



# INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA.

**Materia:** Lenguajes y Autómatas II

**Profesor:** ISC. Ricardo González González

**Alumnos:**

Isaac Salvador Bravo Estrada 2003048

Guillermo Peasland Aguilar 20030737

Maria del Carmen Chávez Patiño 20030296

Luis Fernando Mendoza Javalera 1930536

## ACTIVIDAD ► 4 ◄

**Fecha de Entrega:** 17 de Septiembre  
de 2024.

EQUIPO NO. 3



## DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **SOLICITUD DE ACTIVIDADES**

Celaya, Guanajuato, 10 / septiembre / 2024

### LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ  
**SEMESTRE AGOSTO-DICIEMBRE 2024**

**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
- APPLICACIONES ( 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.





**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, SUPONDRÍA 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	: AI-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUITIVO 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 PERÍODO SEMESTRAL EN CURSO: <b>AGO-DIC</b>

**EJEMPLO :**

SI EL TRABAJO SE SOLICITÓ EN EQUIPO.

2024-09-10\_TNM\_CELAYA\_LAI-A\_A4\_EQUIPO\_99\_9999999\_PEREZ\_PEREZ\_JUAN\_AGO-DIC24.PDF

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

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2023-09-10\_TNM\_CELAYA\_LAI-A\_A4\_9999999\_PEREZ\_PEREZ\_JUAN\_AGO-DIC24.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.





# INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA.

Materia: Lenguajes y Automatas II

Profesor: I.S.C. Ricardo González González

## Alumnos:

Isaac Salvador Bravo Estrada 2003048

Guillermo Peasland Aguilar 20030737

Maria del Carmen Chávez Patiño 20030296

Luis Fernando Mendoza Javalera 1930536

## MONOGRAFIA TÉCNICA SOBRE GENERADORES DE ANALIZADORES LÉXICOS Y APLICACIONES (CASO DE ESTUDIO)

Fecha de Entrega: 17 de Septiembre de  
2024

EQUIPO NO. 3



## Índice:

- Introducción
- Definición y propósito
- Funciones del analizador léxico
- El analizador léxico es la primera fase de un compilador
- Funciones secundarias
- Componentes léxicos y patrones y lexemas:
- Creación de tabla de tokens
- Errores léxicos
- Generadores de analizadores Léxicos:
- Aplicaciones (caso de estudio)
- ¿Qué es MetaComp
- Caso de estudio
- Pasos utilizando MetaComp
- Ejemplo de uso
- Aplicaciones Reales
- Conclusión
- Referencias

pág	1
pág	1
pág	2
pág	3
pág	4
pág	4
pág	5
pág	7
pág	9
pág	10
pág	11
pág	11

## Índice de Figuras y tablas:

- Figura 1. Proceso General
- Figura 2. Funciones Secundarias
- Figura 3. Componentes y patrones
- Tabla 1. Ejemplo de Tokens
- Figura 4. Proceso de Lex
- Figura 5. Proceso de Analizadores
- Figura 6. Secciones de Lex
- Figura 7. Proceso de Flex
- Figura 8. Proceso de JFlexx
- Figura 9. Esquema analizador sintáctico
- Tabla 2. Tabla Comparativa

pág	1
pág	3
pág	3
pág	5
pág	5
pág	6
pág	7
pág	7
pág	8
pág	8
pág	9



# Monografía sobre Generadores de Analizadores Léxicos.

## Introducción:

Un generador de analizadores léxicos es una herramienta utilizada para automatizar la creación de programas que se encargan de descomponer un texto en sus componentes más básicos, conocidos como TOKENS. Estos TOKENS representan las unidades mínimas de significado en un lenguaje, como palabras clave, operadores o identificadores. Los generadores toman como entrada reglas basadas en expresiones regulares y producen un código que realiza el análisis léxico de forma automática, facilitando el desarrollo de compiladores e intérpretes.

## Definición y Propósito:

Un analizador léxico es un módulo destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena que se va a analizar, reconocer subcadenas que correspondan a símbolos del lenguaje y así retornar los TOKENS correspondientes y a sus atributos. Ya que escribir analizadores léxicos eficientes "a mano" puede resultar una tarea tediosa y bastante complicada, para evitarlo se han creado herramientas de software, así que los generadores de analizadores léxicos que generan automáticamente un analizador léxico a partir de una especificación provista por el usuario.

Todas estas herramientas para generar analizadores léxicos permiten definir la sintaxis de los símbolos mediante expresiones regulares, mientras que sus atributos deben ser calculados luego del reconocimiento de una subcadena que pueda constituir un símbolo del lenguaje. De esta forma se puede reducir el riesgo de errores humanos y se optimiza el proceso de análisis del código fuente, sirviendo como un primer paso que es esencial antes del análisis sintáctico y semántico.



## Funciones del analizador léxico:

Un analizador léxico aisla el analizador sintáctico de la representación de lexemas de los componentes léxicos.

Funciones:

- 1: Eliminación de espacios en blanco.
- 2: Reconocimiento de identificadores y palabras clave.
- 3: Analizador léxico "Scanner" lee la secuencia de los caracteres del programa fuente, los agrupa para formar unidades con significado propio.

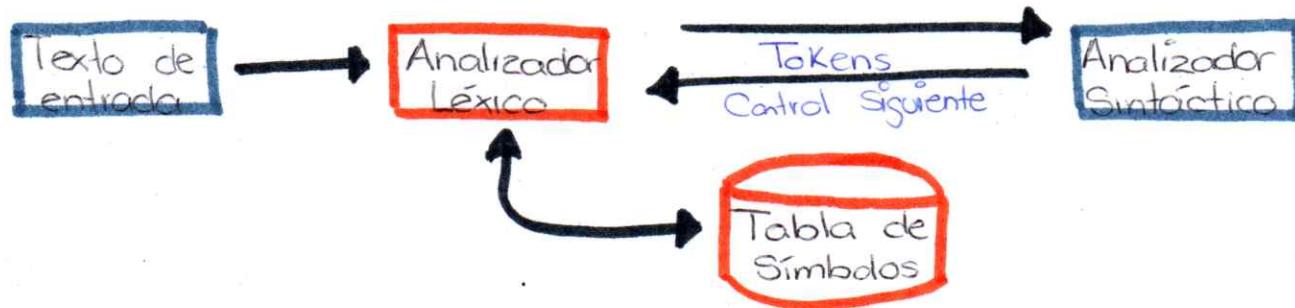


Figura 1. Proceso General.

## El analizador léxico es la primera fase de un compilador:

Esta interacción, suele aplicarse convirtiendo el analizador léxico en una subrutina o corrotina del analizador sintáctico. Recibida la orden "obtén el siguiente componente léxico" del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

## Funciones secundarias:

Ciertas funciones secundarias en la interfaz del usuario, como eliminar del programa fuente comentarios y espacios en blanco en forma de caracteres, de espacio en blanco, caracteres TAB y de linea nueva. Otra función es la de relacionar los mensajes de error del compilador con el programa fuente. Por ejemplo, el analizador léxico puede tener localizado el número de caracteres de nueva línea detectados, de modo que se pueda asociar un número de línea con un mensaje de error. En algunos compiladores, el analizador léxico se encarga de hacer una copia del programa fuente en el que están marcados los mensajes de error. Si el lenguaje fuente es la base de algunas funciones



de pre-procesamiento de macros, entonces esas funciones del procesador también se pueden aplicar al hacer el análisis léxico.

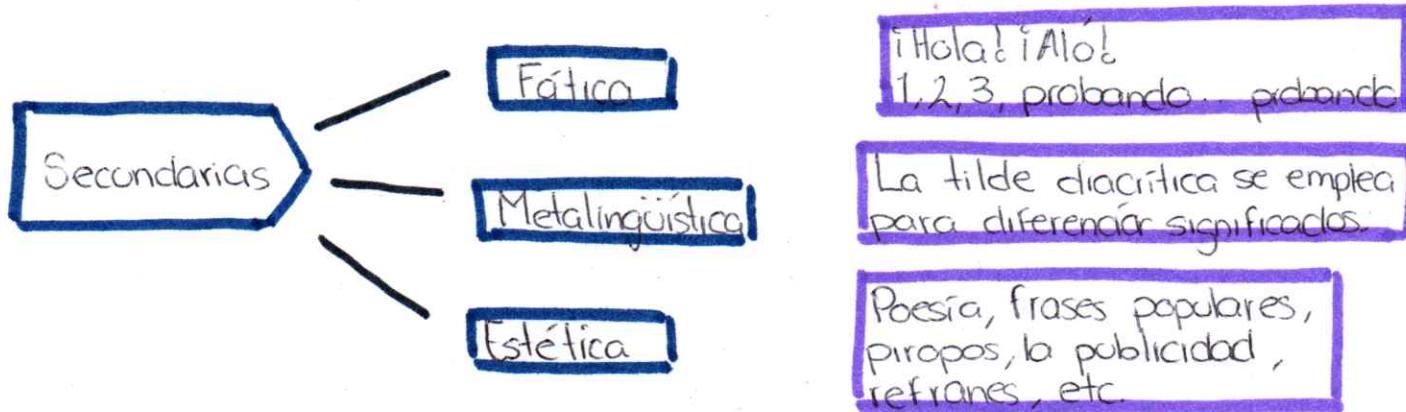


Figura 2. Funciones Secundarias.

## Componentes léxicos patrones y lexemas:

Léxico: Son las unidades lógicas que genera el analizador léxico. Formar caracteres en tokens es muy parecido a formar palabras en un lenguaje natural.

Es el conjunto de cadenas entrada que produce como salida el mismo componente léxico. Cada token es una secuencia de caracteres que representa una unidad de información en el programa fuente.

Los componentes léxicos más comunes son los siguientes:

• Palabras clave o reservadas.

(Operadores aritméticos, Operadores relacionales, Operadores lógicos, Operador de asignación, Identificadores, Constantes, Cadenas, Literales, Signos de Puntuación, Librerías).

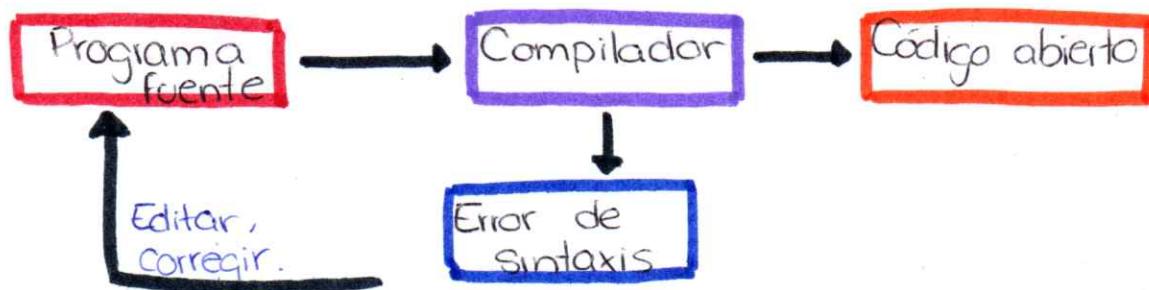


Figura 3. Componentes y Patrones.

Patrón: Es una regla que describe el conjunto de lexemas que pueden representar a un determinado componente léxico en los programas fuente. En otras palabras, es la descripción del componente léxico mediante una regla.

Lexema: Representan cadenas de caracteres en el programa fuente que se pueden tratar juntas como una unidad léxica. Un lexema es una secuencia.



de caracteres en el programa fuente con la que concuerda el patrón para un componente léxico.

## Creación de la tabla de Tokens.

**Tabla:** Es el conjunto de pares clave - valor, llamados elementos de la tabla. La tabla de símbolos es una especie de componentes que es necesaria de un compilador. Al poder declarar un identificador (normalmente una sola vez), éste siendo insertado en la tabla. Cada vez que se utilice el identificador se realizará una búsqueda en la tabla para poder obtener la información asociada (el valor). Esto se segmenta en lo siguiente:

- **Búsqueda:** dada la clave de un elemento, encontrar su valor.
- **Inserción:** Dado un par clave - valor, añadir un nuevo elemento a la tabla.
- **Cambio de valor:** Buscar el elemento y cambiar su valor.
- **Borrado:** Eliminar un elemento de la tabla.
- **Longitud de búsqueda (o tiempo de acceso):**

De una clave:  $L_i$  = número de comparaciones con elementos de la tabla para encontrar esa clave. **Máxima:**  $L_M$  = número máximo de comparaciones para encontrar cualquier clave. **Media (esperada):**  $L_m$  = número medio de comparaciones para encontrar un valor.

Si la frecuencia de todas las claves es la misma:  $L_m = (S L_i) / N$

Si la frecuencia de todas las claves no es la misma:  $L_m = \sum p_i L_i$

**Grado de ocupación:**  $S = n / N$  donde  $n$  = número de elementos en la tabla,  $N$  = capacidad máxima de la tabla.

**Función de búsqueda:**  $B: K \rightarrow E$  asocia a cada clave  $k$  un elemento  $B(k)$ .

**Valor asociado a una clave  $K: v(B(K))$ .** Puede ser múltiple, en cuyo caso normalmente se convierte en un puntero. Si está en la tabla puede almacenarse consecutivamente o en subtablas paralelas. **Tablas de Símbolos** La clave es el **identificador**: El valor está formado por:

**Atributos del identificador:** Puntero a la posición de memoria asignada. La clave sustituirse por un puntero. Los identificadores pueden estar empaquetados. La longitud del identificador puede especificarse en la tabla o delante del nombre, o ser implícita.

**Tablas consecutivas:** Todos los elementos ocupan posiciones de memoria adyacentes. **Tablas ligadas:** cada elemento apunta al siguiente. **Tablas doblemente ligadas:** cada elemento apunta al siguiente y al anterior.

**Tablas no ordenadas Inserción:** en el primer lugar vacío.

◦ **Componentes léxicos (Tokens):** Unidad mínima de información que significa algo a la hora de compilar; concepto de palabra; las fases de un lenguaje constan de cadenas de componentes léxicos.



**Lexema:** Una secuencia de caracteres de entrada que comprenden lo que es un solo componente léxico se hace llamar Lexema. Ya que una cadena de caracteres que extrae el componente abstracto del componente Léxico.

**Patrón:** Descripción de la forma que han de tomar los Lexemas para ajustarse a un componente Léxico.

TOKEN	Descripción Formal	Lexemas de Ej.
if	caracteres i, f	if
else	caracteres e, l, s, e.	else
comparación	$<0>$ O $\leq 0>$ O $\geq 0>$ O $\neq =$	$\leq$ , $\neq$
id	letra seguida por letras y dígitos	Pi, puntuación, D2
número	cualquier constante numérica	3.14159, 0, 6.02e23
literal	cualquier cosa, excepto " rodeada , por ""	"core dumped"

Tabla 1. Ejemplos de Tokens.

## Errores Léxicos:

El análisis léxico constituye la primera fase, ya que se lee de izquierda a derecha y se agrupa en componentes léxicos (tokens) como ya se mencionó, son como "consecuencias" con un significado. Además, todos los espacios en blanco, líneas, comentarios y demás información innecesaria se puede eliminar del programa fuente. También se comprueba que los símbolos del lenguaje (palabras clave, operadores,...) se han escrito correctamente.

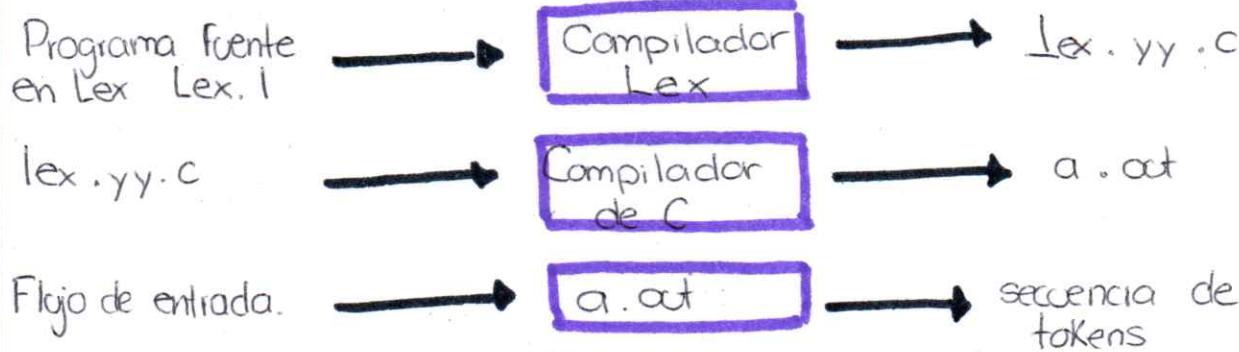


Figura 4. Proceso de Lex.



Como la tarea que realiza el analizador léxico es un caso especial de coincidencia de patrones, se necesitan los métodos de especificación y reconocimiento de patrones, y estos métodos son principalmente las expresiones regulares y los **autómatas finitos**. Sin embargo, en analizador léxico también es la parte del traductor que maneja la entrada del código fuente, y puesto que esta entrada a menudo induce un importante gasto de tiempo, el analizador léxico debe funcionar de manera tan eficiente como sea posible.

Son pocos los errores simplemente en el nivel léxico ya que tiene una visión muy restringida de un programa fuente. El analizador léxico debe devolver el componente Léxico de un identificador y dejar otra fase se ocupe de los errores.

Supongamos que en una situación en la cual el analizador léxico no podría continuar ya que ninguno de los patrones concuerda con un prefijo de la entrada.

El compilador tiene que:

1. Reportar clara y exactamente la presencia de errores.
2. Recuperarse de cada error lo suficientemente rápido para poder detectar errores subsiguientes:
  - Tratar de evitar mensajes falsos de error
  - Un error que produce un token erroneo
  - Errores léxicos posibles.

Un token o componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.

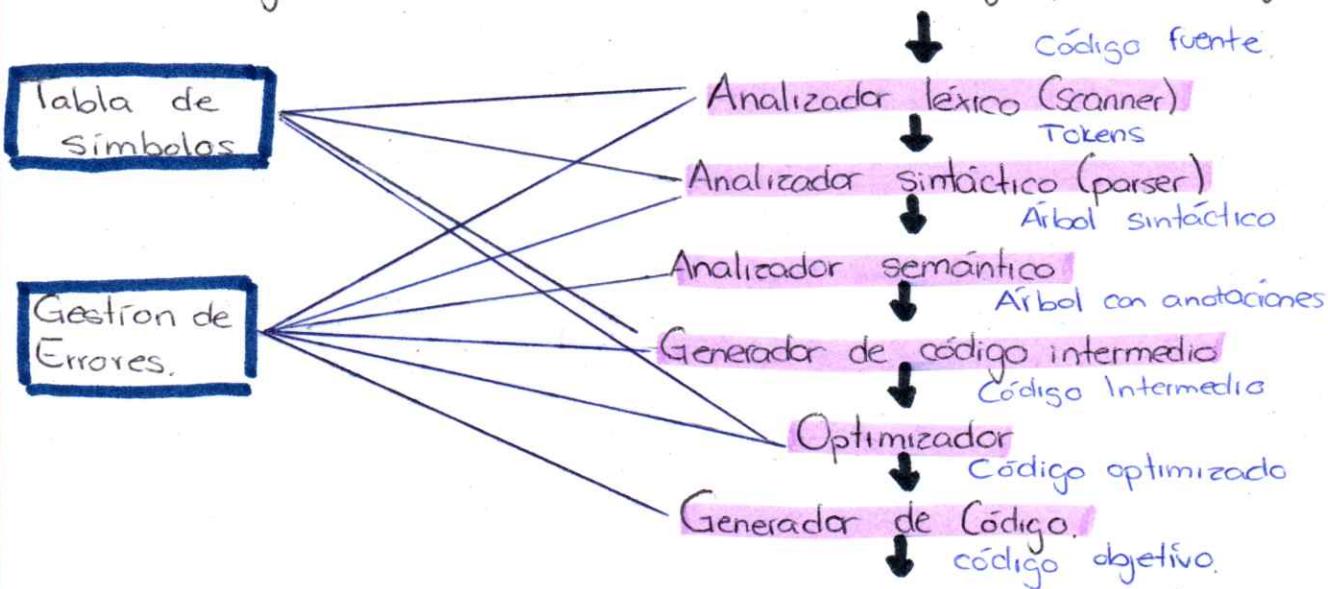


Figura 5. Proceso de Analizadores.



## Generadores de analizadores Léxicos:

Un analizador léxico es un módulo destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar, reconocer subcadenas que corresponden a símbolos del lenguaje y retornar los tokens correspondientes y sus atributos.

**Generador Lex:** Es un programa que es usado para generar analizadores léxicos; su utiliza comúnmente con el programa yacc que se utiliza para generar análisis sintáctico, este fue escrito por Eric Schmidt y Mike Lesk, es el analizador léxico estándar de POSIX. Lex toma como entrada una especificación de analizador léxico y devuelve como salida el código fuente implementando el analizador léxico en C.

Sección de declaraciones

Sección de reglas

Sección de código en C

Figura 6. Secciones de Lex

De estas tres secciones, solo la segunda es obligatoria, y cualquiera de ellas puede estar vacía

**Generador Flex:** Es una herramienta para la generación de programas que realizan concordancia de patrones en texto, es una herramienta para generar escaneos. FLEX lee los archivos de entrada dadas, o la entrada estándar si no se ha indicado ningún nombre de archivo, con la descripción de un escáner general. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas.

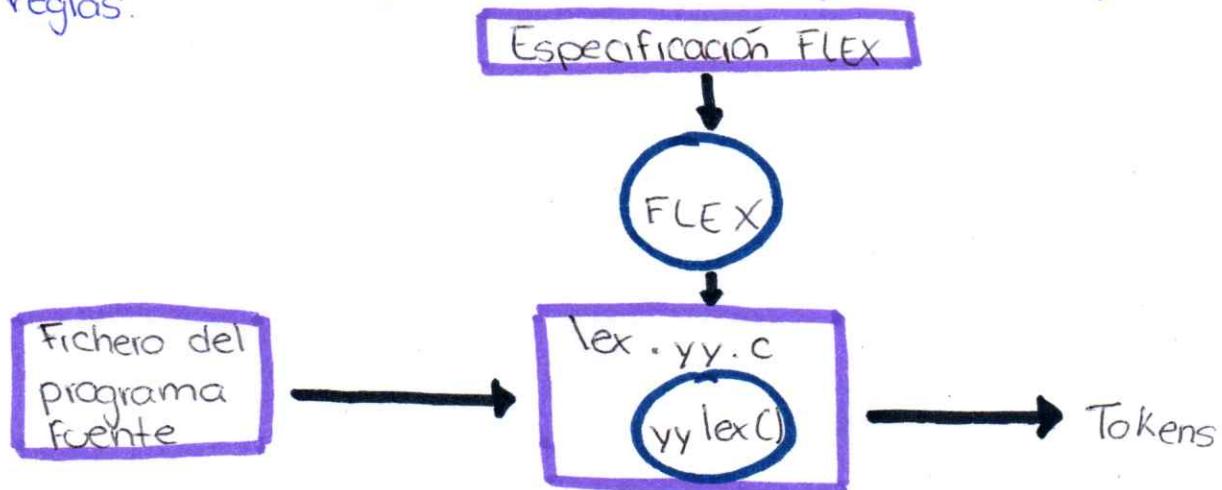


Figura 7. Proceso de Flex



**Generador JTLeXX:** Permite expresar conjuntamente sintaxis y semántica al estilo de los esquemas de traducción. A su vez el proceso de conjunto de atributos es implementado por JTLeXX por un **autómata finito** traductor con las ventajas de eficiencia que esto supone. Una especificación JTLeXX permite no solo asociar un procedimiento, o acción a cada expresión regular, si no también a cada ocurrencia de un símbolo dentro de la expresión.

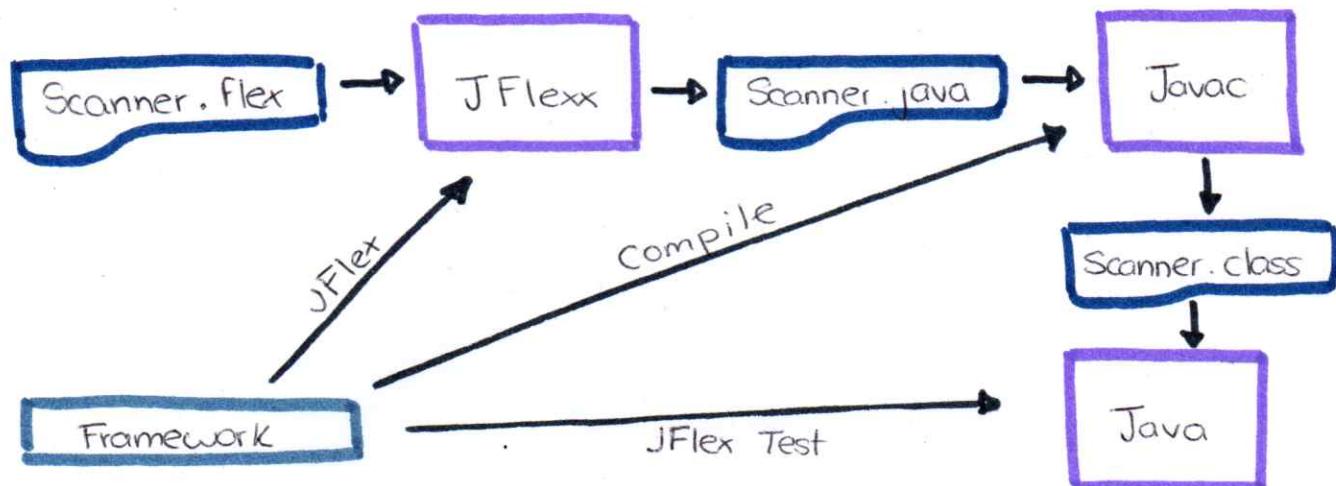


Figura 8. Proceso de JFlexx

**GNU Bison:** Es un programa generador de analizadores sintácticos que utiliza una notación conocida como BNF (Backus-Naur Form) para describir la gramática de un lenguaje. A partir de esta descripción, Bison puede generar un programa C que puede analizar texto y determinar si es sintáticamente correcto.

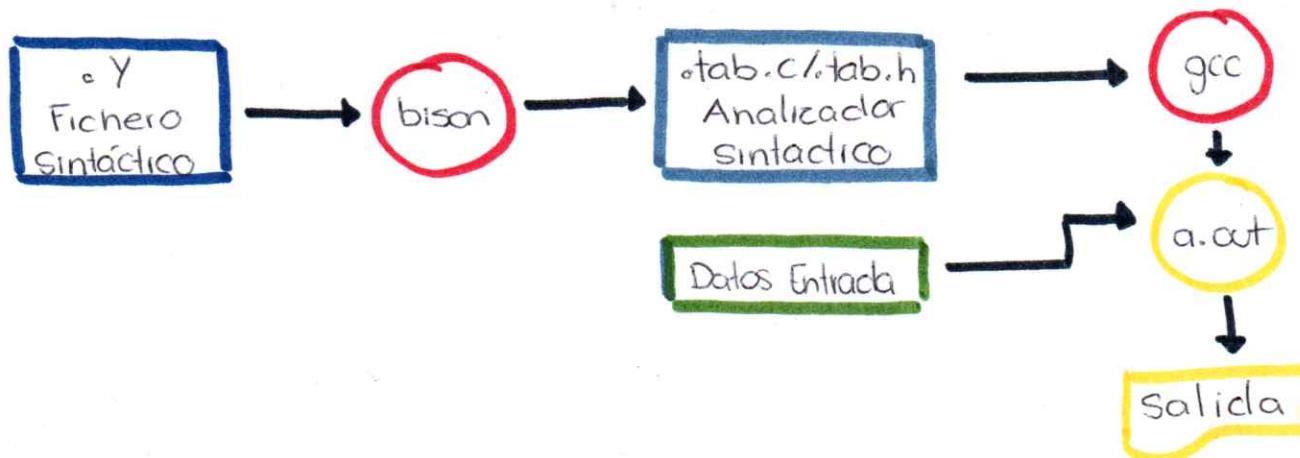


Figura 9. Esquema analizador sintáctico.



Característica	Lex	Flex	Bison
Descripción	Generador de analizadores léxicos.	Versión extendida y más moderna de Lex	Generador de analizadores sintácticos
Propósito	Divide el código fuente en tokens (palabras clave, identificadores, etc.)	Lo mismo que Lex, pero con más funcionalidades y mejoras	Construye un árbol de análisis sintáctico a partir de los tokens.
Lenguaje de entrada.	Especificaciones de expresiones regulares para definir los tokens.	Similar a Lex, pero con extensiones.	Gramática formal (generalmente BNF) para descubrir la estructura del lenguaje.
Lenguaje de salida	Código en C (principalmente) que implementa el analizador léxico.	Código en C o C++ que implementa el analizador léxico.	Código en C, C++ o Java que implementa el analizador sintáctico.
Características adicionales.		Soporte para C++, generación de código más eficiente, mejores mensajes de error.	Soporte para gramáticas ambiguas, acciones semánticas, generación de código intermedio.
Estado	Obsoleto, reemplazado por Flex	Activamente desarrollado y mantenido.	Activamente desarrollado y mantenido.
Licencia		GNU General Public License (GPL)	GNU General Public License (GPL).

Tabla 2. Tabla Comparativa

## Aplicaciones (caso de estudio)

Para construir compiladores e intérpretes, los analizadores léxicos se pueden emplear para muchos programas "convencionales". Los ejemplos más claros son aquellos programas que tienen algún tipo de entrada de texto donde hay un formato razonablemente libre en cuanto espacios y comentarios. En estos casos es bastante engoroso controlar dónde



empieza y termina cada componente, y es fácil lidiarse con los índices de los caracteres. Un analizador léxico simplifica notablemente la interfaz y si se dispone de un generador automático, el problema se resuelve en varias líneas de código.

## ¿Qué es Metacomp?

Metacomp es un metacompilador, una herramienta que genera analizadores léxicos en Python. Esto significa que a partir de una descripción de un lenguaje, Metacomp puede construir automáticamente un programa capaz de reconocer y clasificar los componentes básicos (tokens) de ese lenguaje.

**Caso de estudio:** Creación de un lenguaje de programación simple.

Imaginemos que queremos crear un lenguaje de programación sencillo para realizar cálculos matemáticos básicos. Este lenguaje podría tener una sintaxis así:

expresión = número operador número  
número = dígito +  
dígito = 0 | 1 | 2 | ... | 9  
Operador = + | - | \* | /

## Pasos utilizando Metacomp:

1- Definición del lenguaje: Utilizaremos la sintaxis regular para poder describir los tokens del lenguaje, como antes se mostró.

2- Generación de Analizador Léxico: Introduciríamos esta descripción a MetaComp. MetaComp generaría automáticamente un analizador léxico en Python.

3- Integración de un Compilador: El analizador léxico generado sería la primera etapa de nuestro compilador. Este analizador leería el código fuente y produciría una secuencia de tokens.

4- Análisis Sintáctico y Semántico: Etapas posteriores del analizador comprobarían la estructura gramatical (sintaxis) y el significado (semántica) de los tokens.

## Ejemplo de uso:

# Código generado por MetaComp para el analizador léxico  
def Analizador\_léxico(entrada):

#... implementación del analizador...

for token in analizador\_léxico(entrada):  
 print(token)



Al ejecutar este código con una entrada como "3+4", el analizador produciría la siguiente salida:

Número : 3 | Operador = + | Número : 4

## Aplicaciones Reales:

Creación de lenguajes Específicos de Dominio (DSD): Para tareas como la configuración de sistemas, la definición de pruebas o la descripción de modelos.

Desarrollo de compiladores: Como primera etapa en la construcción de compiladores para lenguajes de programación más complejos.

Herramientas de análisis de texto: Para tareas como la extracción de información, la clasificación de textos o la traducción automática.

## Conclusiones:

"Los analizadores léxicos son una aplicación de los compiladores que se encargan de verificar que el texto esté escrito en un formato aceptado para todo el programa que está escrito en un lenguaje de programación al igual que se encarga de verificar que tenga congruencia, los analizadores léxicos sirven en gran parte para resolver problemas que pueden surgir a causa de que el programa no tenga congruencia o no esté bien estructurado"

## Referencias:

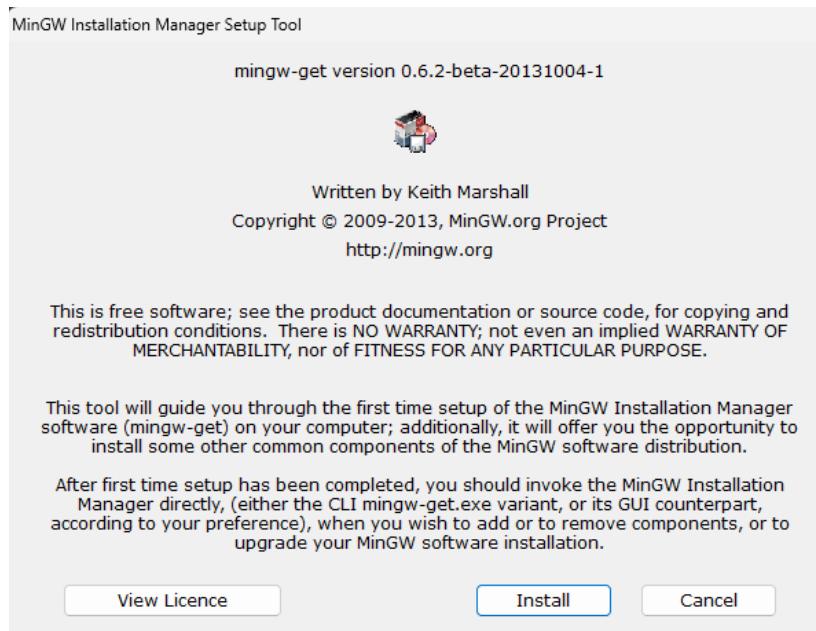
- Generadores de analizadores léxicos (2017, mayo, 15) Lenguajes y automatas 1 unidades : <https://lenguajesyautomatasblog.wordpress.com/2017/05/15/generadores-de-analizadores-lexicos/>
- Josuecelis, PC (2015, mayo 25). Aplicaciones De un Analizador Léxico. <https://jazieljousuelis.wordpress.com/2015/05/25/aplicaciones-de-un-analizador-lexical/>
- Universidad de Guanajuato. (2022, julio 16) Clase digital 16 . Herramientas de software : Lex / Flex . Recursos educativos Abiertos : Sistema Universitario de Multi modalidad Educativo (SUME) - Universidad de Guanajuato. <https://blogs.ugto.mx/rea/clase-digital-2-herramientas-de-software-lex-flex/>
- (s/f-a) , Edu-rar . Recuperado 17 de septiembre de 2024 , de <https://dc.exa.unrc.edu.ar/staff/baverapapers/TesisJTLex-BaveriaNardio-02.pdf>

# Instalación de Flex

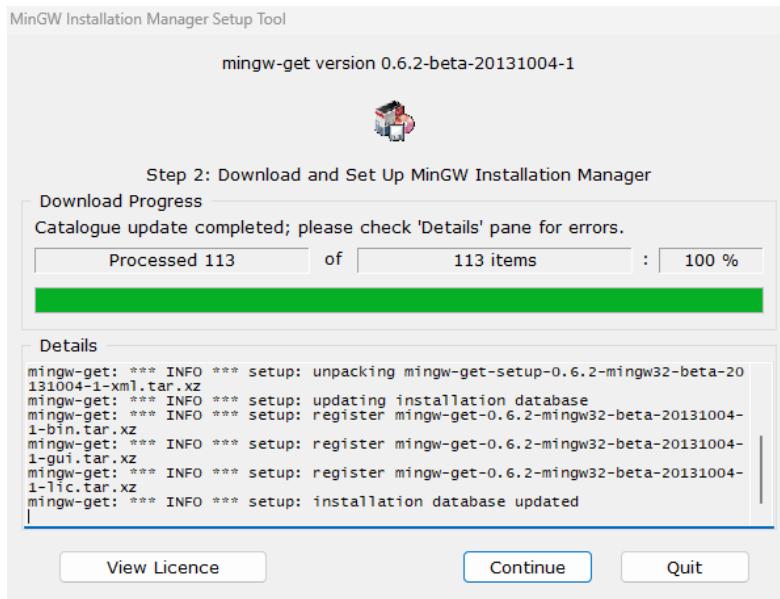
Para comenzar se debe instalar el programa MinGW para Windows a través de la siguiente liga: <https://sourceforge.net/projects/mingw/>. Descarga el instalador al hacer clic en el botón Download.

The screenshot shows the SourceForge project page for MinGW. At the top, there's a navigation bar with links for Open Source Software, Business Software, and Resources. The main title is "MinGW - Minimalist GNU for Windows". Below the title, it says "A native Windows port of the GNU Compiler Collection (GCC)" and "Brought to you by: catraxis, earnie, gressett, keltchmarshall". It features a "Download" button, a "Get Updates" button, and a "Share This" button. To the right, there's a sidebar titled "Recommended Projects" listing "Code::Blocks", "PDFCreator", "MinGW-w64 - for 32 and 64 bit Windows", and "ACRA". Below the main content, there's a "Project Samples" section showing four screenshots of the software interface. Under "Project Activity", it shows a recent modification by "Keith Marshall" on "mingw-get source nonesuch -- fails silently" three years ago. A sidebar on the right lists "Top Searches" including "download installer", "mingw-w64-install.exe", "mingw", "installer", "mingw-w64 - for 32 and 64 bit windows", "mingw compiler for mac", and "mingw-w64".

Una vez descargado el instalador lo ejecutamos para comenzar con una instalación típica al dar clic en el botón install.



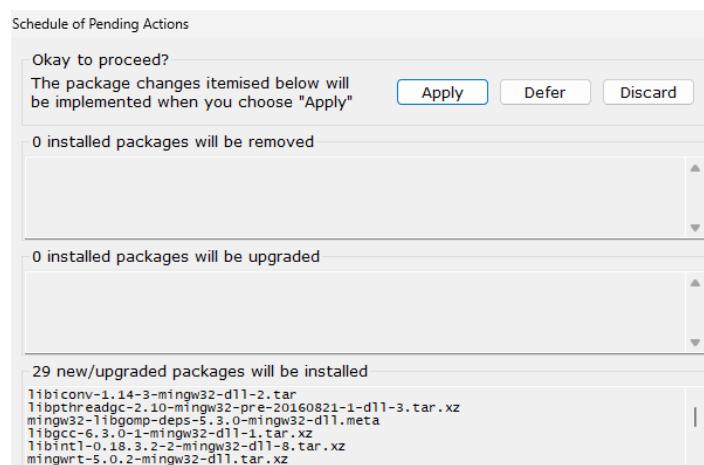
Esperamos un momento a que se termine la instalación y damos clic en el botón de continue.



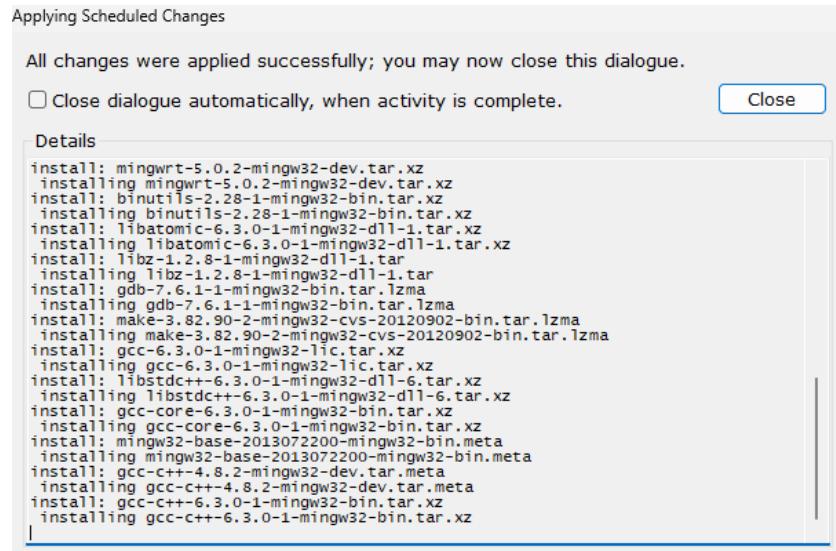
En la siguiente pantalla basta con marcar la segunda y quinta opción para instalar los paquetes que requerimos.

Package	Class	Installed Version	Repository Version	Description
<input type="checkbox"/> mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
<input checked="" type="checkbox"/> mingw32-base	bin		2013072200	A Basic MinGW Installation
<input type="checkbox"/> mingw32-gcc-ada	bin		6.3.0-1	The GNU Ada Compiler
<input type="checkbox"/> mingw32-gcc-fortran	bin		6.3.0-1	The GNU FORTRAN Compiler
<input checked="" type="checkbox"/> mingw32-gcc-g++	bin		6.3.0-1	The GNU C++ Compiler
<input type="checkbox"/> mingw32-gcc-objc	bin		6.3.0-1	The GNU Objective-C Compiler
<input type="checkbox"/> msys-base	bin		2013072300	A Basic MSYS Installation (meta)

Al dar clic en el botón de installation nos mostrara una pantalla donde nos preguntara si proceder con la instalación, solo hay que presionar apply.



Una vez descargados todos los paquetes y librerías solo hay que dar clic al botón close.



Para verificar que se ha instalado correctamente, abriremos la terminal como administrador y entraremos a la carpeta de MinGW y después a la carpeta bin, ahí ejecutaremos el comando dir para ver todos los directorios que se instalaron.

```
C:\MinGW\bin>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: 62E7-0F96
Directorio de C:\MinGW\bin

16/09/2024 12:38 p. m.    <DIR>.
16/09/2024 12:38 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.    1,730,456 as.exe
22/05/2017 04:18 a. m.    992,592 c++.exe
22/05/2017 04:18 a. m.    955,480 c++filt.exe
24/07/2017 11:03 a. m.    996,360 cpp.exe
22/05/2017 04:18 a. m.    1,433,032 c++filt.exe
22/05/2017 04:18 a. m.    163,342 dlwrap.exe
22/05/2017 04:18 a. m.    151,054 lddit.exe
22/05/2017 04:18 a. m.    939,372 ld.exe
24/07/2017 11:03 a. m.    78,070 gcc-ar.exe
24/07/2017 11:03 a. m.    78,070 gcc-rm.exe
24/07/2017 11:03 a. m.    78,070 gcc-ranlib.exe
24/07/2017 11:03 a. m.    985,342 ld.exe
24/07/2017 11:03 a. m.    596,494 gcov-tool.exe
24/07/2017 11:03 a. m.    818,518 gcov.exe
24/07/2017 11:03 a. m.    39,159 gpp.exe
14/09/2017 07:52 p. m.    988,334 gdbserver.exe
22/05/2017 04:18 a. m.    1,924,814 grep.exe
1,393,772 grep-3.6.exe
22/05/2017 04:18 a. m.    1,291,278 ld.exe
29/05/2017 03:01 p. m.    24,310 libatomic-1.dll
11/04/2017 07:38 p. m.    140,344 libatomic�ntherate-1.dll
24/05/2017 03:11 p. m.    938,157 libgcc_s_1.dll
06/09/2017 07:36 a. m.    478,960 libgcc-10.dll
29/05/2017 03:01 p. m.    156,178 libgcc-1.dll
1,400,000 libgcc-1.dll
30/04/2017 06:13 a. m.    404,613 libintl-8.dll
29/05/2017 04:02 p. m.    2,116,501 libisl-15.dll
26/05/2017 06:39 p. m.    1,600,344 libltdl-2.2.6.dll
26/05/2017 06:39 p. m.    188,411 libmpc-3.dll
17/01/2017 03:11 p. m.    378,357 libmpfr-4.dll
29/05/2017 03:08 p. m.    403,344 libquadmath-0.dll
29/05/2017 03:08 p. m.    14,521 libssp-0.dll
29/05/2017 03:08 p. m.    1,598,122 libstdc++-6.dll
04/10/2017 01:28 p. m.    52,224 mingw-get.exe
29/05/2017 03:11 p. m.    939,372 mingw32-g++.exe
29/05/2017 02:59 p. m.    997,960 mingw32-g++.exe
24/07/2017 11:03 a. m.    995,342 mingw32-gcc-6.3.0.exe
24/07/2017 11:03 a. m.    79,070 mingw32-gcc-6.3.0.exe
24/07/2017 11:03 a. m.    78,070 mingw32-gcc-nm.exe
24/07/2017 11:03 a. m.    78,070 mingw32-gcc-ranlib.exe
01/09/2012 08:42 p. m.    219,662 mingw32-gcc.exe
06/12/2017 02:09 p. m.    16,765 mingw32.dll
22/07/2017 04:18 a. m.    178,958 mm.exe
1,168,304 objdump.exe
22/05/2017 04:18 a. m.    1,756,174 objdump-3.dll
21/08/2017 03:34 a. m.    149,774 pthead6c-3.dll
29/05/2017 04:18 a. m.    929,324 readline-8.dll
22/05/2017 04:18 a. m.    617,998 readline-8.exe
22/05/2017 04:18 a. m.    968,200 size.exe
```

Para verificar la versión de los compiladores ejecutamos los comandos "gcc --version, g++ --version".

```
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.

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.
```

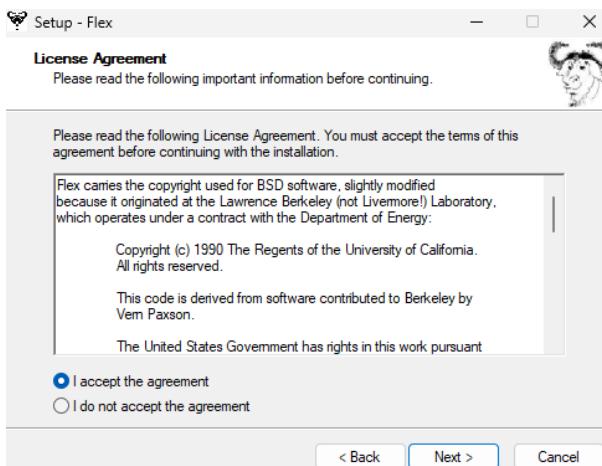
Lo siguiente será la instalación de flex, para poderlo descargar se debe ir a la siguiente liga <https://gnuwin32.sourceforge.net/packages/flex.htm>, solo hay que dar clic en download setup.

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

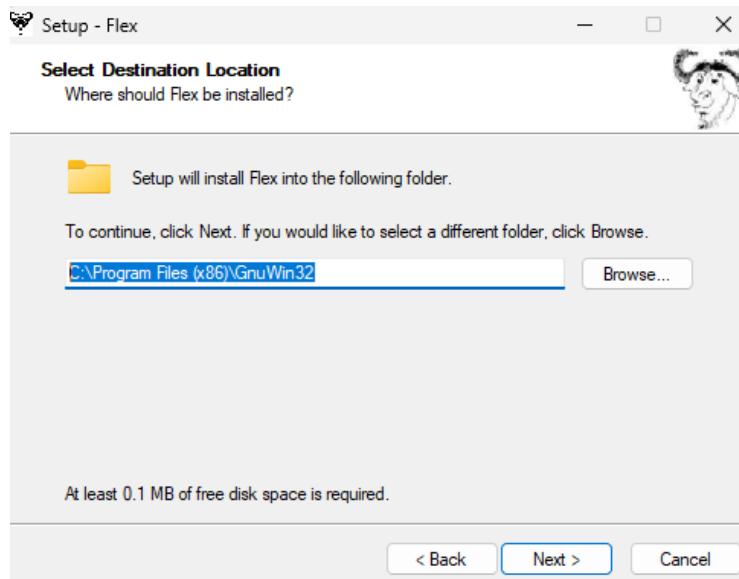
Al ejecutar el instalador del programa en la primera ventana solo hay que dar clic en next.



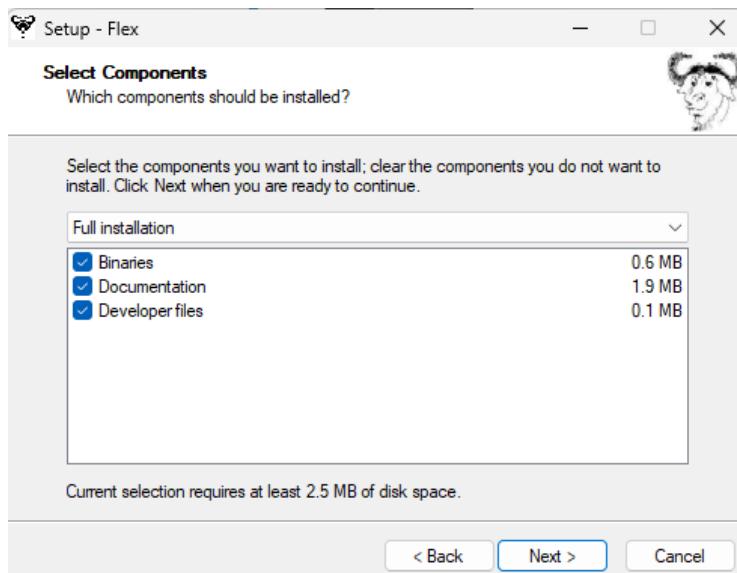
Posteriormente debemos marcar la opción "i accept the agreement" y dar clic en next.



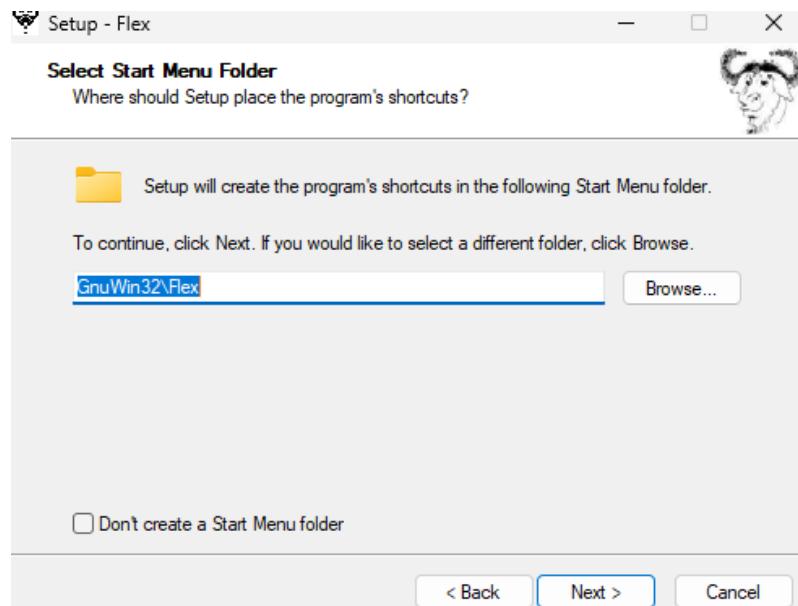
Dejaremos la ruta por defecto para la instalación, es recomendable guardar esta ruta para mas adelante y dar clic en next.



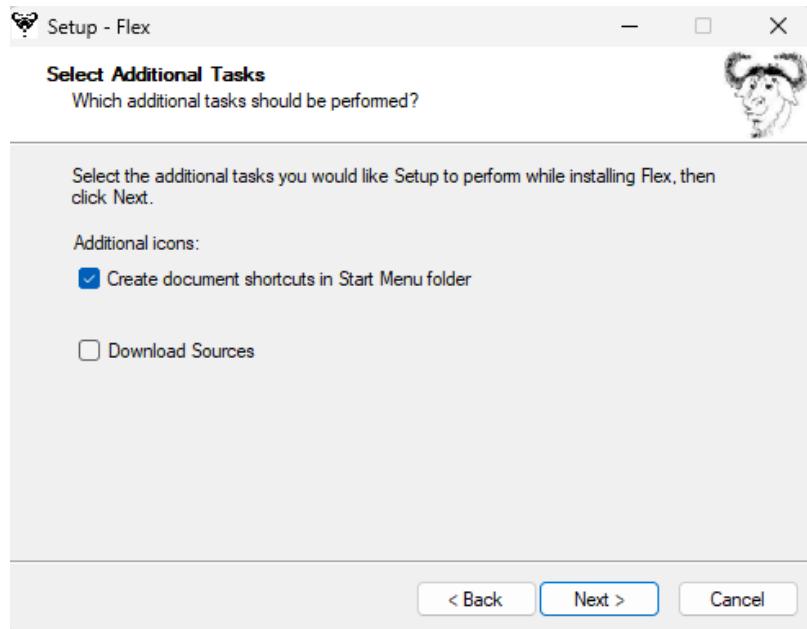
A partir de este punto bastara con dar clic en next.



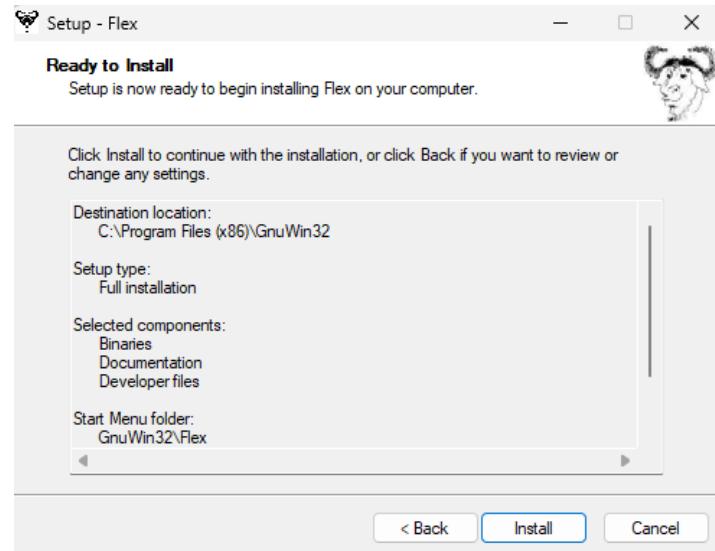
Al igual que la pantalla anterior solo daremos clic en next.



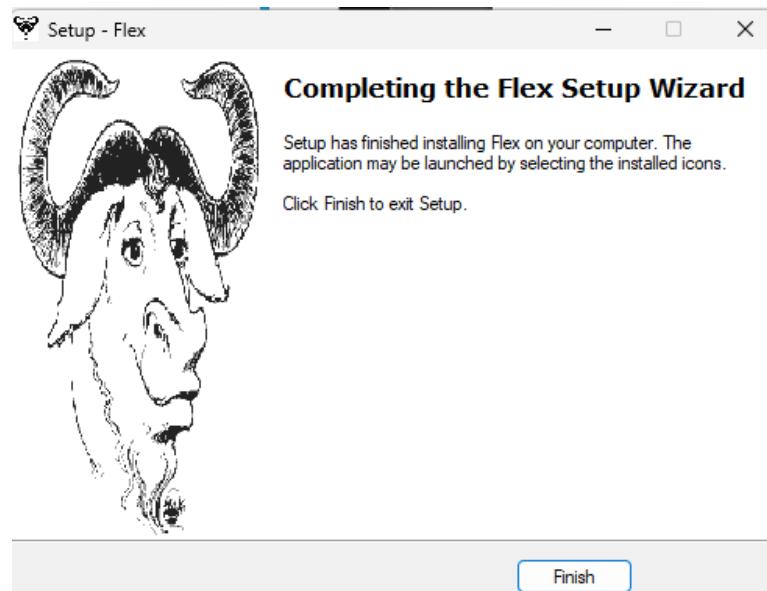
De nueva cuenta solo hay que dar clic en next.



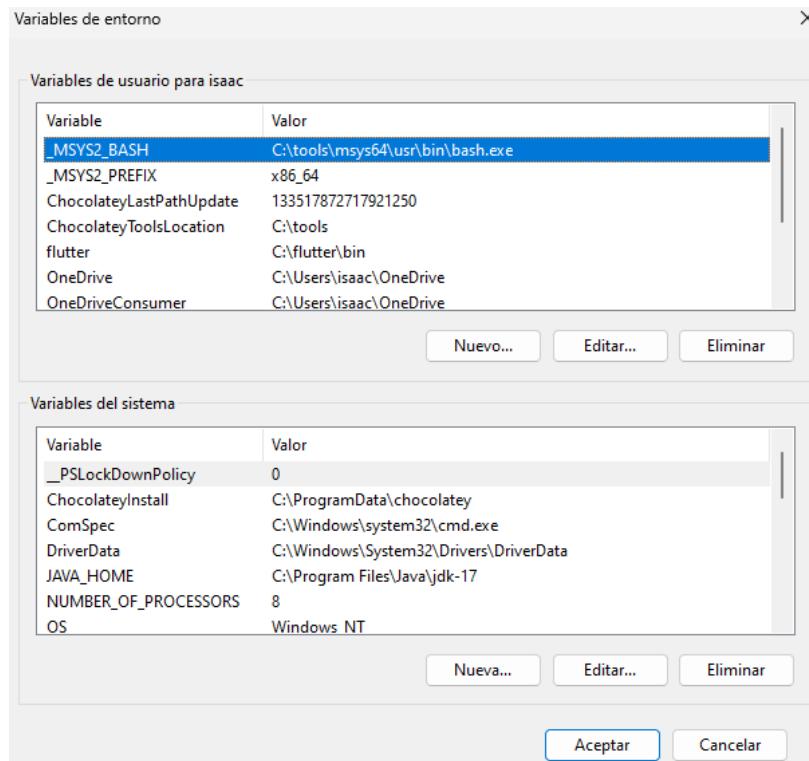
Ahora hay que dar clic en el botón install.



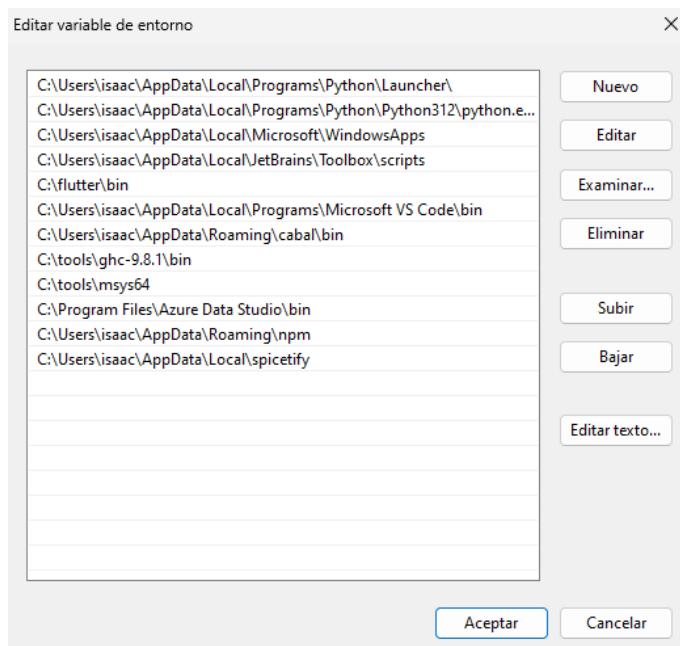
Para concluir damos clic en el botón finish.



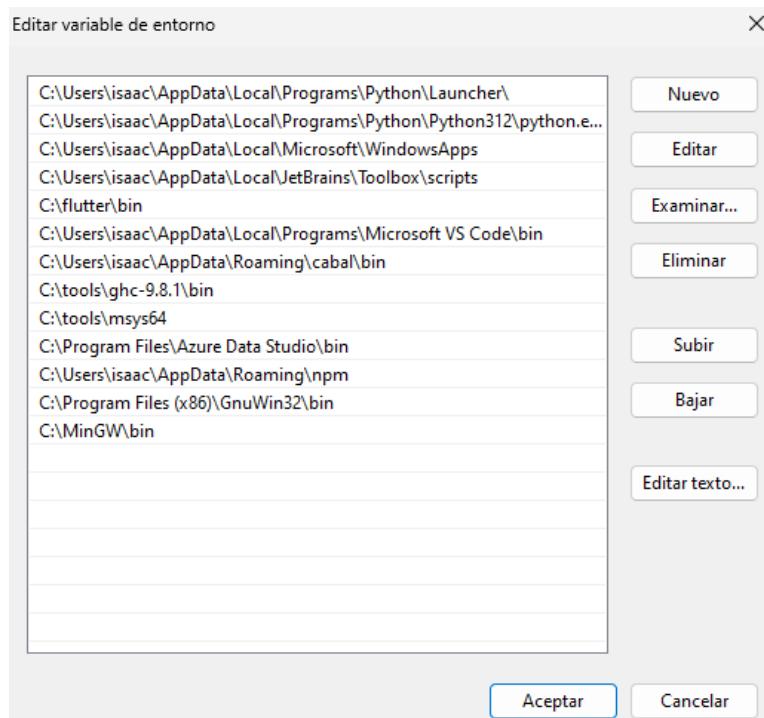
Una configuración extra que se debe realizar es crear las variables de entorno, para esto buscamos en el explorador variables de entorno del sistema y nos mostrara la siguiente pantalla.



Entramos a path y nos mostrara una pantalla como la siguiente, aquí daremos clic en nuevo e introduciremos las dos rutas de los programas que hemos instalado.



Una vez configuradas las rutas damos clic en aceptar.



Con esto concluye la instalación de flex para Windows.

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

DOCENTE DESIGNADO
ISC. RICARDO GONZÁLEZ GONZÁLEZ

PRACTICA No.	NOMBRE DE LA PRACTICA	DURACIÓN ( HORAS )
1	Ejemplo práctico de Flex	7 horas

1	INTRODUCCIÓN
Flex es una herramienta de generación de analizadores léxicos, como principal función tiene el ayudar en la creación de programas que reconozcan patrones en texto. Siendo así especialmente útil en el desarrollo de compiladores, interpretes o aplicaciones que necesiten procesar lenguajes o formatos en específico.	
El uso de Flex se da en una amplia variedad de aplicaciones, sistemas operativos y controladores, además dándole relevancia a la optimización de código para la creación de archivos ya que con esta se puede tener un impacto significativo en el rendimiento y velocidad con la que se ejecuta un programa.	
A lo largo de esta practica se busca explorar Flex, sus instrucciones y como realiza el reconocimiento de palabras, números y asignaciones.	

2	OBJETIVO ( COMPETENCIAS )
Como objetivo de esta practica se busca aprender a usar Flex para poder generar un analizador léxico que sea capaz de reconocer y clasificar elementos léxicos en el lenguaje C enfocándose particularmente en:	
<ul style="list-style-type: none"> <li>• Identificadores</li> <li>• Números</li> <li>• Operadores</li> </ul>	

**3**

### **MARCO TEÓRICO REFERENCIAL**

#### **1. Análisis Léxico**

La primera fase del proceso de compilación es el análisis léxico, durante esta fase el código fuente es leído y dividido en tokens. Un token es una secuencia de caracteres con significado colectivo, estos tokens pueden incluir palabras clave, identificadores, operadores, números, etc.

La tarea del analizador léxico es escanear el código fuente para categorizar las secuencias según patrones predefinidos para posteriormente enviar estos resultados al analizador sintáctico.

#### **2. Flex**

Se trata de una herramienta que genera analizadores léxicos a partir de una especificación escrita por el usuario. La especificación se trata de un conjunto de reglas que relacionan patrones regulares con acciones que se deben tomar al detectar un patrón.

Un archivo de especificaciones en Flex se da por tres partes:

- **Definiciones:** son patrones comunes que pueden ser usados más adelante.
- **Reglas:** asocian patrones con acciones, al detectar un patrón en el texto Flex realiza la acción correspondiente.
- **Código C:** es opcional, pero se puede incluir código en C para el código fuente de Flex.

Tokens del lenguaje C:

- **Identificadores:** secuencias de letras y dígitos, el primer carácter debe ser una letra.
- **Números:** en C, los números son secuencias de dígitos.
- **Operadores:** entre los operadores se encuentran símbolos como +, -, \*, /, y más, los cuales permiten realizar operaciones matemáticas y lógicas.

**4**

### **MATERIALES UTILIZADO**

- Equipo de computo
- Navegador web (para consulta de información)
- Flex y MinGW.
- Visual Studio para la creación del analizador léxico.

**5**

### **REQUISITOS BÁSICOS.**

- Sistema operativo: Windows 2000 / 98 / XP / Vista / 7 / 8 / 10 / 11
- Memoria (RAM): 256 MB
- Espacio disco duro: 100 MB

6

## DESARROLLO DE LA PRÁCTICA ( PASO A PASO )

Para comenzar con esta práctica y recordando lo que se mencionó en el objetivo de esta se realizará un analizador léxico que sea capaz de reconocer y clasificar elementos léxicos en el lenguaje C, principalmente los siguientes: identificadores, números y operadores.

Lo primero de todo será crear un archivo con extensión .l el cual contendrá las definiciones y reglas del analizador léxico.

## 1. Definición de patrones

```

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

En esta parte se define los patrones léxicos usando expresiones regulares. Estos describen los tokens que el analizador léxico buscará en la entrada que brindemos.

- **DIGITO:** Es una expresión regular que representa un único dígito, desde 0 hasta 9 y el patrón queda definido como [0-9].
- **LETRA:** Define un patrón para cualquier letra sea mayúscula o minúscula.
- **IDENTIFICADOR:** Comienza con una letra seguida opcionalmente de una secuencia de letras o dígitos. El símbolo \* indica que puede haber cero o más letras o dígitos después de la primera letra.
- **NUMERO:** Un número es una secuencia de uno o más dígitos. El símbolo + indica que debe haber al menos un dígito.
- **OPERADOR:** Este es un patrón que reconoce los operadores matemáticos comunes de C. Se usa la barra \ para escapar el símbolo de resta porque tiene un significado especial en expresiones regulares.

## 2. Definición de reglas

En esta parte se define por cada línea una regla léxica. Su estructura básica es patrón {acción}, donde:

- **Patrón:** Es la expresión regular por reconocer.
- **Acción:** Es el código en C que se ejecutara cuando el patrón coincide con el texto de entrada.

```

{IDENTIFICADOR}      { printf("IDENTIFICADOR: %s\n", yytext); }
{NUMERO}              { printf("NUMERO: %s\n", yytext); }
{OPERADOR}            { printf("OPERADOR: %s\n", yytext); }

[ \t\n]+               ;
.

{ printf("Caracter no reconocido: %s\n", yytext); }
  
```

Al detectar uno de los patrones definidos, se ejecuta su acción correspondiente:

- **IDENTIFICADOR:** Al reconocer el patrón de un identificador, se imprime el texto del identificador junto con el respectivo mensaje “IDENTIFICADOR”. La variable yytext es una variable de Flex que contiene la porción del texto de entrada que coincide con el patrón.
- **NUMERO:** Al detectar un número, se imprime “NUMERO” seguido del valor del número encontrado.
- **OPERADOR:** Al encontrar un operador, se imprime “OPERADOR” junto al operador encontrado.

- .. El punto es una expresión regular que sirve para cualquier carácter que no haya sido reconocido por alguna de las reglas anteriores, y al ocurrir esto el mensaje de salida será “Carácter no reconocido:” seguido del carácter.

**Flujo del programa:**

1. **Entrada:** El analizador léxico toma una entrada dada por el usuario en texto.
2. **Análisis léxico:** Flex analiza la entrada, busca coincidencias con los patrones definidos.
3. **Acciones:** Por cada coincidencia que se dé, Flex ejecutara una acción.
4. **Finalización:** Al llegar al final de la entrada o encontrar un error el programa terminara.

**Código completo**

```
%{  
#include <stdio.h>  
%}  
  
DIGITO      [0-9]  
LETRA       [a-zA-Z]  
IDENTIFICADOR {LETRA} ({LETRA} | {DIGITO}) *  
NUMERO      {DIGITO} +  
OPERADOR    [+\\-*/=]  
  
%%  
  
{ IDENTIFICADOR }      { printf("IDENTIFICADOR: %s\n", yytext); }  
{ NUMERO }              { printf("NUMERO: %s\n", yytext); }  
{ OPERADOR }            { printf("OPERADOR: %s\n", yytext); }  
  
[ \t\n]+                ;  
  
.                  { printf("Caracter no reconocido: %s\n", yytext); }  
  
%%  
  
int main(int argc, char **argv)  
{  
    yylex();  
    return 0;  
}  
  
int yywrap() {  
    return 1;  
}
```

**Pruebas y resultados**

Declaración de variables y asignación:

```
D:\Códigos\Flex>a
int suma = 10 + 20;
IDENTIFICADOR: int
IDENTIFICADOR: suma
OPERADOR: =
NUMERO: 10
OPERADOR: +
NUMERO: 20
Caracter no reconocido: ;
|
```

Operación aritmética compleja:

```
D:\Códigos\Flex>a
resultado = (a * 3) - (b / 5);
IDENTIFICADOR: resultado
OPERADOR: =
Caracter no reconocido: (
IDENTIFICADOR: a
OPERADOR: *
NUMERO: 3
Caracter no reconocido: )
OPERADOR: -
Caracter no reconocido: (
IDENTIFICADOR: b
OPERADOR: /
NUMERO: 5
Caracter no reconocido: )
Caracter no reconocido: ;
|
```

5

**BITÁCORA DE INCIDENCIAS**

PROBLEMA	FECHA	HORA	SOLUCIÓN	FECHA	HORA
Desconocimiento en el uso de Flex	11/09/2024	3:00 p.m.	Investigación de ejemplos y manuales	11/09/2024	4:00 p.m.

6

**OBSERVACIONES**

A pesar de tratarse de un ejemplo bastante sencillo de las capacidades de Flex, el hecho de desconocer cómo funcionaba Flex propuso un reto ya que había que investigar manuales y algunos foros que proporcionaran dicha información, además de algunos ejemplos para comprender aún mejor su funcionamiento.

Una vez pasado este obstáculo se puede concluir que Flex como herramienta es verdaderamente útil y se entiende su fama, se pueden llevar a cabo ejercicios muy complejos en un futuro que nos permita mejorar y comprender aún más del análisis léxico y sintáctico.

7

**ANEXOS**

Para el desarrollo de esta práctica dado que no se tenía conocimiento previo en el uso de Flex se consultaron algunos ejemplos ya previamente diseñados de analizadores léxicos realizados con Flex pero uno de los más relevantes fue el siguiente:

```
/* escáner para un lenguaje de juguete al estilo de Pascal */

%{
/* se necesita esto para la llamada a atof() más abajo */
#include <math.h>
}

DIGITO    [0-9]
ID        [a-z] [a-z0-9] *

%%

{DIGITO}+  {
    printf( "Un entero: %s (%d)\n", yytext,
            atoi( yytext ) );
}

{DIGITO}+.{DIGITO}*      {
    printf( "Un real: %s (%g)\n", yytext,
            atof( yytext ) );
}

if|then|begin|end|procedure|function      {
    printf( "Una palabra clave: %s\n", yytext );
}
```

```

{ID}          printf( "Un identificador: %s\n", yytext );

"+|-/*"/     printf( "Un operador: %s\n", yytext );

"^{[^}\n]*}"   /* se come una linea de comentarios */

[ \t\n]+       /* se come los espacios en blanco */

.              printf( "Caracter no reconocido: %s\n", yytext );

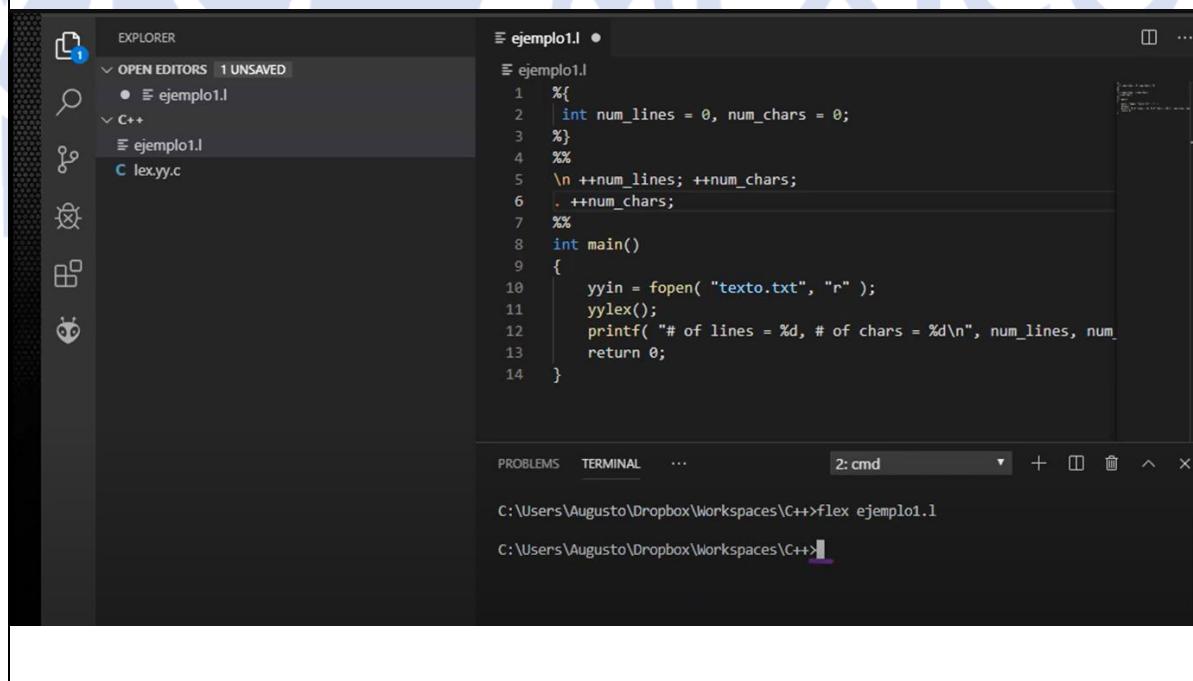
%%

main( argc, argv )
int argc;
char **argv;
{
    ++argv, --argc; /* se salta el nombre del programa */
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

    yylex();
}
  
```

Al ser un ejemplo medianamente complejo fue muy útil para desarrollar el analizador léxico de esta práctica y hacer pruebas con él.

Incluso el ejemplo del video usado para realizar la instalación de Flex fue relevante para entender desde como se crea el archivo así como a través de consola generar el analizador y ejecutarlo.



```

ejemplo1.l
1  %{           int num_lines = 0, num_chars = 0;
2  %}
3  %%
4  \n ++num_lines; ++num_chars;
5  .++num_chars;
6  %%
7  int main()
8  {
9      yyin = fopen( "texto.txt", "r" );
10     yylex();
11     printf( "# of lines = %d, # of chars = %d\n", num_lines, num_
12         return 0;
13     }
14 }

PROBLEMS TERMINAL ...
C:\Users\Augusto\Dropbox\Workspaces\C++>flex ejemplo1.l
C:\Users\Augusto\Dropbox\Workspaces\C++>
  
```

7

**REFERENCIAS BIBLIOGRÁFICAS**

- *Instalación Flex, minGW, c++ para Windows - Compiladores analizador léxico.* (2019, 24 septiembre). [Vídeo]. YouTube. Recuperado 11 de septiembre de 2024, de [https://www.youtube.com/watch?v=nFGcPlW\\_rOw](https://www.youtube.com/watch?v=nFGcPlW_rOw)
- Canonical. (n.d.). *Ubuntu Manpage: flex - generador de analizadores léxicos rápidos.* <https://manpages.ubuntu.com/manpages/trusty/es/man1/flex.1.html>
- Béjar Hernández, R. (2004). *Introducción a Flex y Bison.* Universidad de Zaragoza. [https://webdiis.unizar.es/asignaturas/LGA/material\\_2004\\_2005/Intro\\_Flex\\_Bison.pdf](https://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf)



FOLIO 111

INSTITUTO TECNOLÓGICO NACIONAL DE  
MÉXICO EN CELAYA

MATERIA: LENGUAJES Y AUTÓMATAS II

PROFESOR: ISC. RICARDO GONZÁLEZ GONZÁLEZ

ALUMNOS

ISAAC SALVADOR BRAVO ESTRADA 2003048

GUILLERMO PEASLAND AGUILAR 20030737

MARÍA DEL CARMEN CHÁVEZ PATIÑO 20030296

LUIS FERNANDO MENDOZA JAVALERA 1930536

FECHA DE ENTREGA: 17 DE SEPTIEMBRE DE 2024

MONOGRAFÍA TÉCNICA EN GRAMÁTICAS LIBRES  
DE CONTEXTO Y ÁRBOLES DE DERIVACIÓN

EQUIPO NO. 3

# ÍNDICE

- INTRODUCCIÓN
- HISTORIA Y CONTEXTO DE LAS GLC
- GRAMÁTICAS LIBRES DE CONTEXTO
  - DEFINICIÓN FORMAL
  - TIPOS DE REGLAS DE PRODUCCIÓN
  - PROPIEDADES CLÁSICAS DE LAS GLC
  - RELACIÓN ENTRE LENGUAJES FORMALES
  - NOTACIÓN DE BACKUS-NAUR Y GRAMÁTICA AUMENTADA
- ÁRBOLES DE DERIVACIÓN
  - DEFINICIÓN DE ÁRBOLES DE DERIVACIÓN
  - ESTRUCTURA Y COMPONENTES
  - PROCESO DE DERIVACIÓN: IZQUIERDA Y DERECHA
  - PROPIEDADES DE LOS ÁRBOLES DE DERIVACIÓN
  - INTERPRETACIÓN SINTÁCTICA
  - INTERPRETACIÓN SEMÁNTICA
  - INTERPRETACIÓN PRÁCTICA
- AMBIGÜEDAD Y SU IMPACTO EN EL DISEÑO DE LENGUAJES
  - AMBIGÜEDAD INHERENTE DE LAS GLC
  - EJEMPLOS CLÁSICOS DE GRAMÁTICA AMBIGUA
  - TÉCNICAS DE ELIMINACIÓN Y MITIGACIÓN DE AMBIGÜEDAD
- ANÁLISIS DE ALGORITMOS Y COMPLEJIDAD EN GLC
  - ALGORITMOS DE ANÁLISIS SINTÁCTICO (PARSING)
  - ALGORITMOS DETERMINISTAS Y NO DETERMINISTAS
  - COMPLEJIDAD COMPUTACIONAL EN EL PARSING
- APLICACIONES DE LAS GRAMÁTICAS LIBRES DE CONTEXTO
  - COMPILADORES Y LENGUAJES DE PROGRAMACIÓN
  - PROCESAMIENTO DEL LENGUAJE NATURAL
  - APLICACIONES EN BIONFORÁTICA
  - VERIFICACIÓN FORMAL DE SISTEMAS
- DESAFIOS EN LA APLICACIÓN PRÁCTICA
  - LIMITACIONES DE LAS GLC
  - EXTENSIONES DE LAS GLC
  - IMPACTO EN EL DISEÑO DE LENGUAJES MODERNOS
- CONCLUSIONES

# GRAMÁTICAS LIBRES DE CONTEXTO Y ÁRBOLES DE DERIVACIÓN

## INTRODUCCIÓN

Las gramáticas libres de contexto (GLC) son uno de los elementos más influyentes y estudiados en la teoría de lenguajes formales y la computación. Introducidas por Noam Chomsky en la década de 1950 como parte de su trabajo pionero en lingüística formal, las GLC son capaces de modelar un amplio rango de lenguajes, desde lenguajes naturales hasta lenguajes de programación. Estos lenguajes, definidos formalmente por un conjunto de reglas de producción, no solo permiten representar la estructura de cadenas de símbolos, sino también implementar procesos de análisis sintáctico en sistemas computacionales.

Por otro lado, los árboles de derivación proporcionan una representación gráfica y jerárquica de cómo una cadena se deriva de un símbolo inicial. Esta relación entre gramáticas y árboles es esencial para la construcción de compiladores, parsers y otras herramientas computacionales que requieren el análisis sintáctico de una entrada simbólica.

# HISTORIA Y CONTEXTO DE LAS GRAMÁTICAS LIBRES DE CONTEXTO

Las GLC nacen del trabajo de Chomsky en 1956, cuando formuló la jerarquía de Chomsky, un sistema que clasifica los lenguajes formales en distintos tipos según su complejidad. Las GLC ocupan un lugar prominente en la jerarquía, ya que son lo suficientemente expresivas para capturar la estructura de la mayoría de los lenguajes de programación y varios lenguajes naturales, pero también lo suficientemente restringidas para permitir la creación de algoritmos eficientes para el análisis sintáctico.

En los años 1960 y 1970, las GLC se convirtieron en la piedra angular del diseño de compiladores. El trabajo de John Backus y Peter Naur en la creación del lenguaje de descripción de Sintaxis BF o BNF (Backus-Naur Form) permitió formalizar el uso de GLC en el análisis sintáctico de lenguajes de programación. Desde entonces, las GLC se han utilizado para definir la sintaxis de muchos lenguajes de programación. Desde entonces, lenguajes como Pascal, C y Java, y continúan siendo un área activa de investigación en el campo del procesamiento del lenguaje natural y otras áreas.

# GRAMÁTICAS LIBRES DE CONTEXTO

## DEFINICIÓN FORMAL

Una GLC se define como una cuádrupla

$$G = (V, T, P, S)$$

donde:

• V es el conjunto de símbolos no terminales.  
Estos símbolos representan las categorías sintácticas que se expanden en símbolos terminales.

• T es el conjunto de símbolos terminales. Estos son los símbolos del alfabeto que constituyen las cadenas finales del lenguaje generado.

• P es el conjunto de reglas de producción. Cada regla tiene la forma  $A \rightarrow \alpha$ , donde  $A \in V$  y  $\alpha \in (V \cup T)^*$ .

• S es el símbolo inicial a partir del cual se comienza la derivación.

Una cadena de un lenguaje posee una GLC si puede obtenerse mediante la aplicación sucesiva de reglas de producción a partir del símbolo inicial  $S$ .

## TIPOS DE REGLAS DE PRODUCCIÓN

Las reglas de producción en una GLC no están condicionadas por el contexto de los símbolos no terminales que se van a expandir, lo que las diferencia de las gramáticas dependientes de contexto.

Las reglas de producción pueden ser de las siguientes formas:

- **Reglas Simples:**  $A \rightarrow a$ , donde A es un no terminal y a es un terminal.
- **Reglas recursivas:**  $A \rightarrow Ax$ , lo que permite la repetición de estructuras dentro del lenguaje.
- **Reglas complejas:**  $A \rightarrow BC$ , donde A,B,C son no terminales. Estas reglas permiten la combinación de múltiples componentes en una cadena.

Estas reglas permiten describir lenguajes complejos con estructuras repetitivas y jerárquicas, características típicas de los lenguajes humanos y de programación.

### PROPIEDADES CLÁSICAS DE LAS GLC

Los GLC poseen varias propiedades clave que son útiles en diversas aplicaciones:

1. **Cierre bajo operaciones:** Las GLC son cerradas bajo operaciones como la unión, la concatenación y las estrellas de Kleene.
2. **Capacidad de generar lenguajes balanceados:** Las GLC son especialmente útiles para generar lenguajes que requieren el balanceo de símbolos, como los lenguajes de paréntesis平衡ados o los lenguajes de expresiones aritméticas.

3. Dependencia del contexto: Al no depender del contexto de los símbolos, las reglas de producción de una GLC son aplicables en cualquier lugar una derivación.

La jerarquía de Chomsky introdujo cuatro tipos principales de gramática:

1. **Tipo 0:** Gramáticas recursivas enumerables.
2. **Tipo 1:** Gramáticas dependientes del contexto.
3. **Tipo 2:** Gramáticas libres de contexto (GLC)
4. **Tipo 3:** Gramáticas Regulares.

Las gramáticas libres de contexto, situadas en el tipo 2, representan un equilibrio entre simplicidad y expresividad, lo que hace aplicables tanto en la teoría de automatas como en la ingeniería de software.

## RELACIONES ENTRE LENGUAJES FORMALES

Los GLC están situados en un nivel intermedio en la jerarquía de Chomsky. Son más expresivas que las gramáticas regulares, pero menos poderosas que las gramáticas dependientes de contexto. Esto las convierte en una herramienta ideal para modelar lenguajes que tienen una estructura más compleja que los lenguajes regulares, pero que aún pueden ser procesados por algoritmos eficientes.

## NOTACIÓN DE BACKUS-NAUR Y GRÁFICAS AUMENTADAS

El trabajo de Backus y Naur formalizó un estilo de notación conocido como Backus-Naur Form (BNF), que simplifica la definición de GLC, permitiendo especificar lenguajes de manera concisa. Por ejemplo, la siguiente regla en BNF:

```
PHP
<expresión> ::= <expresión> + <términos> | <término>
<términos> ::= <términos> * <factores> | <factores>
<factores> ::= ( <expresión> ) | id.
```

describe expresiones aritméticas, permitiendo la representación jerárquica de operadores y operandos.

## ÁRBOLES DE DERIVACIÓN

### DEFINICIÓN DE ÁRBOLES DE DERIVACIÓN

Un árbol de derivación es una estructura que representa graficamente la derivación de una cadena en una GLC. El árbol se construye a partir del símbolo inicial y expande sus nodos mediante la aplicación de reglas de producción, hasta que todas las hojas del árbol corresponden a símbolos terminales. Los árboles de derivación permiten visualizar el proceso de generación de una cadena y la estructura jerárquica del lenguaje subyacente.

## ESTRUCTURA Y COMPONENTES

Un árbol de derivación consta de los siguientes componentes:

- **RAÍZ:** El nodo inicial del árbol, que corresponde al símbolo inicial de la gramática.
- **Nodos INTERNOS:** Representan los símbolos no terminales. Estos nodos se expanden según las reglas de producción.
- **HOJAS:** Representan los símbolos terminales. Cuando se alcanzan las hojas, se ha completado la derivación de la cadena.

La construcción de un árbol de derivación sigue un proceso iterativo, donde en cada paso se aplica una regla de producción a uno de los nodos no terminales.

## PROCESO DE DERIVACIÓN: IZQUIERDA Y DERECHA

Existen dos enfoques principales para realizar una derivación:

**DERIVACIÓN POR LA IZQUIERDA:** En cada paso, se expande el símbolo no terminal más a la izquierda. Esta estrategia garantiza que las reglas se apliquen en un orden secuencial.

**DERIVACIÓN POR LA DERECHA:** Se expande el símbolo terminal más a la derecha en cada paso. Aunque este enfoque es menos intuitivo, es útil para ciertos algoritmos de parsing.

## PROPIEDADES DE LOS ÁRBOLES DE DERIVACIÓN

Los árboles de derivación tienen varias propiedades clave:

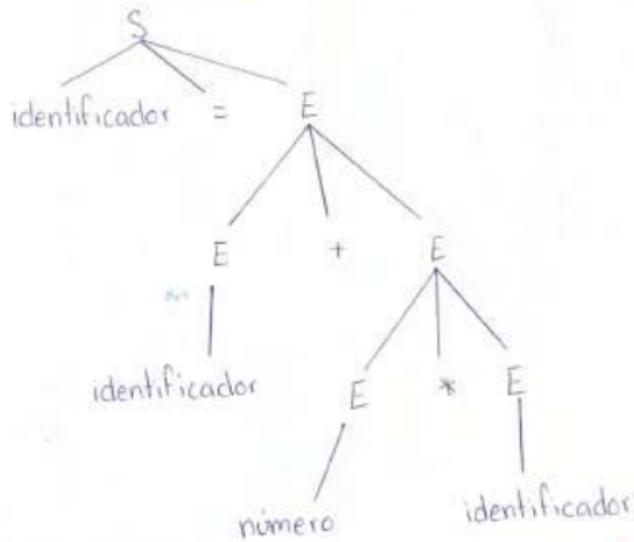
- **UNICIDAD:** Para gramáticas no ambiguas, cada cadena tiene un único árbol de derivación.
- **AMBIGÜEDAD:** Si una gramática es ambigua, una cadena puede tener múltiples árboles de derivación.
- **PROFOUNDIDAD Y COMPLEJIDAD:** La profundidad del árbol puede afectar la complejidad del análisis sintácticos.

## INTERPRETACIÓN SINTÁCTICA

El análisis de un árbol de derivación permite interpretar la sintaxis de la cadena generada. En lenguajes de programación, los árboles de derivación se utilizan para identificar la estructura de una expresión o una instrucción, lo que permite su posterior análisis semántico y optimización.



## ÁRBOL SINTÁCTICO DE DERIVACIÓN



## AMBIGÜEDAD Y SU IMPACTO EN EL DISEÑO DE LENGUAJES

### AMBIGÜEDAD INHERENTE DE LAS GLC

Una gramática es ambigua si una misma cadena puede derivarse de más de un modo, es decir, si existen múltiples árboles de derivación para una misma cadena. La ambigüedad es un problema importante en el diseño de lenguajes, ya que puede causar que el significado de una cadena sea incierto o interpretable de múltiples maneras.

## EJEMPLOS CLÁSICOS DE GRAMÁTICAS AMBIGUAS.

Uno de los ejemplos más comunes de ambigüedad que genera expresiones aritméticas. Consideremos la siguiente gramática:

- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow (E)$
- $E \rightarrow id$
- $I \rightarrow a$
- $I \rightarrow b$
- $I \rightarrow Ia$
- $I \rightarrow Ib$
- $I \rightarrow I0$
- $I \rightarrow II$

Esta gramática es ambigua porque una expresión como  $id + id * id$  puede derivarse de múltiples maneras. La ambigüedad puede ser problemática, ya que en este caso afecta el orden en el que se realizan las operaciones aritméticas.

## TECNICAS DE ELIMINACIÓN Y MITIGACION DE AMBIGÜEDAD

Existen varias técnicas para eliminar o mitigar la ambigüedad en una GLC:

**1.- FACTORIZACIÓN:** Reestructurar la gramática para evitar conflictos en las reglas de producción.

**2.- DEFINIR PRECEDENCIA Y ASOCIATIVIDAD:** En el caso de expresiones aritméticas, se pueden introducir reglas que especifican el orden de evaluación

de las operaciones

**3.- Uso DE GRAMATICAS NO AMBIGUAS:** En algunos casos, es posible diseñar gramáticas alternativas que no presentan ambigüedad, aunque esto puede ser complicado para ciertos lenguajes.

## ANÁLISIS DE ALGORITMOS Y COMPLEJIDAD EN GLC.

### ALGORITMOS DE ANÁLISIS SINTÁCTICO (PARSING)

El parsing es el proceso de analizar una cadena de entrada para determinar si puede generarse mediante una GLC y, en caso afirmativo, construir un árbol de derivación que representa la estructura de la cadena. Existen varios algoritmos para el análisis sintáctico, entre ellos:

- **ALGORITMOS DE PARSING DESCENDENTE:** Estos algoritmos intentan construir el árbol de derivación desde la raíz hasta las hojas, aplicando reglas de producción de manera recursiva

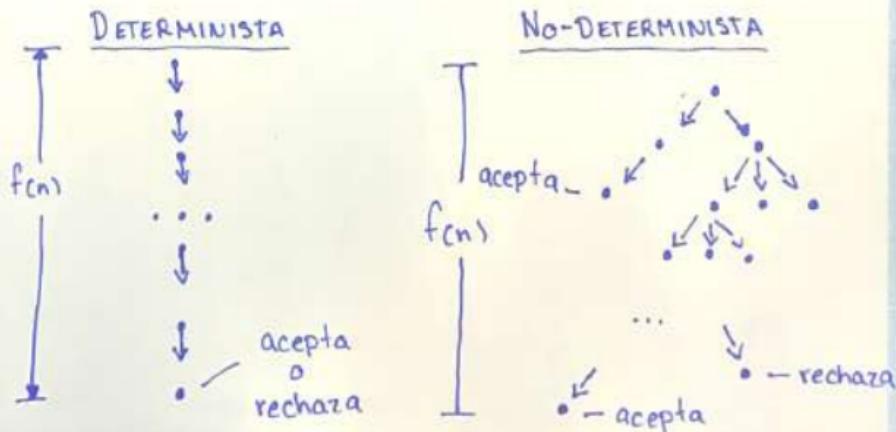
- **ALGORITMOS DE PARSING ASCENDENTE:** Estos algoritmos construyen el árbol de derivación de abajo hacia arriba, empezando con los símbolos terminales y combinándolos para formar nodos <sup>no</sup>terminales

## ALGORITMOS DETERMINISTAS Y NO DETERMINISTAS

Los algoritmos de parsing pueden clasificarse como deterministas o no deterministas:

• **PARSING DETERMINISTA:** En este caso, el algoritmo tiene la capacidad de tomar decisiones sin ambigüedad en cada paso del proceso de análisis. Los lenguajes libres de contexto determinista (LALR, LR) permiten el uso de parsers eficientes y rápidos.

• **PARSING NO DETERMINISTA:** En este caso, el algoritmo puede tener múltiples opciones en cada paso y debe probar cada una de ellas para encontrar la derivación correcta. El parsing no determinista suele ser más costoso en términos computacionales.



## COMPLEJIDAD COMPUTACIONAL EN EL PARSING

La complejidad del análisis sintáctico de un lenguaje libre de contexto depende del tipo de gramática y del algoritmo utilizado. Para gramáticas LL(1) o LR(1), el parsing puede realizarse en tiempo lineal respecto al tamaño de la cadena de entrada. Sin embargo, en el caso de gramáticas más complejas o ambiguas, la complejidad puede aumentar significativamente, llegando a ser polinómica o incluso exponencial en algunos casos.

## APLICACIONES DE LAS GRAMATICAS LIBRES DE CONTEXTO

### COMPILADORES Y LENGUAJES DE PROGRAMACION

Una de las aplicaciones más importantes de las GLC es en el diseño de compiladores. Los compiladores utilizan GLC para definir la sintaxis de los lenguajes de programación y construir parsers que analicen los programas de entrada. Los árboles de derivación generados durante este proceso son esenciales para la posterior generación de código intermedio y la optimización del programa.

## PROCESAMIENTO DEL LENGUAJE NATURAL

En el procesamiento del lenguaje natural (PLN), las GLC se utilizan para modelar la estructura sintáctica de las oraciones. Los árboles de derivación permiten representar la jerarquía gramatical de una oración, lo que es útil para tareas como el análisis sintáctico, la traducción automática y la generación del lenguaje natural.

## APLICACIONES EN BIOINFORMÁTICA

Las GLC también encuentran aplicaciones en bioinformática, especialmente en el análisis de secuencias de ADN y ARN. Al modelar las estructuras secundarias de estas moléculas, las GLC pueden utilizarse para prever cómo se pliegan las secuencias de bases nitrogenadas, lo que tiene implicaciones importantes para la comprensión de su funcionalidad biológica.

## VERIFICACIÓN FORMAL DE SISTEMAS

Las GLC se utilizan en la verificación formal de sistemas, donde se requiere garantizar que ciertos sistemas cumplen con especificaciones predefinidas. Los lenguajes libres de contexto proporcionan una forma eficaz de especificar las propiedades estructurales de los sistemas que se desean verificar, lo que permite realizar análisis automáticos y rigurosos.

## DESAFIOS EN LA APLICACIÓN PRACTICA

### LIMITACIONES DE LAS GLC

A pesar de su potencia, las GLC presentan algunas limitaciones. Por ejemplo, no pueden modelar dependencias de contexto, como aquellas que se encuentran en algunos lenguajes naturales o en ciertas características avanzadas de los lenguajes de Programación.

### EXTENSIONES DE LAS GLC

Para superar estas limitaciones, se han propuesto varias extensiones de las GLC, como las gramáticas dependientes de contexto y las gramáticas sensibles al contexto. Estas extensiones permiten una mayor expresividad a costa de una mayor complejidad computacional.

### IMPACTO EN EL DISEÑO DE LENGUAJES

#### MODERNOS

El diseño de lenguajes de programación modernos se ve influido por las capacidades y limitaciones de las GLC. Lenguajes como C++ y Java tienen características que requieren parsers más avanzados para manejar su complejidad sintáctica, lo que ha llevado al desarrollo de nuevos enfoques y herramientas en el análisis sintáctico.

## CONCLUSIONES

Las gramáticas libres de contexto y los árboles de derivación son componentes esenciales en el análisis y diseño de lenguajes formales. Su impacto en la computación es vasto, desde el diseño de compiladores hasta aplicaciones en el procesamiento del lenguaje natural y la bioinformática. Si bien presentan ciertos desafíos, las GLC siguen siendo una herramienta poderosa para modelar y analizar la estructura de los lenguajes, y su estudio continúa siendo relevante en la investigación contemporánea.

## JERARQUÍA DE CHOMSKY

LENGUAJE	GRAMATICA	MAQUINA	EJEMPLO
Dependiente del Contexto	Tipo 1 $(\alpha\beta \rightarrow \alpha\gamma)$	??	$ww, a^n b^n c^n$
Independiente del Contexto	Tipo 2 $(V \rightarrow \alpha)$	Autómata de pila	$ww^t, a^n b^n$
Regular	Tipo 3 $(V \rightarrow aA) \cup \epsilon$	Autómata Finito	$w, a^*$

## Referencias

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd ed.). Addison-Wesley.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3), 113–124.
- Hopcroft, J. E., & Ullman, J. D. (1979). Introduction to Automata Theory, Languages, and Computation. Addison-Wesley.
- Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing (3rd ed.). Pearson.
- Sippu, S., & Soisalon-Soininen, E. (1990). Parsing Theory. Volume 1: Languages and Parsing. Springer-Verlag.

# Instituto Tecnológico Nacional de México en Celaya

[Diagramas de Sintaxis]

## Alumnos

Isoarc Salvador Bravo Estrada - 20030048

Guillermo Pearsland Aguilar - 20030737

Maria del Carmen Chávez Partida - 20030296

Luis Fernando Mendoza Javalera - 19030536

## Profesor

ISC. Ricardo González González

Equipo — No. 3

# Contenido

Introducción	2
Concepto	3
Notaciones	3
Escritas	4
Diagramas	5
Componentes	5
¿Cómo leerlos?	5
Pasos para crear un diagrama de sintaxis	7
Ejemplos	8
Ventajas de los diagramas de sintaxis	8
Aplicaciones de los diagramas de sintaxis	9
Conclusión	9
Bibliografía	10

# Diagramas de sintaxis

Los diagramas sintácticos, son una representación gráfica utilizada para describir la gramática libre de contexto de un lenguaje formal. Estos diagramas son una alternativa visual a las notaciones escritas. (Gramática de Forma de Backus-Naur BNF y Gramática Extendida de Backus-Naur EBNF) las cuales son ampliamente utilizadas para definir la estructura de los lenguajes de programación. Su principal objetivo es simplificar la comprensión y el análisis de las reglas gramaticales que determinan cómo deben estructurarse las sentencias en un lenguaje determinado.

Además de ser una herramienta clave en la informática y la lingüística computacional, los diagramas de sintaxis son fundamentales para el diseño y análisis de compiladores. Su capacidad para representar gráficamente las secuencias de reglas gramaticales facilita tanto el análisis sintáctico como la construcción de compiladores eficientes y robustos. En un compilador, estos diagramas juegan un papel crucial al ilustrar cómo los elementos léxicos de un programa deben ordenarse para cumplir con las reglas gramaticales del lenguaje.

Una de las principales ventajas de los diagramas de sintaxis, es que proporcionan una forma visual clara e intuitiva para comprender las construcciones válidas de un lenguaje. Al ofrecer una representación gráfica de la gramática de un lenguaje de programación,

## Concepto

Los diagramas de sintaxis son representaciones gráficas de las reglas gramaticales de un lenguaje. Estos diagramas muestran cómo se organizan las secuencias de símbolos (palabras clave, identificadores, operadores, entre otros) para formar estructuras válidas.

Utilizando círculos, rectángulos y flechas, los diagramas ilustran las opciones y secuencias posibles, facilitando la comprensión de las reglas gramaticales. Estos símbolos pueden ser terminales (elementos básicos) o no terminales (estructuras más complejas), y se conectan en el diagrama para mostrar cómo deben ordenarse en una sentencia válida.

## Notación

En el análisis sintáctico, se utilizan diversas notaciones y técnicas para representar y procesar las reglas gramaticales. Las más comunes incluyen:

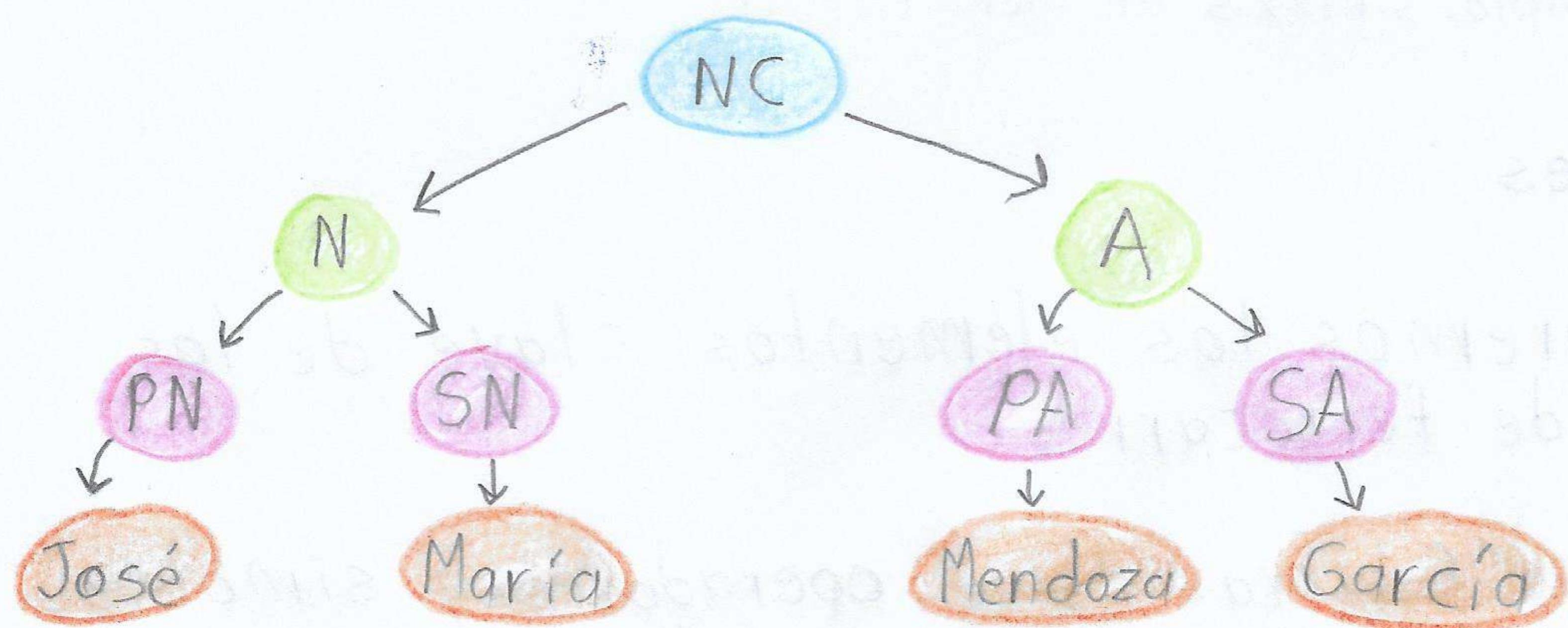
### Escritas:

- Gramática de forma de Backus-Naur (BNF)
  - Es una notación formal para expresar las reglas gramaticales de un lenguaje, utilizando símbolos como <> para denotar reglas y producciones.
- Gramática Extendida de Backus-Naur (EBNF)
  - Es una extensión de BNF que presenta una sintaxis más concisa y flexible.

## Diagramas:

### • Diagramas de Árbol Sintáctico:

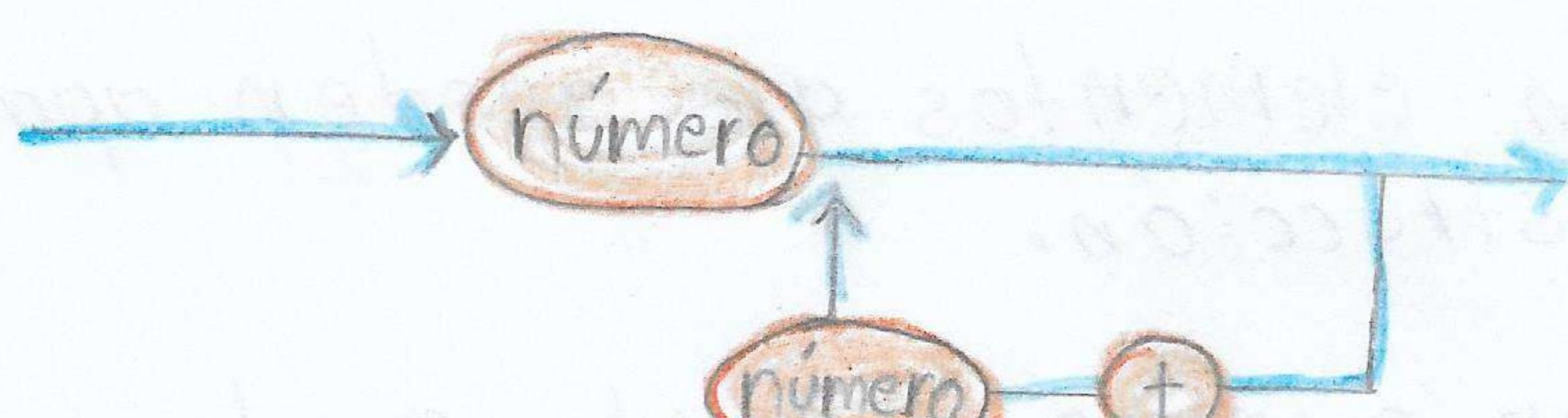
Los diagramas de árbol sintáctico son una forma común de visualizar la estructura de enunciado o expresión. Consiste en nodos que representan elementos gramaticales (como terminales o no terminales) y aristas que indican la relación entre ellos.



Ejemplo: Nombre completo.

### • Diagramas de Forma de flechas o Ferrocarril:

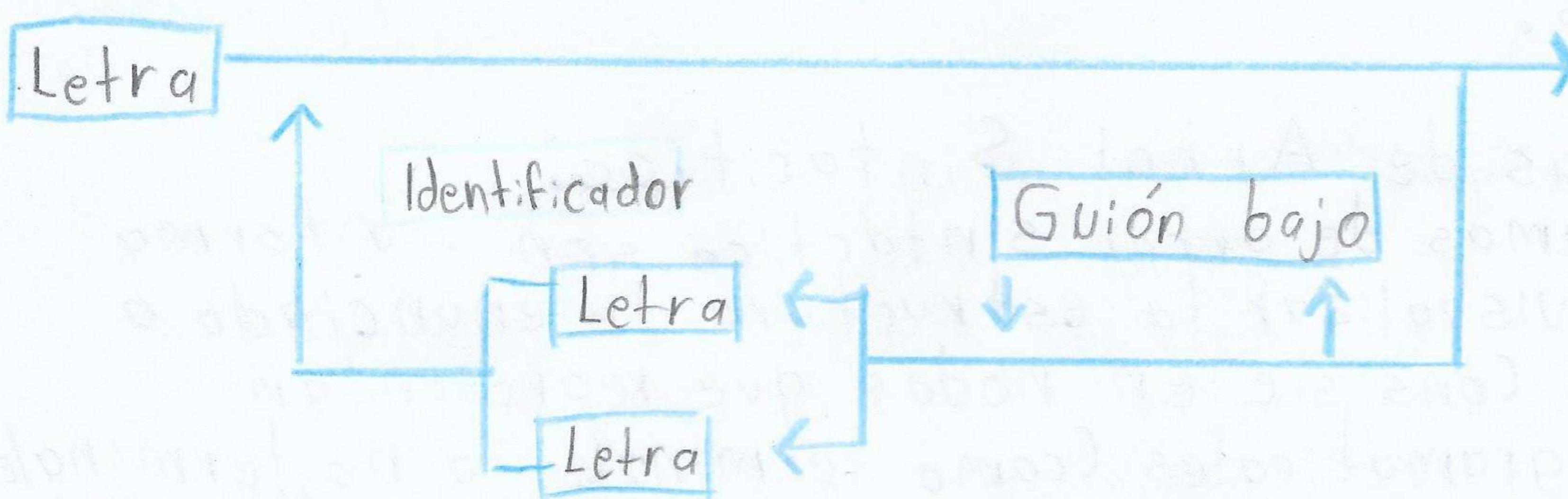
En esta notación, los elementos gramaticales se representan como cajas o elipses, y las flechas indican las relaciones entre ellos.



Ejemplo: sumar números

### • Diagramas de Conway:

Utilizan símbolos terminales (círculos o rectángulos) y símbolos no terminales (rectángulos) y flechas para enlazar.



Ejemplo: sintaxis de identificador

## Componentes

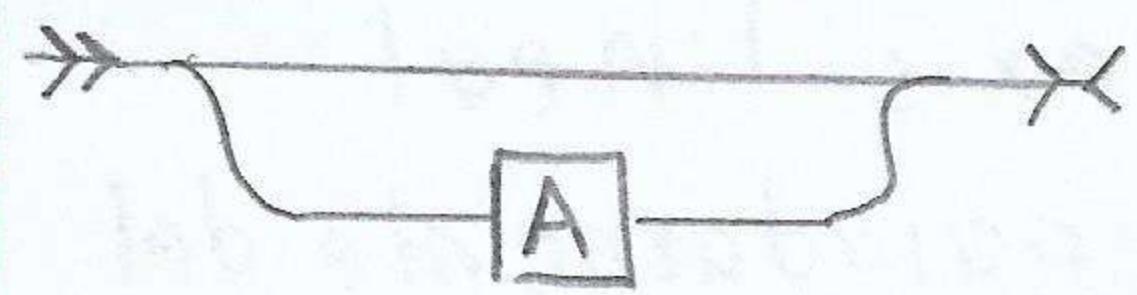
Aquí explicaremos los elementos clave de los diagramas de ferrocarril.

- **Terminales:** Palabras clave, operadores y símbolos que no pueden descomponerse en partes más pequeñas.
- **No terminales:** Conceptos abstractos que representan categorías de construcciones sintácticas.
- **Flechas:** Indican la dirección del análisis sintáctico y las secuencias de elementos.
- **Opciones:** Representan elementos que pueden aparecer o no en una construcción.
- **Repeticiones:** Indican que un elemento puede aparecer una o más veces.

## Como leerlos

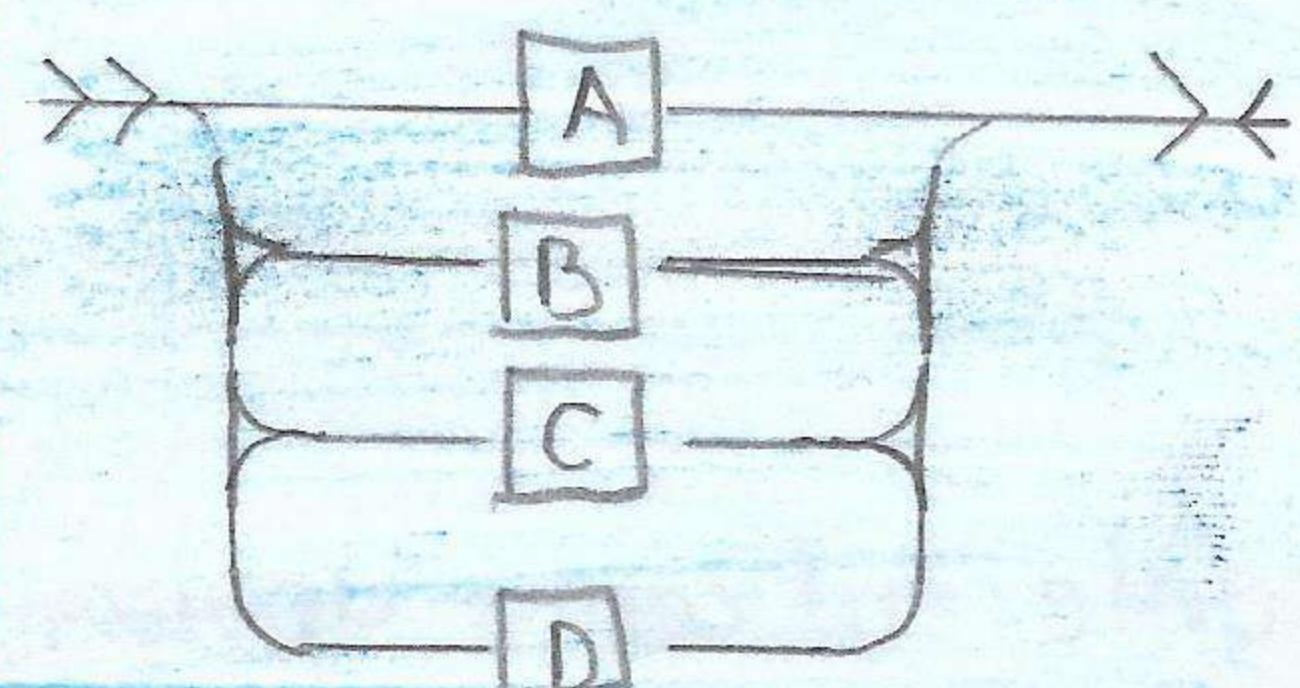
Convenio	Significado
$\Rightarrow A - B - C \rightarrow$	Debe especificar los valores A, B, C. Los valores necesarios se muestran en la línea principal de un diagrama de ferrocarril.

## Convenio

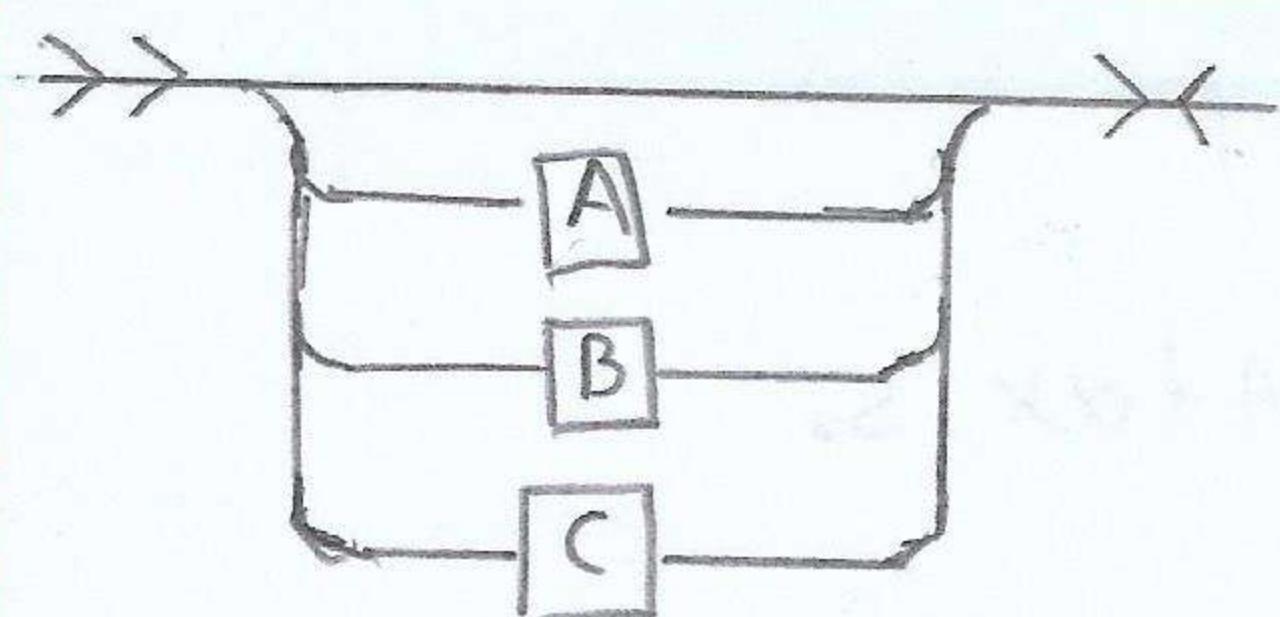


## Significado

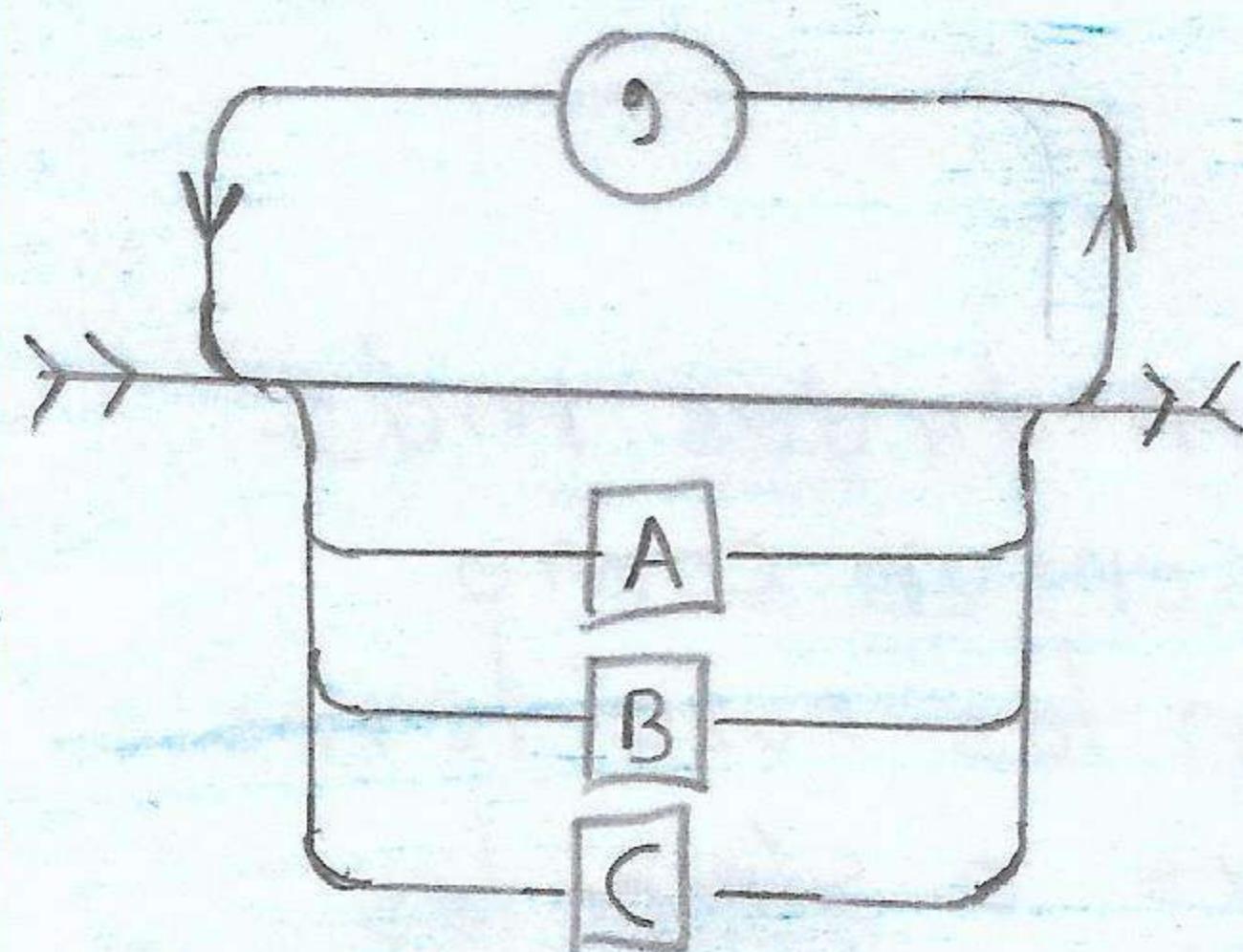
Puede especificar el valor A. Los valores optionales se muestran debajo de la linea principal de un diagrama de ferrocarril.



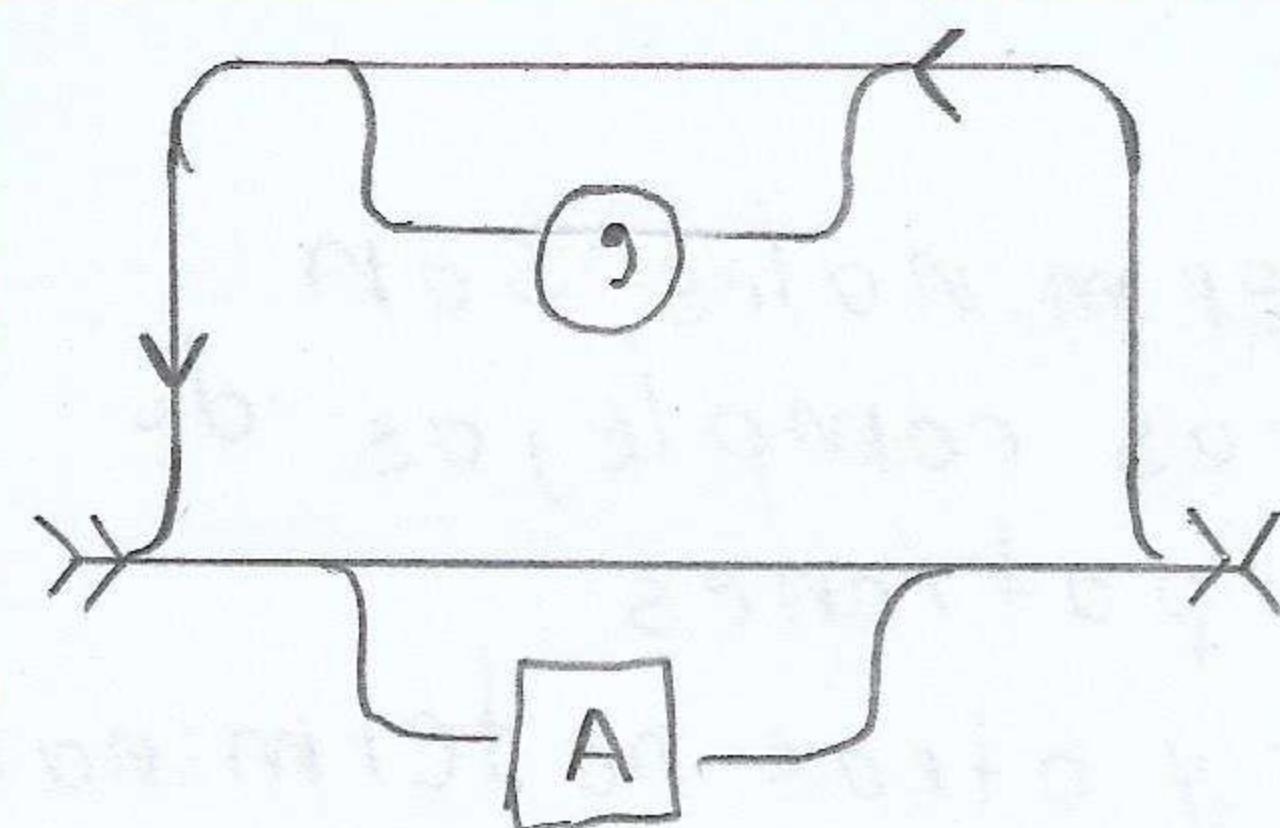
Los valores A, B, C y D son alternativas, una de las cuales debe especificar.



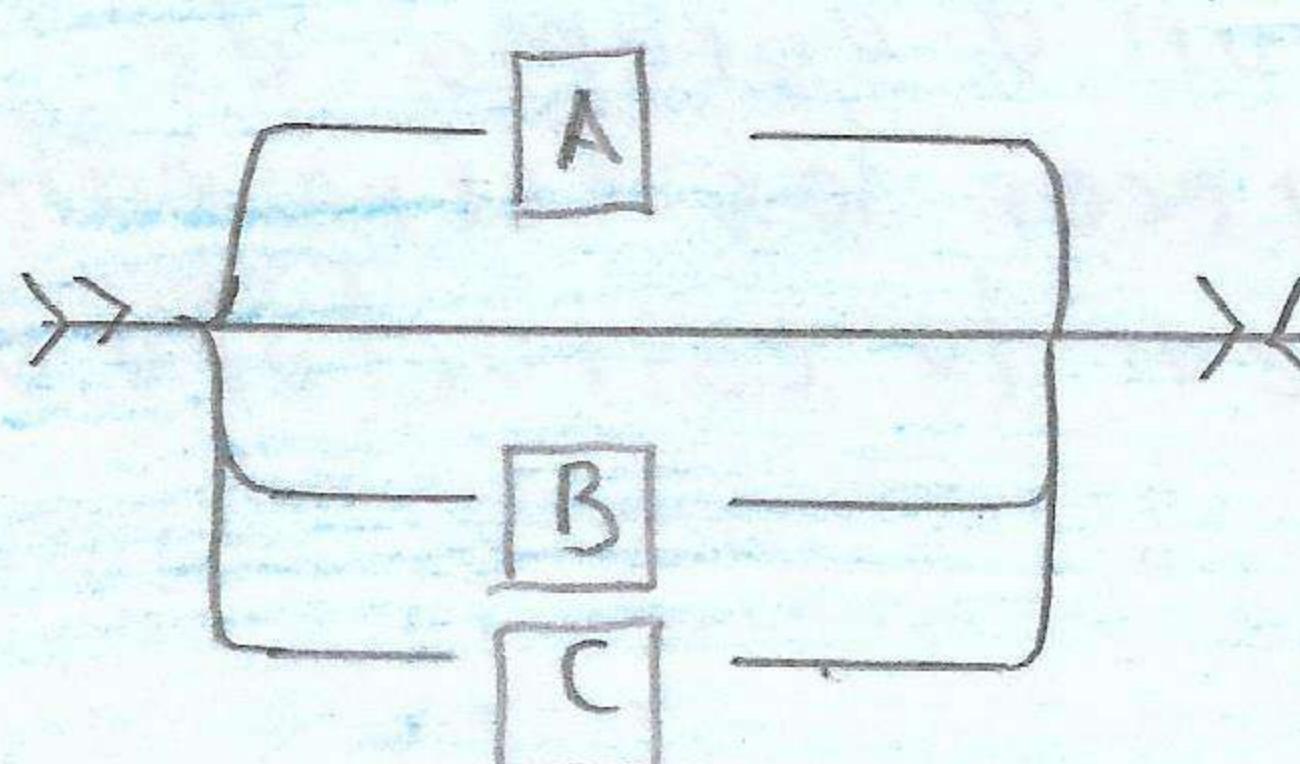
Los valores A, B y C son alternativas



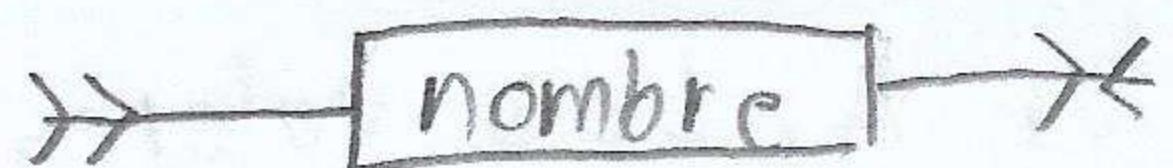
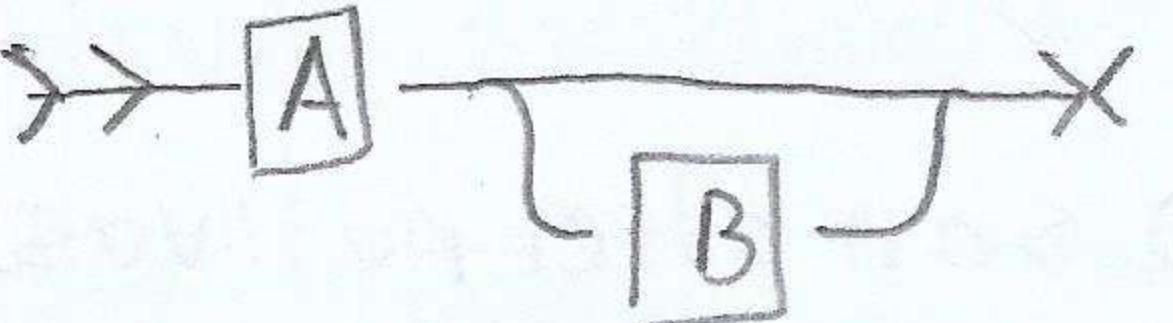
Esto muestra que se debe seleccionar un valor (por ejemplo A, B ó C) y si se va a seleccionar otros, se debe utilizar una coma (9) entre los valores.



Puede especificar el valor A varias veces. El separador en este ejemplo es opcional.



Los valores A, B y C son alternativas, una de las cuales pueden especificar. Si no especifica ninguno de los valores mostrados, se utiliza el valor predeterminado A (el valor mostrado por encima de la linea principal)

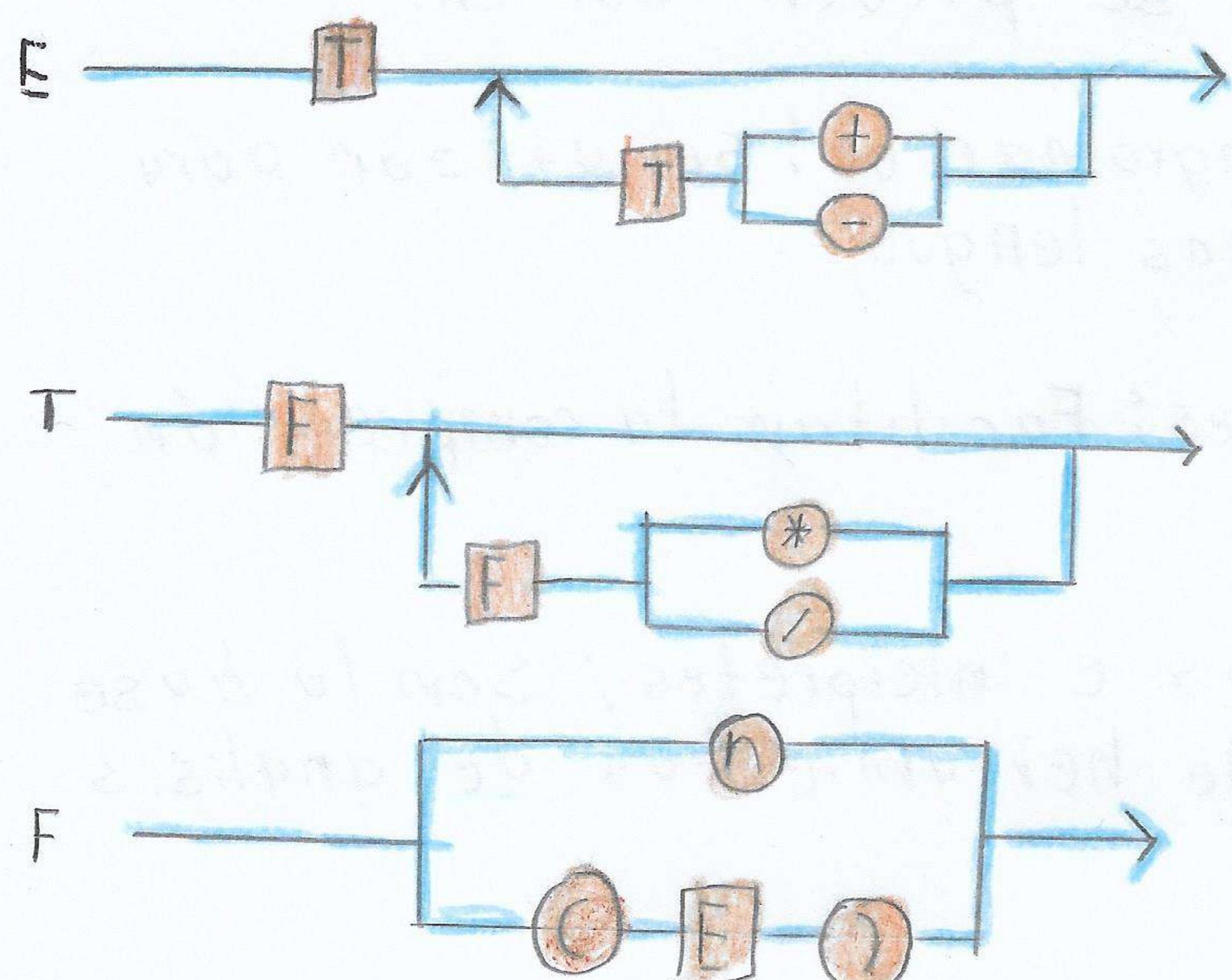
Convenio	Significado
 Nombre	El fragmento de sintaxis lineal nombre se muestra separadamente del diagrama de sintaxis linea principal.
 Valores de puntuacion	Especificar exactamente tal como aparecen.

## Pasos para crear un diagrama de sintaxis.

Los pasos para crear un diagrama de sintaxis de ferrocarril son los siguientes.

- 1- Identifica las **Fichas**: Las fichas son unidades más pequeñas de un lenguaje, a menudo representan como terminales en un diagrama de sintaxis. Son los caracteres literarios del lenguaje, como dígitos, letras o símbolos especiales
- 2- Determina tus **No-terminales**: Los no terminales son representaciones simbólicas de estructuras complejas de tu lenguaje. Suelen representar reglas o patrones compuestos por una serie de terminales y otros no terminales.
- 3- Define las **secuencias**: Las secuencias en un diagrama de sintaxis son el orden en que deben aparecer los terminales y los no terminales. Son útiles para esbozar la estructura de tu lenguaje.
- 4- Crea **Bucles** para la repetición: En situaciones en las que un patrón puede ocurrir varias veces, los bucles son beneficiosos. Permiten que una secuencia se repita.

# Ejemplos



$E ::= E + T \mid E - T$
$T ::= T * F \mid T / F$
$F ::= (E) \mid n$

## Ventajas de los diagramas de Sintaxis

El uso de estos diagramas en el diseño de compiladores ofrece varias ventajas. Algunas de las más importantes incluyen

- **Claridad Visual:** Los diagramas permiten visualizar de manera clara la estructura gramatical de un lenguaje.
- **Simplificación de reglas complejas:** Las reglas pueden parecer complejas al expresarse en forma textual.
- **Detección de errores:** Al permitir una representación visual del lenguaje, los diagramas de sintaxis facilitan la detección de inconsistencias o errores.
- **Herramienta Didáctica:** Son una excelente herramienta para enseñar conceptos de gramática y sintaxis.

## Aplicaciones de los diagramas de Sintaxis

Los diagramas de sintaxis se pueden usar en:

- **Diseño de lenguajes de programación:** Se utilizan para definir la sintaxis de nuevos lenguajes.
- **Documentación de lenguajes:** Facilitan la comprensión de lenguajes existentes.
- **Desarrollo de compiladores e intérpretes:** Son la base para la construcción de herramientas de análisis sintáctico.
- **Educación:** Se utilizan para enseñar los fundamentos de la sintaxis de los lenguajes de programación.

## Conclusión

El análisis sintáctico y los diagramas de sintaxis son herramientas esenciales tanto en la informática como en lo lingüístico computacional. Los diagramas de Sintaxis ofrecen una representación visual clara y comprensible de las reglas gramaticales de un lenguaje formal, facilitando su análisis y comprensión. Permitiendo a los diseñadores de lenguajes y compiladores visualizar la estructura jerárquica de un lenguaje.

## Referencias

Aguilera Sierra María del Mar, S. G. R. (s/f). Traductores, Compiladores e Intérpretes. Uma.es. Recuperado el 15 de septiembre de 2024, de  
<http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf>

Cruz, J. G. D. (s/f). TEMA 5. ANÁLISIS SINTÁCTICO. Genially. Recuperado el 15 de septiembre de 2024, de <https://view.genially.com/65dce529d9791d0013591890/presentation-tema-5-analisis-sintactico>

Diagrama de Sintaxis. (s/f). StudySmarter ES. Recuperado el 16 de septiembre de 2024, de <https://www.studysmarter.es/resumenes/ciencias-de-la-computacion/teoria-de-la-computacion/diagrama-de-sintaxis/>

Diagramas de Conway. (2016, noviembre 17). studylib.es.  
<https://studylib.es/doc/6023051/diagramas-de-conway>

IBM MQ 9.4. (2024, julio 2). Ibm.com. <https://www.ibm.com/docs/es/ibm-mq/9.4?topic=reference-how-read-syntax-diagrams>

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

DOCENTE DESIGNADO
ISC. RICARDO GONZÁLEZ GONZÁLEZ

PRACTICA No.	NOMBRE DE LA PRACTICA	DURACIÓN ( HORAS )
2	Gramáticas Libres de Contexto	6 horas

1

### INTRODUCCIÓN

Para el propósito de esta práctica se buscará el explorar el concepto de gramáticas libres de contexto, desde su definición para los lenguajes formales, así como entender sus características y ejemplos prácticos para poder exemplificar de mejor manera el entendimiento de dicho concepto así como comprender el cómo los árboles de derivación desempeñan un papel crucial para el análisis sintáctico.

A si mismo como se mencionó anteriormente se buscará consolidar el conocimiento acerca del análisis sintáctico.

2

### OBJETIVO ( COMPETENCIAS )

Comprender el concepto de gramáticas libres de contexto, abarcando sus características y a través de ejemplos prácticos demostrar su importancia en los lenguajes formales, así como aplicar este conocimiento en los árboles de derivación y el análisis sintáctico.

**3**

### **MARCO TEÓRICO REFERENCIAL**

Las gramáticas libres de contexto también conocidas como GLC son un tipo de gramática formal que es utilizada en la teoría de lenguajes y autómatas, así como en la implementación de lenguajes de programación.

Sus principales propiedades son:

- Libres de contexto: las producciones en las GLC dependen solo del símbolo no terminal y no del contexto en el que aparece.
- Lenguajes generados: los lenguajes que se generan por GLC se les conoce como libres de contexto y son un subconjunto importante en la jerarquía de Chomsky, así como también son utilizados en la definición de sintaxis de lenguajes de programación.

Las GLC tienen como principales aplicaciones las siguientes:

- Compiladores: las GLC son usadas ampliamente en el análisis sintáctico para interpretar la estructura de un programa o expresión matemática.
- Analizadores sintácticos: algoritmos como el CYK y parsers LL y LR son basados en GLC para poder validar la sintaxis de las expresiones de entrada.

Los árboles de derivación son una representación gráfica de cómo se derivan cadenas de un lenguaje a partir de la gramática formal. En las GLC, los árboles de derivación ilustran el proceso de generación de una cadena mediante la aplicación de reglas de producción de la gramática.

Su estructura viene dada de la siguiente manera:

- Tiene como raíz el símbolo inicial de la gramática.
- Los nodos internos en el árbol representan los símbolos no terminales.
- Los hijos de un nodo representan los símbolos en el lado derecho de la regla de producción que se aplica al no terminal del nodo.
- Las hojas son símbolos terminales, estos forman la cadena generada por la gramática al leerlo de izquierda a derecha.

La relación entre GLC y árboles de derivación viene dado de que las GLC definen las reglas para construir las cadenas de un lenguaje, por su parte los árboles de derivación muestran el como se van aplicando esas reglas paso a paso.

**4**

### **MATERIALES UTILIZADO**

- Equipo de cómputo.
- Navegador Web para consulta de información.
- Libros o videos referentes al tema.

5

**REQUISITOS BÁSICOS.**

- Sistema operativo: Windows 2000 / 98 / XP / Vista / 7 / 8 / 10 / 11

6

**DESARROLLO DE LA PRÁCTICA ( PASO A PASO )**



## ¿Qué son las gramáticas libres de contexto?

Las gramáticas libres de contexto (GLC) son un sistema formal el cual describe un conjunto de reglas de producción usadas para generar cadenas o también conocidas como secuencias de símbolos los cuales pertenecen a un lenguaje.

Las GLC se definen por cuatro elementos:

- **No terminales (V):** Son un conjunto de símbolos que pueden ser reemplazados por otros símbolos. Son usados por las reglas de producción para definir la estructura sintáctica de las cadenas en el lenguaje y representan categorías gramaticales abstractas.
- **Terminales ( $\Sigma$ ):** Son un conjunto de símbolos que no se pueden reemplazar (se podría decir que son las “palabras del lenguaje”).
- **Símbolo inicial (S):** Es el símbolo no terminal desde donde se comienzan las derivaciones.
- **Reglas de producción (P):** Son reglas que indican como un símbolo no terminal puede ser reemplazado por una secuencia de símbolos terminales y no terminales.

Una característica clave de las GLC es que las reglas de producción tienen la forma:  $A \rightarrow \alpha$ , Donde:

- A es un símbolo no terminal.
- $\alpha$  es una secuencia de símbolos terminales y no terminales.

El que tengan libertad de contexto se refiere a que el símbolo no terminal A se puede reemplazar independientemente del contexto en el que se encuentre, lo que de forma simple quiere decir que no depende de los símbolos que lo rodean.

**Ejemplo simple de GLC:** Un ejemplo básico para ejemplificar las GLC es una que genere las cadenas con el mismo número de ‘a’s seguidas del mismo número de ‘b’s, de esta manera: ab, aabb, aaabbb.

- **Símbolos no terminales (V):** S (símbolo inicial).
- **Símbolos terminales ( $\Sigma$ ):** a, b.
- **Símbolo inicial (S):** S.
- **Reglas de producción (P):**  $S \rightarrow aSb$ ,  $S \rightarrow \epsilon$  (donde  $\epsilon$  representa la cadena vacía).

En esta gramática se generan cadenas que comienzan con una o más ‘a’s, seguidas por el mismo número de ‘b’s como se dijo anteriormente.

La primera regla de producción  $S \rightarrow aSb$  genera una ‘a’ al inicio y una ‘b’ al final para mantener el patrón. La segunda regla  $S \rightarrow \epsilon$  detiene la derivación al ya no haber más símbolos que agregar.

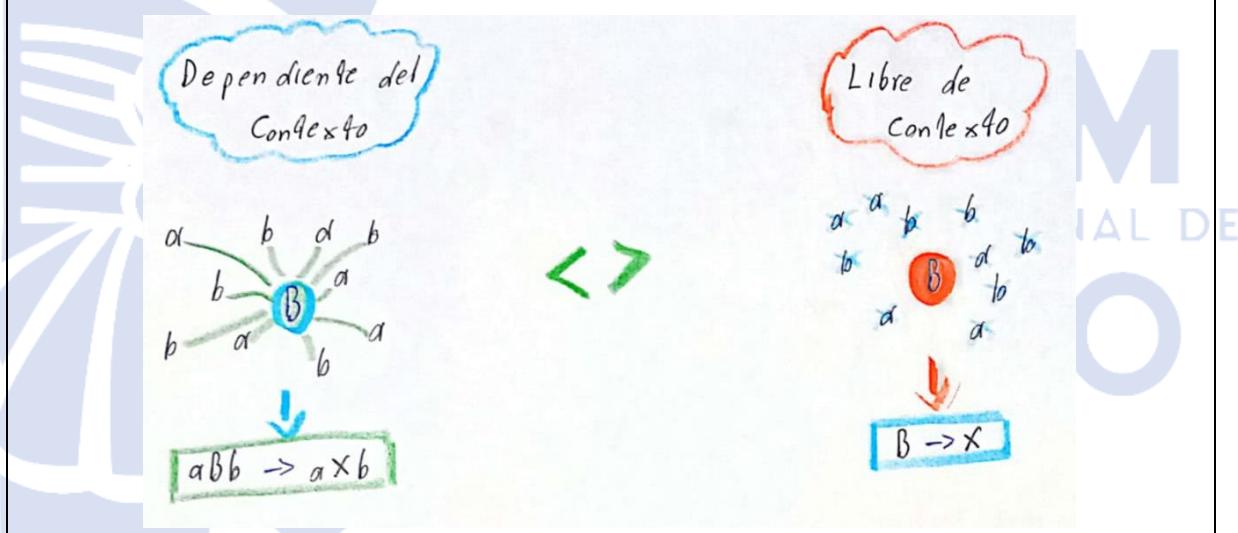
### ¿Qué es un contexto?

De manera general pero no por eso alejado del punto de las gramáticas, el contexto es todo aquello que rodea y afecta a una palabra u oración que se da en una conversación. El contexto permite entender adecuadamente lo que está ocurriendo o comunicando.

En el ejemplo del lenguaje el contexto son las palabras que rodean una expresión, y en situaciones mas amplias puede tratarse del entorno físico, las relaciones sociales, tiempo y lugar.

El termino contexto en la teoría de lenguajes formales hace referencia a los elementos que rodean a un símbolo o secuencia de símbolos en una expresión, dicho de otra manera, es la información alrededor de una palabra o frase y que influye en el significado y función gramatical. Entender el contexto es primordial para comprender como funcionan las palabras en una oración, ya que dependiendo de este las palabras tendrán un significado totalmente distinto dependiendo del contexto.

Para una gramática dependiente del contexto, las reglas de producción varían dependiendo de los símbolos que se encuentren antes o después del símbolo a sustituir. Mientras que, en las gramáticas libres de contexto, las reglas de producción no dependen del contexto el símbolo a derivar o sustituir.



**Ejemplo de contexto:** Supongamos que se tiene una regla de producción en una gramática dependiente del contexto:  $aBb \rightarrow abb$ ; Se observa que el contexto de B es determinado por los símbolos a y b, lo que significa que solo se puede aplicar esta regla si B está rodeada por dichos símbolos.

### ¿Sirve dicho contexto a los propósitos de una gramática que defina un lenguaje formal?

El contexto en si es de suma importancia si se quiere definir la gramática de un lenguaje formal, esto debido a que gracias a este se establecen las reglas sintácticas, semánticas y pragmáticas que rigen la construcción e interpretación de las expresiones lingüísticas. Entonces una gramática que defina un lenguaje formal tiene que tomar en cuenta al contexto en el que se produce y se usa el lenguaje.

Es tan importante el contexto para una gramática que defina un lenguaje formal debido a que este proporciona todos los criterios necesarios para describir y explicar tanto funcionamiento como la estructura de un lenguaje.

Pensemos de forma en que el contexto influye demasiado en el significado y función de alguna expresión del lenguaje formal. En el caso de programadores por ejemplo el contexto puede determinar el uso de variables, constantes, símbolos, conectores y operadores. En este caso también gracias al contexto se facilita comprender un lenguaje formal, es como tener un manual que evite que nos perdamos entre las reglas que rigen a el lenguaje.

Con todo lo anterior podemos retomar la parte de las GLC, ya que si bien en estas se tiene un enfoque libre de contexto el cual permite la simplificación de los lenguajes y hace que la gramática sea más fácil de analizar y procesar para permitir a su vez un análisis sintáctico más eficiente. No se debe considera al contexto como algo para dejar tan de lado ya que este como hemos venido hablando proporciona los criterios necesarios para describir y explicar cómo funciona un lenguaje.

### Ejemplos de árboles de derivación

1. Gramática para expresiones aritméticas con operadores de suma y multiplicación.

#### Definición de gramática

**Símbolos no terminales:** E (Expresión), T (Término), F (Factor).

**Símbolos terminales:** +, \*, (,), id (identificador).

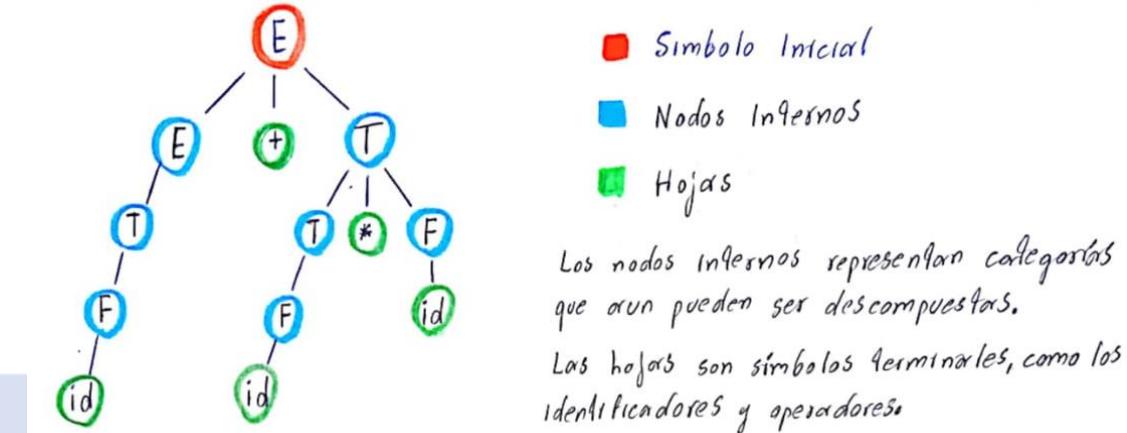
Reglas de producción:

- $E \rightarrow E+T$
- $E \rightarrow T$
- $T \rightarrow T*F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow id$

Derivación de la cadena **id + id \* id**

#### Pasos de la derivación:

1.  $E \rightarrow E+T$
2.  $E \rightarrow T+T$
3.  $T \rightarrow F+T$
4.  $F \rightarrow id$
5.  $T \rightarrow T*F$
6.  $T \rightarrow F*F$
7.  $F \rightarrow id$
8.  $F \rightarrow id$



Para el ejemplo anterior se puede observar como el árbol de derivación ilustra la precedencia de operadores: ya que la multiplicación tiene mayor precedencia que la suma se deriva primero el subárbol de  $T * F$  antes de completar la suma de  $E + T$ .

- Gramática para sentencias condicionales anidadas.

#### Definición de gramática

**Símbolos no terminales:** S (Sentencia), C (condición), B (Bloque).

**Símbolos terminales:** if, then, else, id, {}, .

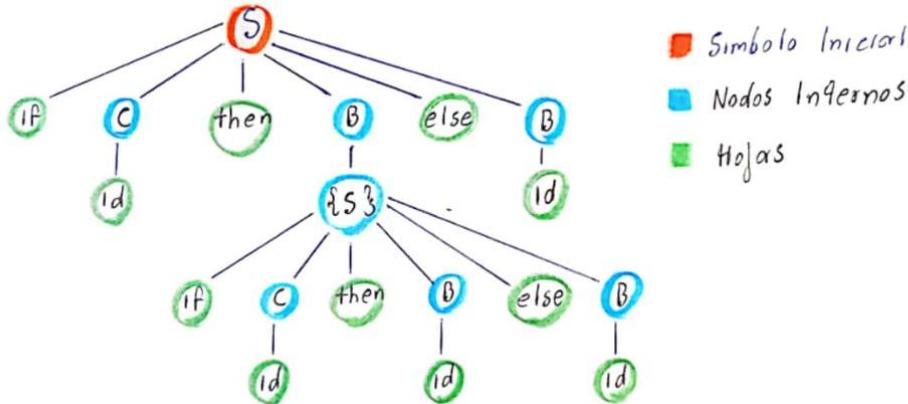
Reglas de producción:

- $S \rightarrow \text{if } C \text{ then } B \text{ else } B$
- $C \rightarrow \text{id}$
- $B \rightarrow S$
- $B \rightarrow \text{id}$

Derivación de la cadena **if id then { if id then id else id } else id**

#### Pasos de la derivación:

1.  $S \rightarrow \text{if } C \text{ then } B \text{ else } B$
2.  $C \rightarrow \text{id}$
3.  $B \rightarrow S$
4.  $S \rightarrow \text{if } C \text{ then } B \text{ else } B$
5.  $C \rightarrow \text{id}$
6.  $B \rightarrow \text{id}$
7.  $B \rightarrow \text{id}$



Los nodos internos representan las sentencias, condiciones y bloques que se descomponen en componentes más específicos.

Las hojas son palabras clave, identificadores y delimitadores.

En el árbol de derivación del ejemplo anterior se puede observar que muestra como las sentencias condicionales se anidan dentro de un bloque, reflejando así la estructura jerárquica de las sentencias if-then-else.

- Gramática para listas de parámetros en una llamada de función.

#### Definición de gramática

**Símbolos no terminales:** L (Lista), P (Parámetro).

**Símbolos terminales:** ',', id.

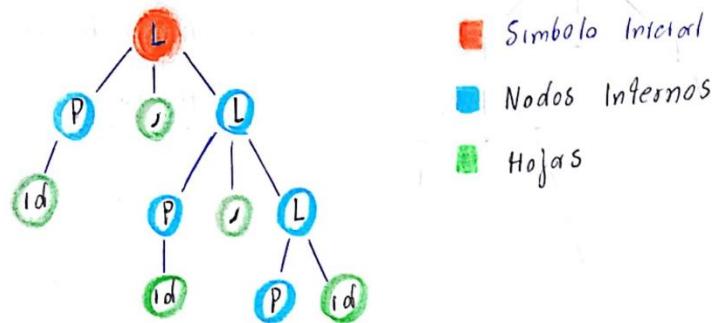
Reglas de producción:

- $L \rightarrow P, L$
- $L \rightarrow P$
- $P \rightarrow id$

Derivación de la cadena **id, id, id**

#### Pasos de la derivación:

1.  $L \rightarrow P, L$
2.  $P \rightarrow id$
3.  $L \rightarrow P, L$
4.  $P \rightarrow id$
5.  $L \rightarrow P$
6.  $P \rightarrow id$



Los nodos internos son subdivisiones que se siguen descomponiendo en parámetros y listas.

Las hojas son identificadores y los comas que separan los elementos de la lista.

En el ejemplo anterior se observa que el árbol de derivación ilustra una lista de parámetros de longitud 3. Cada vez que se aplica la regla  $L \rightarrow P, L$ , se agrega un parámetro a la lista, esto refleja la naturaleza recursiva de listas de parámetros en varios lenguajes de programación.

5

**BITÁCORA DE INCIDENCIAS**

PROBLEMA	FECHA	HORA	SOLUCIÓN	FECHA	HORA
No saber hacer un árbol de derivación.	14/09/2024	2:00 p.m.	Consulta de algunas fuentes con ejemplos.	14/09/2024	2:30

6

**OBSERVACIONES**

En el caso de las GLC se observó que, a pesar de ser libres de contexto, el contexto en si sigue siendo relevante para poder comprender de manera adecuada un lenguaje haciendo así que sea importante el considerar la estructura sintáctica como el significado semántico en el diseño de lenguajes.

Si bien antes no se tenían conocimientos relevantes sobre arboles de derivación se observó que es una herramienta importante dado a la capacidad de visualizar a través de estos el como funciona la estructura de las expresiones que genera una GLC, además estos mismos arboles permiten la creación de ejemplos mas complejos ya que al tenerlos como ayuda visual es muy fácil comprender como se realiza la derivación de una cadena.

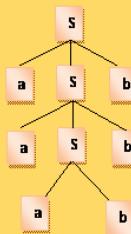
7

**ANEXOS**

Como parte relevante para el desarrollo de esta práctica se hizo uso de ejemplos como el siguiente para poder comprender el cómo se realizaban los árboles de derivación, dado que es un conocimiento nuevo el consultar ejemplos ayudo a comprender mejor la estructura y creación de árboles de derivación. Un ejemplo consultado fue el siguiente:

**Árbol de derivación. Ejemplo**  
 Sea  $G=(N, T, S, P)$  una GLC con  $P: S \rightarrow ab|aSb$

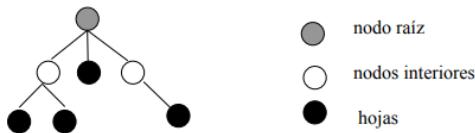
La derivación de la cadena  $aaabbb$  será:  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$   
 y el árbol de derivación:



También para entender aún mejor la estructura y su creación este ejemplo fue relevante:

El árbol de derivación tiene las siguientes propiedades:

- el nodo raíz está rotulado con el símbolo distinguido de la gramática;
- cada hoja corresponde a un símbolo terminal o un símbolo no terminal;
- cada nodo interior corresponde a un símbolo no terminal.



7

REFERENCIAS BIBLIOGRÁFICAS

- Alejandra, M. (n.d.). Arboles de derivaciòn.  
<https://teodelacomp.blogspot.com/2011/03/arboles-de-derivacion.html>
- Árboles de derivación. (n.d.). Lenguajes Formales Y Autómatas.  
[https://ivanvladimir.gitlab.io/lfy\\_a\\_book/docs/04abropar%C3%A9ntesisabropar%C3%A9ntesiscierrepar%C3%A9ntesis/05arbolesdederivaci%C3%B3n/](https://ivanvladimir.gitlab.io/lfy_a_book/docs/04abropar%C3%A9ntesisabropar%C3%A9ntesiscierrepar%C3%A9ntesis/05arbolesdederivaci%C3%B3n/)
- Platzi. (n.d.). ¿Qué es un contexto? [Video]. /Clases/1928-curso-de-thick-data/29194-que-es-un-contexto/. <https://platzi.com/clases/1928-curso-de-thick-data/29194-que-es-un-contexto/>
- Autor desconocido. (2008). Gramáticas libres del contexto. Universidad Nacional del Centro de la Provincia de Buenos Aires.  
<https://users.exa.unicen.edu.ar/catedras/ccomp1/Apunte5.pdf>



## Videos

<https://youtu.be/KNvo80XXJbo?si=b5w74I1H7X3oQHff>

<https://youtu.be/mG355IMkvWg?si=YUENgNH3WurWQ4h1>