarrow \text{Etiqueta}$ ) para derivar Etiqueta.
  - Utilizamos la regla 2 ( $\text{Etiqueta} \rightarrow <\text{PalabraClave}> \text{Contenido} </\text{PalabraClave}>$ ) para derivar <title> Ejercicio 3 </title>.
  - Dividimos el contenido en dos partes: Texto.
  - Procedemos con Texto: "L y A II".

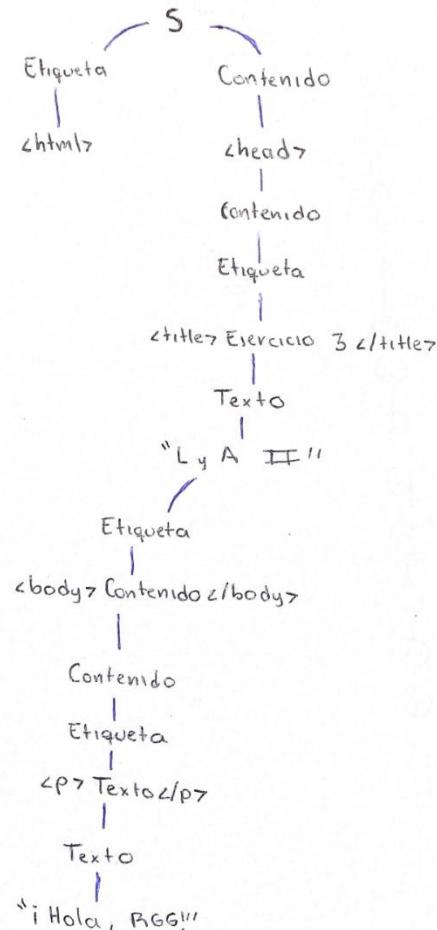
Regresamos a <head> Contenido </head> y procedemos con Contenido:

- Utilizamos la regla 3 ( $\text{Contenido} \rightarrow \text{Etiqueta}$ ) para derivar Etiqueta.
- Utilizamos la regla 2 ( $\text{Etiqueta} \rightarrow <\text{PalabraClave}> \text{Contenido} </\text{PalabraClave}>$ ) para derivar <body> Contenido </body>.
- Dividimos el contenido en dos partes: Contenido.
- Procedemos con Contenido:
  - Utilizamos la regla 3 ( $\text{Contenido} \rightarrow \text{Etiqueta}$ ) para derivar Etiqueta.
  - Utilizamos la regla 2 ( $\text{Etiqueta} \rightarrow <\text{PalabraClave}> \text{Contenido} </\text{PalabraClave}>$ ) para derivar <p> Contenido </p>.
  - Dividimos el contenido en dos partes: Texto.
  - Procedemos con Texto: ";Hola, RGG!".

*Ajua*

*Juan*

*Dan*



6

BITÁCORA DE  
INCIDENCIAS

Fecha	Problema encontrado	Solución
25/09/2023	No se conocía como hacer un árbol de derivación	Se consultaron diferentes fuentes para poder entender.

7

## OBSERVACIONES

Se observa que el contexto es una consideración crítica en la comprensión de lenguajes formales y

TECNOLÓGICO  
NACIONAL DE MÉXICO**INSTITUTO TECNOLÓGICO DE CELAYA****PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II****AUTOR: EQUIPO 4**

gramáticas, ya que determina cómo se interpretan y se estructuran las cadenas dentro de un lenguaje formal. Sin una consideración adecuada del contexto, la comprensión completa de la sintaxis sería difícil de lograr.

Observación sobre las Gramáticas Libres de Contexto: Se destaca la versatilidad y utilidad de las gramáticas libres de contexto en la descripción de la sintaxis en una amplia variedad de aplicaciones. Esto podría incluir ejemplos en la programación de computadoras, donde las GLC se utilizan para definir la estructura de un lenguaje de programación, o en la lingüística, donde se aplican para describir la estructura de las oraciones en un idioma.

Observación sobre los Diagramas de Derivación Sintáctica: Se observa que los diagramas de derivación sintáctica son herramientas visuales efectivas para representar procesos de generación y análisis en gramáticas libres de contexto. Estos diagramas simplifican la comprensión de cómo se aplican las reglas gramaticales para derivar o analizar cadenas específicas.

Observación sobre la Interdisciplinariedad: Se puede hacer una observación más general sobre cómo estos conceptos son fundamentales en campos interdisciplinarios como la lingüística computacional, donde se combinan los principios lingüísticos con la teoría de lenguajes formales para desarrollar aplicaciones como el procesamiento del lenguaje natural y la traducción automática.

**8****CONCLUSIÓN**

En resumen, el contexto desempeña un papel crucial en el estudio de lenguajes formales y gramáticas, y esta importancia se extiende de manera particular a las gramáticas libres de contexto (GLC). El contexto, ya sea en forma de información sintáctica o semántica, es esencial para determinar cómo las cadenas de símbolos se estructuran y se interpretan dentro de un lenguaje formal.

Las gramáticas libres de contexto, conocidas por su capacidad para describir la sintaxis de una amplia variedad de lenguajes, ofrecen un conjunto de reglas de producción que son independientes del contexto en el que se encuentran las partes de una cadena. Esto las hace útiles en la representación de estructuras sintácticas en una gama diversa de aplicaciones, desde la definición de lenguajes de programación hasta la descripción de la sintaxis en el análisis de texto.

Además, los diagramas de derivación sintáctica son una herramienta valiosa para visualizar y comprender el proceso de generación o análisis de cadenas en una gramática libre de contexto. Estos diagramas proporcionan una representación gráfica de cómo se aplican las reglas gramaticales para derivar una cadena específica, lo que facilita la comprensión intuitiva de la estructura sintáctica y cómo las diferentes partes de una cadena se relacionan entre sí.

En conjunto, el contexto, las gramáticas libres de contexto y los diagramas de derivación sintáctica son componentes esenciales en la formalización y comprensión de la sintaxis en lenguajes formales. Estos conceptos desempeñan un papel crítico en campos como la lingüística computacional, la teoría de compiladores y la programación de computadoras, ya que permiten definir, analizar y generar estructuras sintácticas en lenguajes formales de una manera precisa y coherente.

**9****REFERENCIAS**

- ✓ IBM documentation. (s. f.). <https://www.ibm.com/docs/es/odm/8.5.1?topic=logic-defining-context>
- ✓ Usme,N(s.f ) ¿Qué es un contexto? <https://platzi.com/clases/1928-curso-de-thick-data/29194-que-es-un-contexto/>
- ✓ www.improntadigital.com. (s. f.). *Gramática en contexto*. Edelsa.
- ✓ [https://www.edelsa.es/catalogo\\_Gramatica\\_contexto.php](https://www.edelsa.es/catalogo_Gramatica_contexto.php)



TECNOLÓGICO  
NACIONAL DE MÉXICO®

## INSTITUTO TECNOLÓGICO DE CELAYA



**PRÁCTICA DE LABORATORIO LENGUAJES Y  
AUTÓMATAS II**

**AUTOR: EQUIPO 4**

- ✓ <https://users.exa.unicen.edu.ar/catedras/ccomp1/Apunte5.pdf>
- ✓ Árboles de derivación. (n.d.). Lenguajes Formales Y Autómatas.  
[https://ivanvladimir.gitlab.io/lfy\\_a\\_book/docs/04abropar%C3%A9ntesisabropar%C3%A9ntesisiscierropar%C3%A9ntesis/05arbolesdederivaci%C3%B3n/](https://ivanvladimir.gitlab.io/lfy_a_book/docs/04abropar%C3%A9ntesisabropar%C3%A9ntesisiscierropar%C3%A9ntesis/05arbolesdederivaci%C3%B3n/)

*Ajua*

*Juan*

*Dan*

## Notación BNF

Redacción de Juan Ángel

La notación BNF es un método utilizado para definir formalmente la gramática de un lenguaje, ya sea de programación o un lenguaje marcado. Esta notación se utiliza principalmente en la especificación de lenguajes de programación y la creación de analizadores sintácticos.

La notación BNF se basa en reglas gramaticales que describen cómo deben estructurarse las sentencias o expresiones en un lenguaje. Estas reglas están así representados en un formato que incluye símbolos no terminales, símbolos terminales y símbolos especiales, como " ::= " para denotar la producción de una regla.

Ademas de que la notación BNF es un sistema utilizado para descubrir la estructura sintáctica de un lenguaje, utiliza una serie de reglas gramaticales para definir cómo se deben construir todos los sentencias o expresiones en ese lenguaje.

Los componentes clave de la notación BNF son:

1.- Símbolos no terminales (<...>): Los símbolos no terminales se colocan entre ángulos o corchetes angulares, como "expresión" y representan estructuras generales en el lenguaje y son el punto de partida para definir la gramática.

2.- Símbolos terminales: Los símbolos terminales son elementos concretos del lenguaje, como palabras clave, caracteres específicos o cadenas de texto. En BNF, se escriben directamente sin ángulos ni corchetes angulares. Por ejemplo, "+" y "-" son símbolos terminales en la expresión aritmética.

3.- Producción de reglas ( $::=$ ): Las reglas de producción se utilizan para definir como se construyen todas las estructuras del lenguaje a partir de simbolos no terminales y simbolos terminales.

### Ejemplo

$\langle \text{expresión} \rangle ::= \langle \text{término} \rangle "+" \langle \text{expresión} \rangle \mid \langle \text{término} \rangle "-" \langle \text{expresión} \rangle \mid \langle \text{término} \rangle$

$\langle \text{término} \rangle ::= \langle \text{factor} \rangle "*" \langle \text{término} \rangle \mid \langle \text{factor} \rangle "/" \langle \text{término} \rangle \mid \langle \text{factor} \rangle$   
 $\langle \text{factor} \rangle ::= "(" \langle \text{expresión} \rangle ")" \mid \langle \text{número} \rangle$

$\langle \text{número} \rangle ::= \langle \text{dígito} \rangle \mid \langle \text{dígito} \rangle \langle \text{número} \rangle$

$\langle \text{dígito} \rangle ::= "0" \mid "1" \mid "2" \mid \dots \mid "9"$

En conclusión, la notación BNF proporciona una forma sistemática de describir la estructura de un lenguaje, lo que facilita la comprensión de cómo deben construirse las expresiones o sentencias válidas en ese lenguaje.

## Notación BNF

La Notación BNF, también conocida como Backus-Naur Form, es un formalismo ampliamente utilizado en sistemas y lingüística para describir la sintaxis de lenguajes formales, como lenguajes de programación, lenguajes de marcado (HTML o XML) y gramáticas en general.

Fue desarrollada por John Backus y Peter Naur en la década de 1950 y se ha convertido en un estándar para describir gramáticas formales de una manera precisa y legible por humanos.

Utiliza reglas de producción para definir cómo se pueden construir las cadenas válidas en un lenguaje formal.

La notación utiliza símbolos especiales como " $::=$ " para indicar la relación de reemplazo.

**Ejemplo:** Gramática simple para expresiones aritméticas que involucran números enteros, operadores de suma y paréntesis

Expr ::= Term | Expr + Term

Term ::= Factor | Term \* Factor

Factor ::= (Expr) | Num

Num ::= 0 | 1 | 2 | ... | 9

Dentro y siendo así:

• Exp  
Term  
Factor  
Num } Símbolos no terminales

• +  
\*  
(  
) } Símbolos terminales

- :: = Utilizado para denotar "se deriva en"
- | Utilizado para separar las alternativas de reglas de producción.

# NOTACIÓN BNF

La notación BNF (Backus-Naur Form) es un método para describir la sintaxis de los lenguajes de programación, es decir, las reglas que determinan cómo se construyen las expresiones válidas en esos lenguajes. Se basa en símbolos terminales (que representan los elementos básicos del lenguaje, como palabras reservadas, operadores, identificadores, etc.). Los símbolos no terminales se definen mediante reglas de producción, que indican cómo se pueden formar a partir de otros símbolos.

Por ejemplo una regla de producción podría ser:

$$\langle \text{expresión} \rangle ::= \langle \text{expresión} \rangle + \langle \text{expresión} \rangle \mid \langle \text{expresión} \rangle - \langle \text{expresión} \rangle \mid \text{numero}$$

La regla indica que la expresión puede ser la suma o resta de las expresiones. Los símbolos entre paréntesis angulares son no terminales y los demás terminales. El símbolo " $\mid$ " significa " $\cup$ ".

Esta notación se utiliza para gramáticas libres de contexto. Se aplica donde quiera que se necesiten descripciones exactas de los lenguajes. Por ejemplo: En las especificaciones oficiales del lenguaje, en manuales y en libros de texto sobre teoría de lenguajes de programación.

### Metasímbolos:

:: = se define como

| OR

{ } repetición

[ ] opcional

• Los terminales entre comillas y negrita.

'1F', '5'

### Ejemplo:

<numero> :: = <entPos>  
| <entPos> { } | <entPos>

<entPos> :: = <digito>  
| <digito> <entPos>

<digito> :: = '0' | '1' | '2' | '3' | '4' | '5' | '6'  
| '7' | '8' | '9'

## Bibliografía

- ✓ [Abadi 96] M. Abadi, L. Cardelli, A Theory of Objects. Monographs in Computer Science, Springer-Verlag, ISBN: 0-387-94775-2, 1996IBURG: : su aplicación para la generación de generadores de código en un proyecto
- ✓ [Abelson 85] H. Abelson, G. J. Sussman, J. Sussman, Structure and Interpretation of Computer Programs. MIT Press, 1985
- ✓ [Agesen93] O. Agesen and J. Palsberg and M. I. Schwartzbach, Type inference of SELF: analysis of objects with dynamic and multiple inheritance, European Conference on Object Oriented Programming, Springer-Verlag, 1993
- ✓ [Aho 85] A. V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques and Tools. Addison-Wesley, ISBN: 0-201-10088-6, 1985
- ✓ [Aït Kaci 91] H. Aït Kaci, Warren's Abstract Machine, a tutorial reconstruction. MIT Press, 1991
- ✓ [Anderson 94] Andersen, P. H. "Partial Evaluation applied to ray tracing". Res. Report, DIKU. Dept. of Computer Science, Univ. of Copenhagen
- ✓ [Appel 97] A. W. Appel, Modern Compiler Implementation in ML. Cambridge University Press, ISBN: 0-521-58775-1, 1997
- ✓ [Appel 98] A. W. Appel, Modern Compiler Implementation in Java. Cambridge University Press, ISBN: 0-521-58388-8, 1998
- ✓ [Beskers97] C. Beskers, S. Feiner. "Generating Efficient Virtual Worlds for Visualization Using Partial Evaluation and Dynamic Compilation". ACM SIGPLAN Conference on Partial Evaluation and Semantics-based Program Manipulation.
- ✓ [Bracha, 98] G. Bracha, M. Odersky, D. Stoutamire, P. Wadler. Making the future safe for the past: Adding Genericity to the Java programming language, Object-Oriented Programming Systems, Languages and Applications (OOPSLA), 1998
- ✓ [Cardelli, 85] L. Cardelli, P. Wegner, On Understanding Types, Data Abstraction, and Polymorphism, Computing Surveys, vol 17, number 4, 1985
- ✓ [Cardelli, 97] L. Cardelli, Type Systems. Handbook of Computer Science and Engineering, Ch. 103, CRC Press 1997
- ✓ "En el libro 'Intérpretes y Diseño de Lenguajes de Programación' de Labra Gayo, J.E., Cueva Lovelle, J.M., Izquierdo Castanedo, R., Juan Fuente, A.A., Luengo Díez, M.C., y Ortín Soler, F., publicado por la editorial Servitec en el año [2003], se abordan temas relacionados con el diseño de lenguajes de programación" (Labra Gayo et al., [2003]).
- ✓ de Autómatas y lenguajes formales Federico Simmross Wattenberg, T. (s/f). El generador de analizadores léxicos lex. Uva.es. Recuperado el 27 de septiembre de 2023, de <https://www.infor.uva.es/~mluisa/talf/docs/lab0/L3.pdf>
- ✓ (S/f). Edu.ar. Recuperado el 27 de septiembre de 2023, de <https://dc.exa.unrc.edu.ar/staff/fbavera/papers/TesisJTLex-Bavera-Nordio-02.pdf>
- ✓ (S/f-b). Unizar.es. Recuperado el 27 de septiembre de 2023, de <https://webdiis.unizar.es/~ezpeleta/lib/exe/fetch.php?media=misdatos:compi:2bis.introflex.pdf>
- ✓ Lexico, A. (s/f). II26 Procesadores de lenguaje. Uji.es. Recuperado el 27 de septiembre de 2023, de <https://repositori.uji.es/xmlui/bitstream/handle/10234/5877/lexico.apun.pdf?sequence=1>
- ✓ Generadores de analizadores Léxicos. (2017, mayo 15). Lenguajes y automatas 1 unidades: <https://lenguajesyautomatasblog.wordpress.com/2017/05/15/generadores-de-analizadores-lexicos/>
- ✓ LibreOffice. (s. f.). How to Read Syntax Diagrams and Statements recuperado el 26 de septiembre de 2023 <https://help.libreoffice.org/latest/es/text/sbasic/shared/conventions.html?DbPAR=BASIC>

- ✓ eraData Online Documentation | Quick access to technical manuals. (s. f.). Convenciones de los diagramas de sintaxis. recuperado el 26 de septiembre de 2023  
[https://docs.teradata.com/r/NKEHw\\_aG4HmBtnGH~34GVw/j5WIbS759qyWbR4bXvwpCA](https://docs.teradata.com/r/NKEHw_aG4HmBtnGH~34GVw/j5WIbS759qyWbR4bXvwpCA)
- ✓ ¿Qué es la forma normal de Backus (BNF)? - Definición de Techopedia - Desarrollo 2023. (2023). Icy Science. recuperado el 26 de septiembre de 2023 <https://es.theastrologypage.com/backus-normal-form>
- ✓ BM documentation. (s. f.-b).Cómo leer los diagramas de sintaxis recuperado el 26 de septiembre de 2023  
<https://www.ibm.com/docs/es/i/7.3?topic=programming-how-read-syntax-diagrams>

## Links a Videos

### Análisis Léxico

<https://drive.google.com/file/d/1Aq0pgeiJPubExSxdom1-VaOJIzjTM31L/view?usp=sharing>

### Análisis Sintáctico

<https://drive.google.com/file/d/1ctBc7KSt9XycwchrLfZieFhvdp5dNfGV/view?usp=sharing>