



# Análisis sintáctico

Ing. Karina Cabrera Chagoyan



# Función

Entrada: sucesión de componentes léxicos

Salida: la entrada es o no correcta  
sintácticamente

Además:

- ➡ si la entrada es correcta, árbol de derivación
- ➡ si no indicación de errores

## Uso de gramáticas independientes de contexto

- Especificación precisa y fácil de entender
- Construcción automática de analizadores sintácticos
- Imparten una estructura al programa fuente, a la que se asigna el significado. Detección adecuada de errores
- Fácil mantenimiento: evolución del lenguaje

<programa> → <cabecera> <bloque>

<cabecera> → program <identificador>;

<cabecera> → program <identificador> (<ids.archivos>);

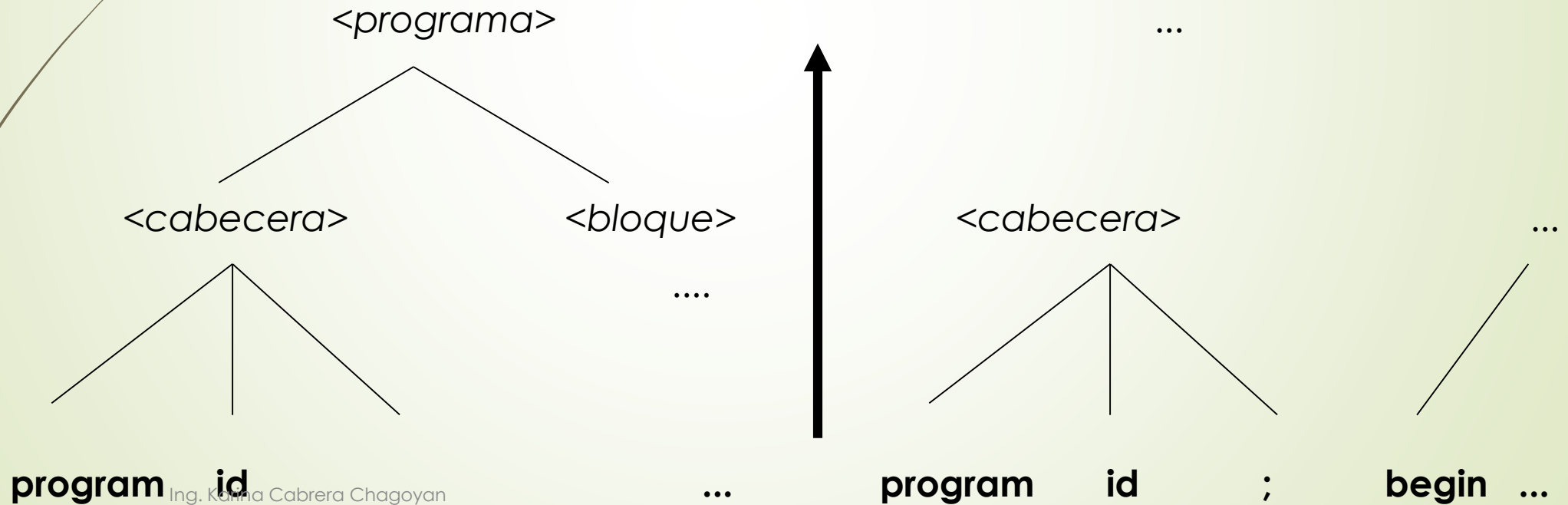
<bloque> → <declaraciones y definiciones de entorno>

    <def. de subprogramas> <sent. secuencial>.

<sent. secuencial> → begin <sentencias> end

# Tipos de análisis sintácticos

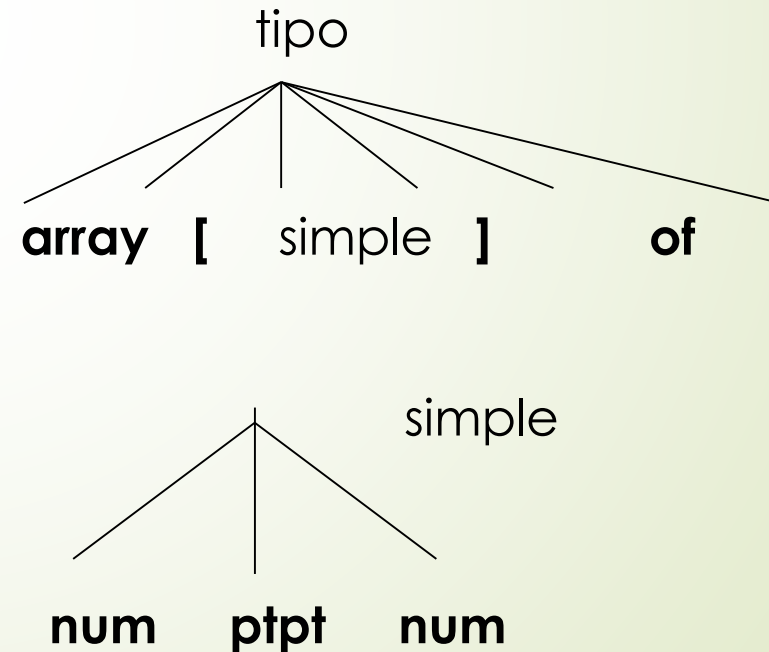
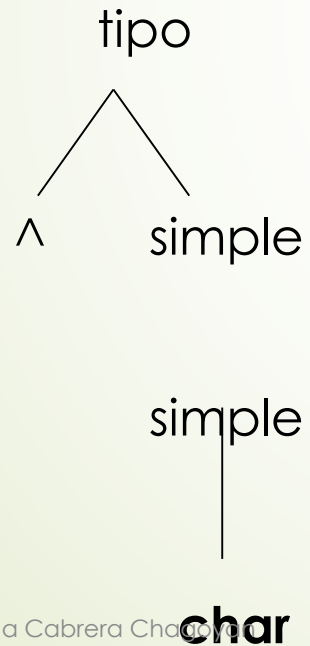
- Descendentes. Subclases de gramáticas adecuadas (LL)
- Ascendentes. Subclases de gramáticas adecuadas (LR)



# Análisis sintáctico descendente

*tipo* → *simple*  
|  $\wedge$  *simple*  
| **array** [*simple*] **of** *tipo*

*simple* → **integer**  
| **char**  
| **num ptpt num**





# Diagramas de sintaxis

- Las producciones de una gramática también se pueden representar en forma grafica, utilizando DIAGRAMAS DE SINTAXIS.
- Un diagrama de sintaxis se define como una herramienta útil para representar, de manera sencilla, el orden que deben seguir los elementos del lenguaje.
- La aplicación principal de los mismos, como su nombre lo indica, es representar los patrones sintácticos de los lenguajes, utilizando para ello uno o varios diagramas, dentro de los cuales debe existir uno que lleve el nombre del símbolo sentencial de la gramática

# Notación

<N>

Nombre del diagrama de sintaxis

N

Referencia a un diagrama de sintaxis



Indica la secuencia del orden a seguir

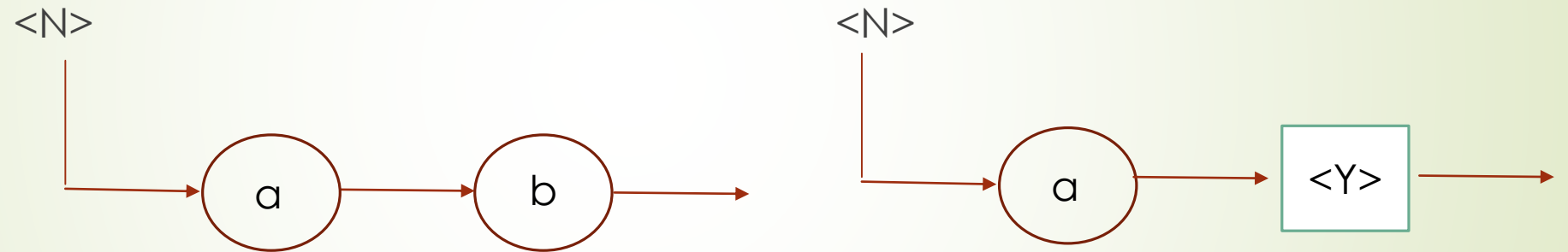
†

Referencia a un símbolo terminal



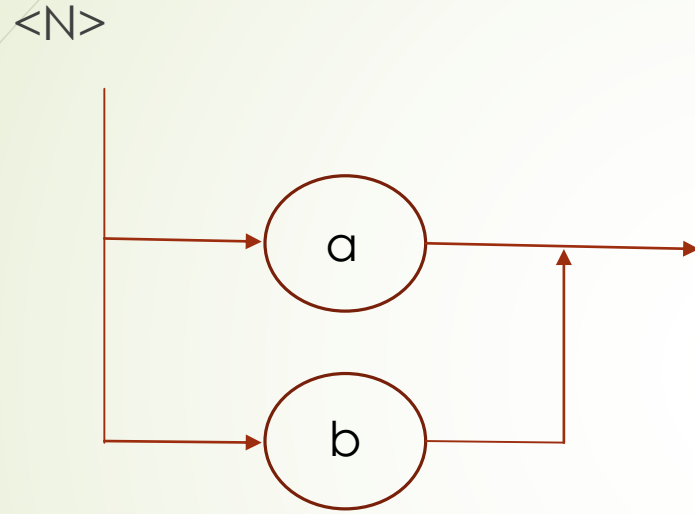
# Patrones sintácticos comunes

## ▀ A) secuencia de símbolos

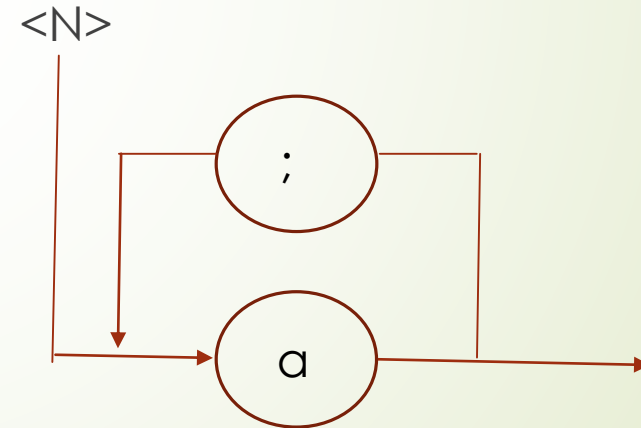
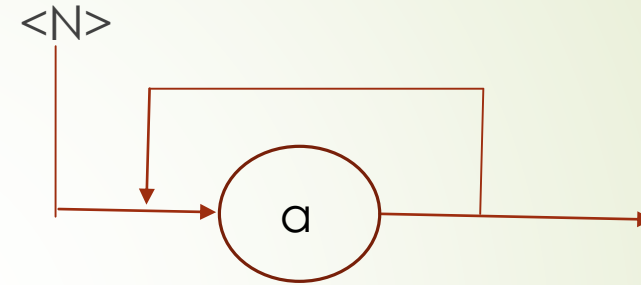




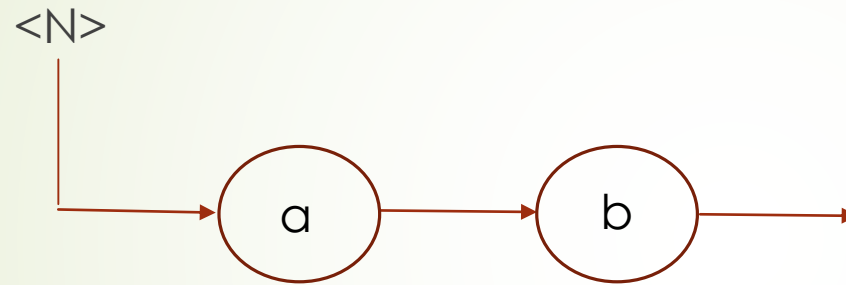
■ B) Alternativa entre varios símbolos



■ B) Estructura cíclica



# Estructura de secuencia



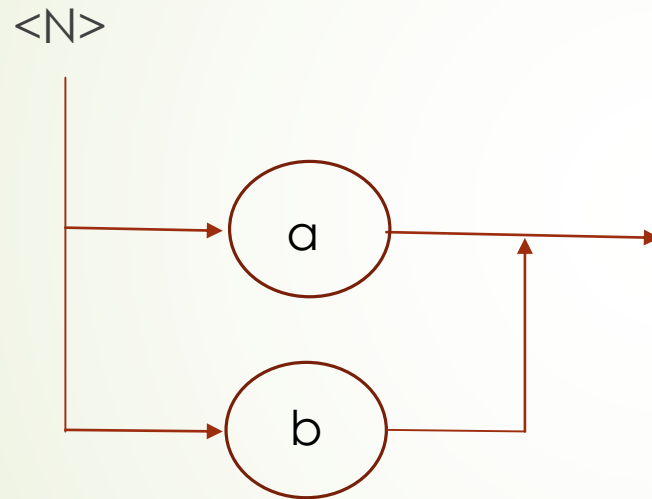
**Gramática equivalente**

$X \rightarrow a b$

**Expresión regular**

$X = a b$

# Estructura de alternativa



## Gramática equivalente

$N \rightarrow a$

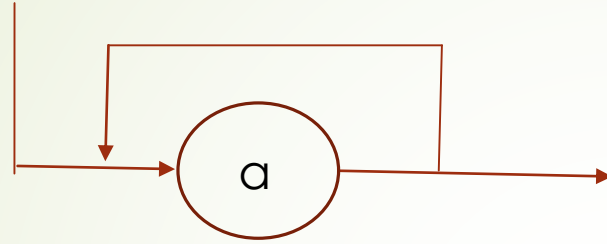
$N \rightarrow b$

## Expresión regular

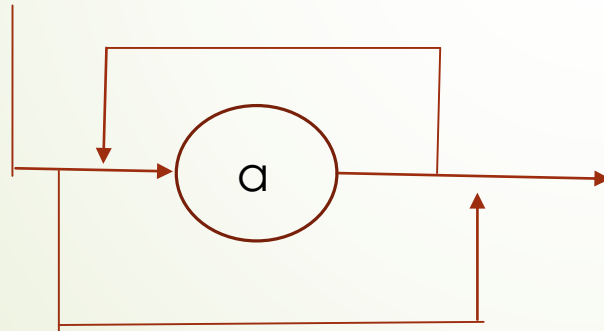
$N = (a \mid b)$

# Estructura cíclica

<N>



<N>



## Gramática equivalente

1)  $N \rightarrow N a$

$N \rightarrow a$

2)  $N \rightarrow a N$

$N \rightarrow a$

## Expresión regular

$N = a^+$

## Gramática equivalente

1)  $N \rightarrow N a$

$N \rightarrow \epsilon$

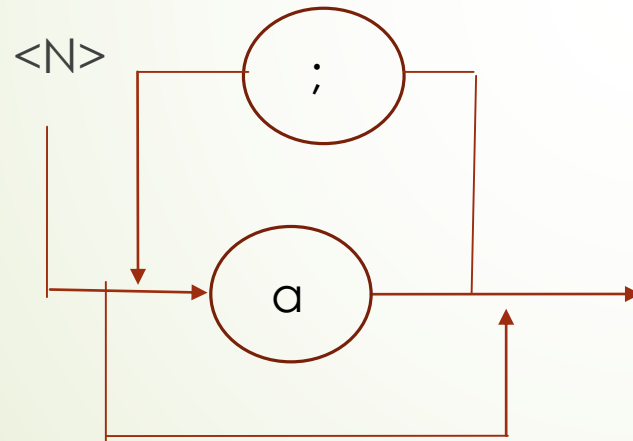
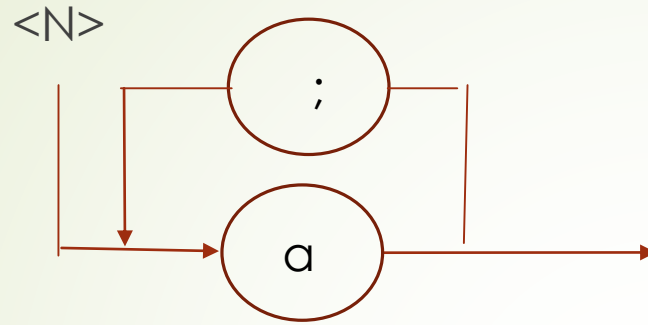
2)  $N \rightarrow a N$

$N \rightarrow \epsilon$

## Expresión regular

$N = a^*$

# Estructura cíclica con separador



## Gramática equivalente

1)  $N \rightarrow N ; a$

$N \rightarrow a$

2)  $N \rightarrow a ; N$

$N \rightarrow a$

## Expresión regular

$N = a ( ; a ) ^ *$

## Gramática equivalente

1)  $N \rightarrow a Y$

$N \rightarrow \epsilon$

$Y \rightarrow ; a Y$

$Y \rightarrow \epsilon$

## Expresión regular

$N = ( \epsilon \mid a ( ; a ) ^ * )$



# Método descenso recursivo

Este método trata de encontrar el árbol con la derivación de más a la izquierda para un string de entrada


Es la técnica mas sencilla que existe para realizar el análisis sintáctico, sin embargo requiere demasiada programación (genera muchas líneas de código).

## Esta técnica consiste en:

- **Implementar una rutina** (procedimiento o función) para cada símbolo no Terminal que se tenga en los diagramas de sintaxis
- **Programar un estatuto condicional** (IF-THEN-ELSE) para cada uno de los símbolos que aparezcan en los diagramas

El análisis comienza en el diagrama principal del lenguaje y va solicitando tokens al léxico conforme “acepta” el token que actualmente analiza. Si el token que envía léxico no era el esperado por la sintaxis se generará un error del tipo “ Se esperaba : \_\_\_\_\_ ”



- 
- El programa hace uso del procedimiento **AVANZA ()**, el cual se define como una llamada a la rutina LEXICO que genera como resultado el proximo “token” a analizar, dejandolo en la variable global **NEXT**.
  - Ademàs se utiliza el procedimiento **ERROR()** que se encarga de manejar los errores de sintaxis que se generan y el procedimiento **ACEPTAR()** que despliega aviso cuando el programa fuente cumpla satisfactoriamente con las reglas sintacticas del lenguaje

# Ejemplo

Suponga  $G =$   
 $\{(a,b,d,m,n), (X,A,B,C), (X), (P)\}$

$P: X \rightarrow ABC \mid dB \mid aC$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid C$

$C \rightarrow m \mid n$

Se deberá crear una rutina, procedimiento o función para cada símbolo No terminal (VN) de acuerdo a la producción(es) correspondiente, incluyendo la rutina que invoca a la raíz o símbolo sentencial

Ing. Karina Cabrera Chagoyan

RUTINA MODULO  
PRINCIPAL

< Principal >

Inicio

Avanza()

X

Si next = eof entonces

    Éxito ("Análisis exitoso")

Sino

    Error ("Esperaba eof")

Finsi

Fin

# Diagrama de sintaxis símbolo no terminal X

## RUTINA MODULO SIMBOLO NO TERMINAL X

<X>

Inicio

Si next = "d" entonces

Avanza()

B

Sino

Si next = "a" entonces

Avanza()

C

Sino

A

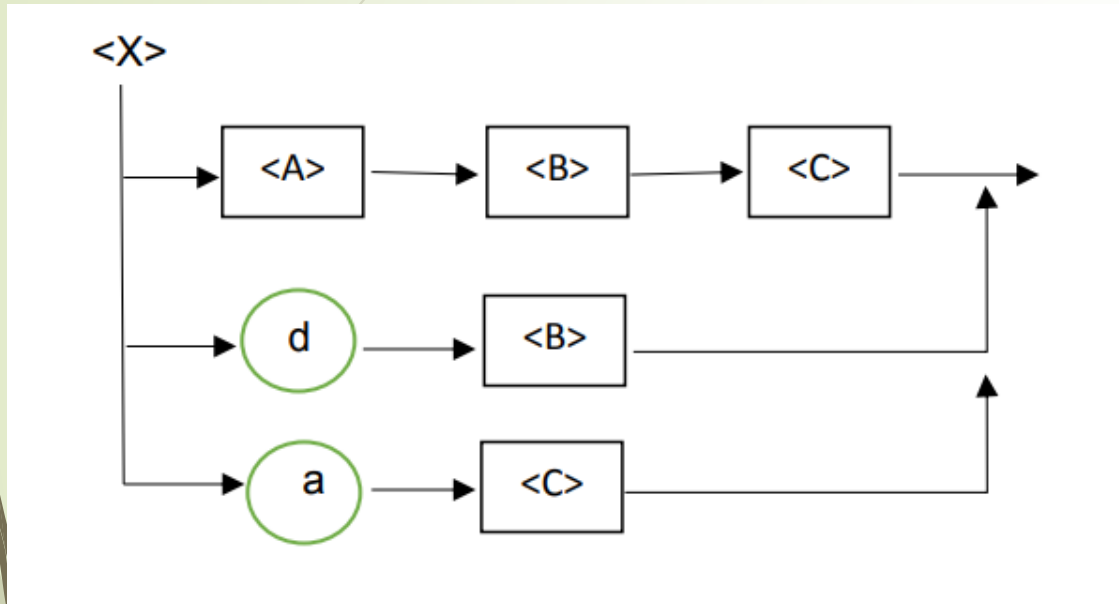
B

C

Finsi

Finsi

Fin



## Diagrama de sintaxis símbolo no terminal A

### RUTINA MODULO SIMBOLO NO TERMINAL A

<A>

Inicio

Si next= "a" entonces

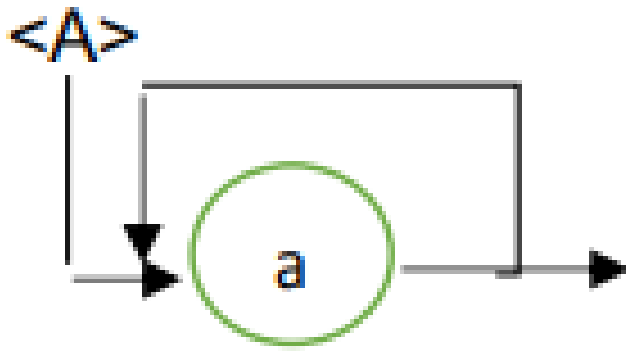
Avanza ()

A

Sino Error ("esperaba a")

finsi

Fin



# Diagrama de sintaxis símbolo no terminal B

## RUTINA MODULO SIMBOLO NO TERMINAL B

<B>

Inicio

Si next = "b" entonces

Avanza()

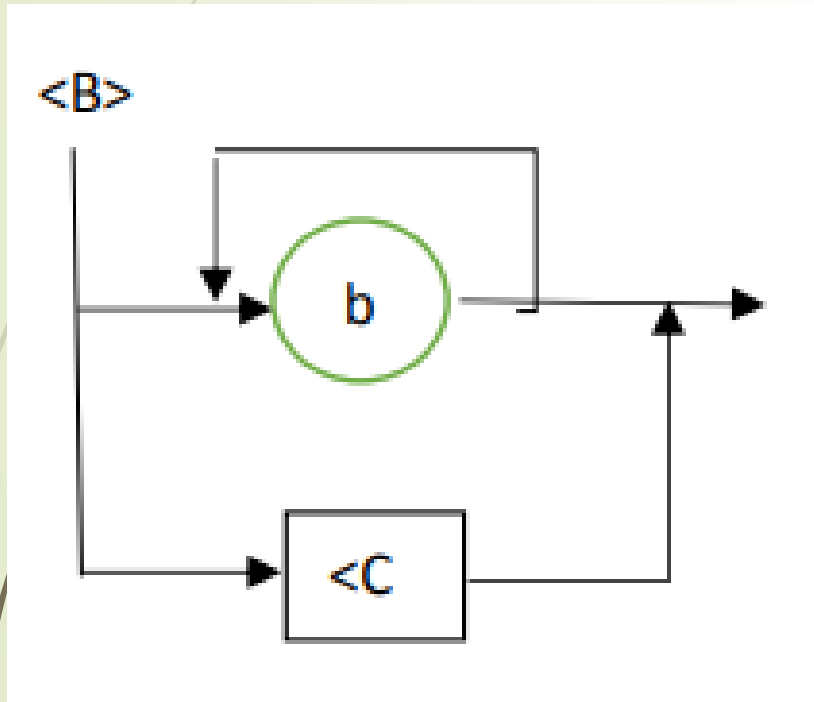
B

Sino

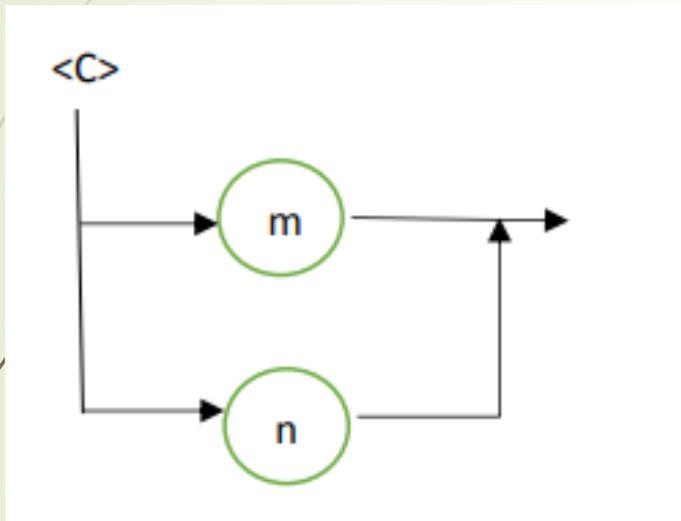
C

finsi

Fin



# Diagrama de sintaxis símbolo no terminal C



## RUTINA MODULO SIMBOLO NO TERMINAL C

<C>

Inicio

Si next="m" entonces

Avanza()

Sino

Si next = "n" entonces

Avanza()

Sino Error("esperaba n")

finsi

finsi

Fin



# Gramáticas



# Definición

- Una gramática libre de contexto tiene cuatro componentes:
- Un conjunto de **no-terminales** ( $V$ ). No terminales son variables sintácticas que denotan conjuntos de cadenas. Los no-terminales definen conjuntos de cadenas que ayudan a definir el lenguaje generado por la gramática.
- Un conjunto de símbolos, conocido como **símbolos terminales** ( $\Sigma$ ) o ( $T$ ). Los terminales son los símbolos básicos de las cadenas que se forman.
- Un conjunto de **producciones** ( $P$ ). Las producciones de una gramática específica la forma en la que los terminales y no terminales se pueden combinar para formar cadenas. Cada producción consiste en un **no-terminal** llamado el lado izquierdo de la producción, una flecha, y una secuencia de los tokens y/o **en los terminales**, llamado el lado derecho de la producción.
- Uno de los terminales es designado como el símbolo de arranque ( $S$ ); desde donde comienza la producción.

# Definición Formal

- Las cadenas se derivan del símbolo de arranque (inicial) varias veces por sustitución de un no-terminal (en un principio, el símbolo de arranque) por el lado derecho de una producción.

$G = \{ V, T, P, S \}$  donde

$V = \{ \text{conjunto de símbolos no terminales} \}$

$T = \{ \text{conjunto de símbolos terminales} \}$

$P = \{ \text{conjunto de producciones} \}$

$S = \text{símbolo sentencial o símbolo de inicio}$

# Ejemplo

Esta gramática describe palíndromes, tales como: 1001, 11100111, 1010101, 00100, 11111, etc

- $V = \{ Q, Z, N \}$
- $T = \{ 0, 1 \}$
- $P = \{ Q \rightarrow Z \mid Q \rightarrow N \mid Q \rightarrow \varepsilon \mid Z \rightarrow 0Q0 \mid N \rightarrow 1Q1 \}$
- $S = \{ Q \}$

- Se puede utilizar el símbolo “ $\mid$ ” para denotar la alternativa “o” y escribir en una sola línea el conjunto de producciones en lugar de varios renglones con el mismo símbolo no terminal
- En el ejemplo anterior, es lo mismo que escribirlo de la siguiente manera
- $Q \rightarrow Z \mid N \mid \varepsilon$
- Para el no terminal  $Q$

## Otra forma

- $V = \{ Q, Z, N \}$

- $T = \{ 0, 1 \}$

- $P = \{$

$$Q \rightarrow Z$$

$$Q \rightarrow N$$

$$Q \rightarrow \varepsilon$$

$$Z \rightarrow 0Q0$$

$$N \rightarrow 1Q1 \}$$

- $S = \{ Q \}$

Forma 1

- $V = \{ Q, Z, N \}$

- $T = \{ 0, 1 \}$

- $P = \{$

$$Q \rightarrow Z \mid N \mid \varepsilon$$

$$Z \rightarrow 0Q0$$

$$N \rightarrow 1Q1 \}$$

- $S = \{ Q \}$

Forma 2



# Derivación

- Una derivación es básicamente una secuencia de reglas de producción, con el fin de obtener la cadena de entrada. Durante el análisis, tomamos dos decisiones:
- Decidir el no terminal que se va a sustituir.
- Decidir sobre la producción, por lo cual, el no-terminal será reemplazado.
- Para decidir que no terminal que se va a reemplazar por una producción, puede tener dos opciones.



# Tipos de derivaciones

## Derivación más a la izquierda

- Si el símbolo no terminal que se va a sustituir se encuentra a la izquierda y se cambia de izquierda a derecha, se llama derivación por la izquierda

## Derivación más a la derecha

- Si el símbolo no terminal que se va a sustituir se encuentra más a la derecha, se llama derivación por la derecha



# Ejemplo

► Normas de producción:

►  $E \rightarrow E + E$

►  $E \rightarrow E * E$

►  $E \rightarrow id$

Cadena de entrada:  $id + id * id$ .

Derivación por la izquierda

$$\begin{aligned} E &\rightarrow E * E \\ &\rightarrow E + E * E \\ &\rightarrow id + E * E \\ &\rightarrow id + id * E \\ &\rightarrow id + id * id \end{aligned}$$

Ing. Karina Cabrera Chagoyan

Derivación por la derecha

$$\begin{aligned} E &\rightarrow E + E \\ &\rightarrow E + E * E \\ &\rightarrow E + E * id \\ &\rightarrow E + id * id \\ &\rightarrow id + id * id \end{aligned}$$



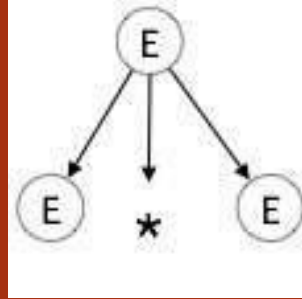
# Parse Tree o árbol gramatical

► Suponga la derivación mas a la izquierda para  $\text{id} + \text{id} * \text{id}$

$E \rightarrow E * E$   
 $\rightarrow E + E * E$   
 $\rightarrow \text{id} + E * E$   
 $\rightarrow \text{id} + \text{id} * E$   
 $\rightarrow \text{id} + \text{id} * \text{id}$

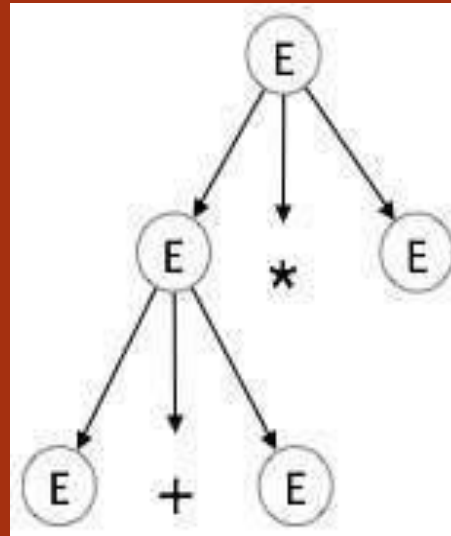
## Paso 1

**$E \rightarrow E * E$**



## Paso 2

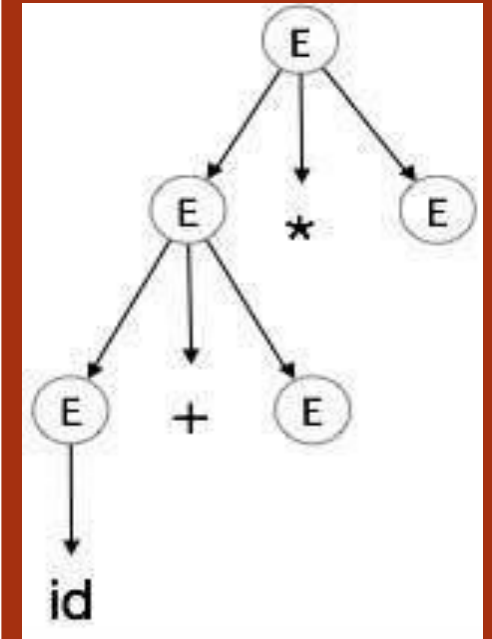
**$E \rightarrow E + E * E$**



Ing. Karina Cabrera Chagoyan

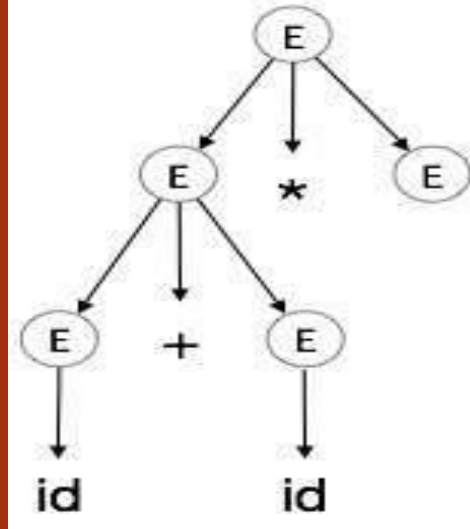
## Paso 3

**$E \rightarrow id + E * E$**



## Paso 4

**$E \rightarrow id + id * E$**



## Paso 5

**$E \rightarrow id + id * id$**

