

Introducción

domingo, 4 de agosto de 2019

06:05 p. m.

► Compiladores

- Temario

- 1] Panorama General
 - 2] Análisis Léxico (Lenguajes Regulares)
 - 3] Análisis Sintáctico
 - 4] Análisis Sintáctico Descendente
 - 5] Análisis Sintáctico Ascendente
 - 6] Traducción dirigido por sintaxis
 - 7] Organización de memoria en tiempo de ejecución
 - 8] Generación de código intermedio y análisis semántico
 - 9] Optimización y generación de código objeto
- } Lenguajes Libres de Contexto

- Bibliografía

- Arquitectura de Computadoras

- AHO, Alfred  This one
Compiladores, principios, técnicas y herramientas
Addison-Wesley Iberoamericano, 2000
- LOUDEN, KENNETH
Compiler Construction. Principles and Practice
Thomson Learning, 1997
- Grune, Dick  PDF
Modern Compiler Design
Springer Second Edition

- Evaluación

- 50% Exámenes
 - * 2 reprobados, reponer
 - * 1º final
 - * 2º final completo
- 20% Programas
- 30% Proyecto Final (Obligatorio)
- 10% Tareas y participaciones

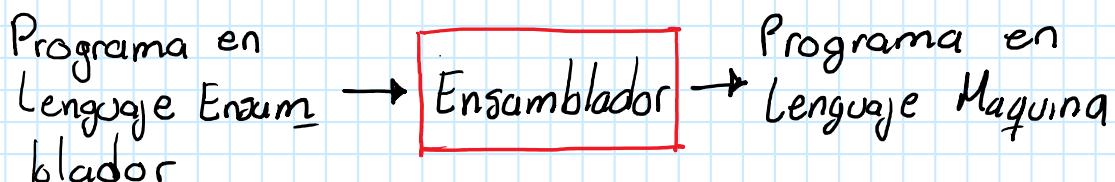
- Correo

aulysesmm@gmail.com

- **Traductor:** Programa que transforma un lenguaje de entrada a otro de salida.

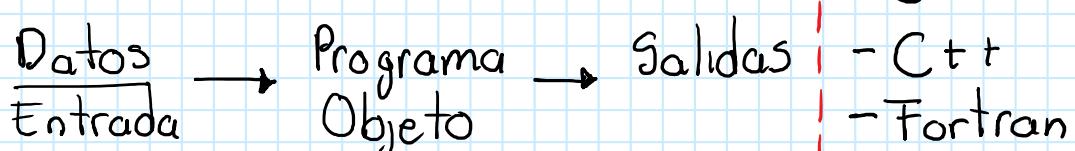
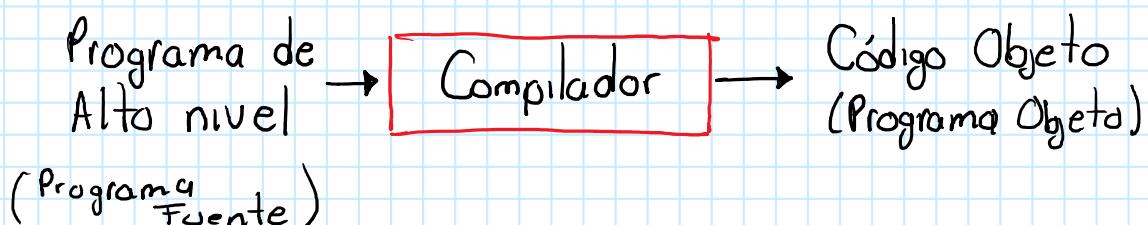
① Ensamblador

NOTA: No es lo mismo ensamblador (Traductor) que el lenguaje ensamblador (Lenguaje)



Lenguajes	① x86
Ensamblador	② z80
	③ HC11

② Compilador



- // Programa Objeto es independiente
- // Detecta errores en tiempo de Com
- // Permite ocultar el código fuente
- // Se puede crear virus
- // No necesitamos ...

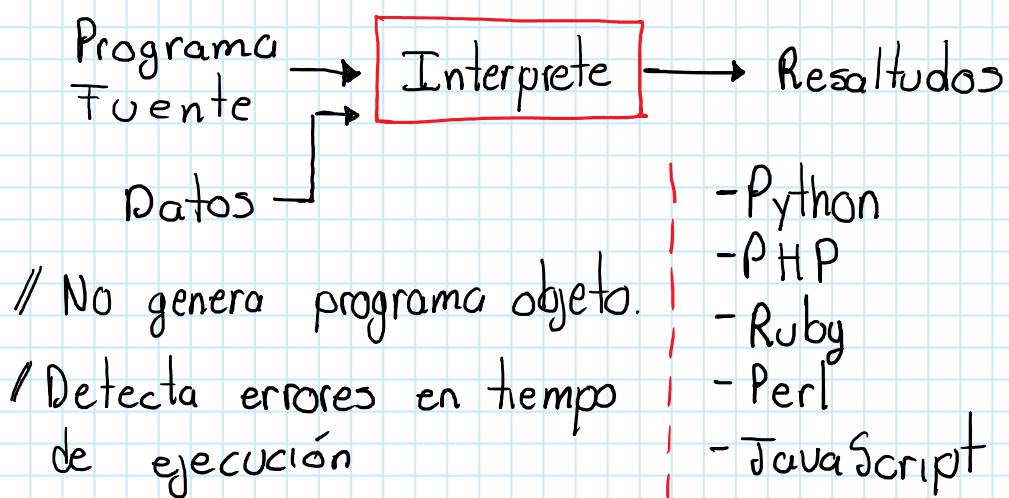
- C
- C++
- Fortran
- Pascal
- ADA
- Delphi*
- VHDL
- LaTeX

// Se puede crear virus

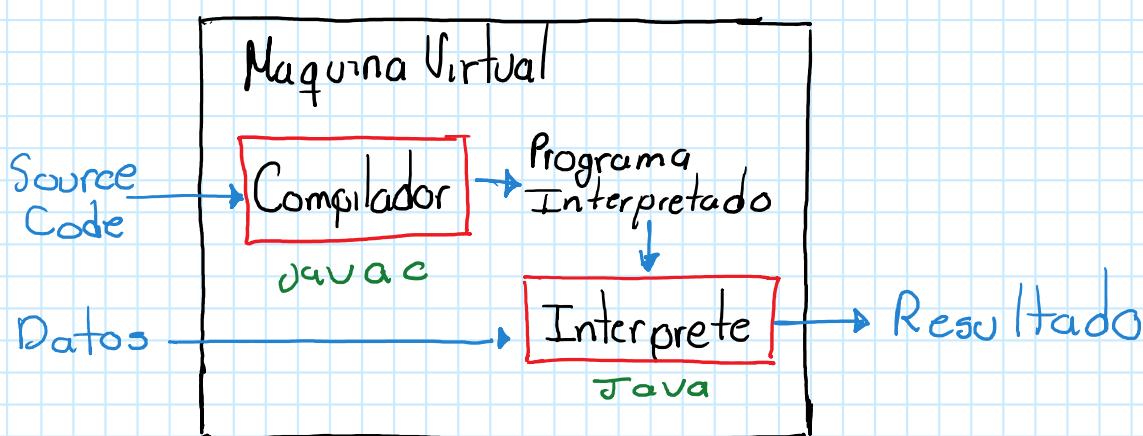
| LaTeX

// No necesariamente arroja ejecutable

③ Interpretes



④ Maquina Virtual



// Es más lento que un compilador tradicional

// Es portable (Necesario interprete)

// Detecta errores en time compilación y ejecución

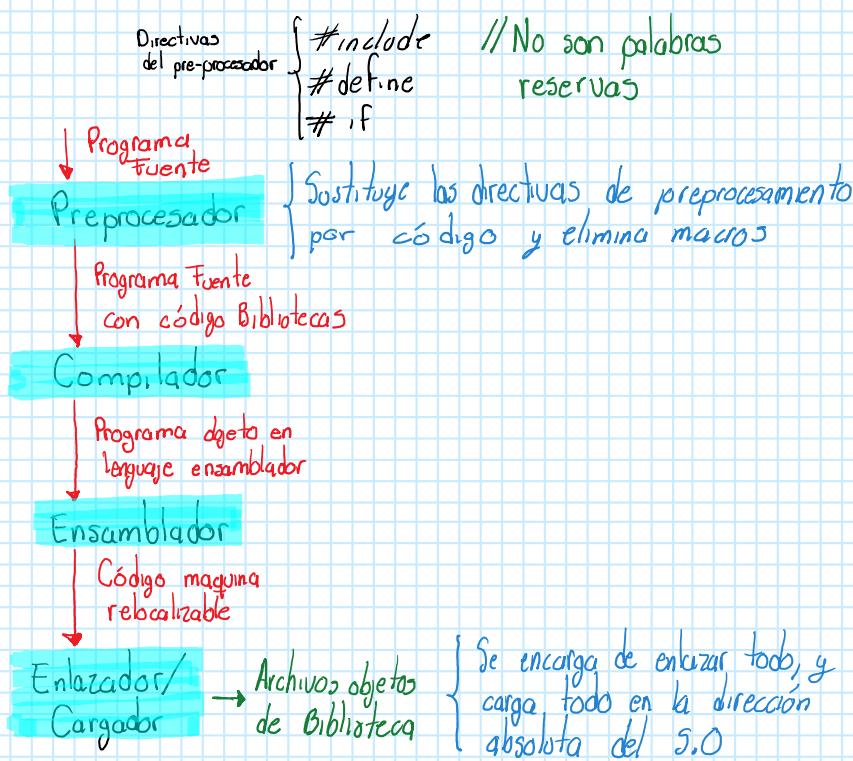
// Podemos esconder código Fuente

// Es más rápido que el interpretado ya que no hay errores de código → Es más ROBUSTO*

Compiladores

- Source to source
 - Pascal → Compilador → C
 - Source to Source
- Compilador Cruzado
 - Una arquitectura (Código Fuente) → Compilador → Programa Resultado (Otra arquitectura)
 - // Microcontroladores
- Meta Compilador
 - Compilador que compila compiladores
 - ↳ Lex / yacc (C)
 - ↳ JFlex / Byacc (Java)

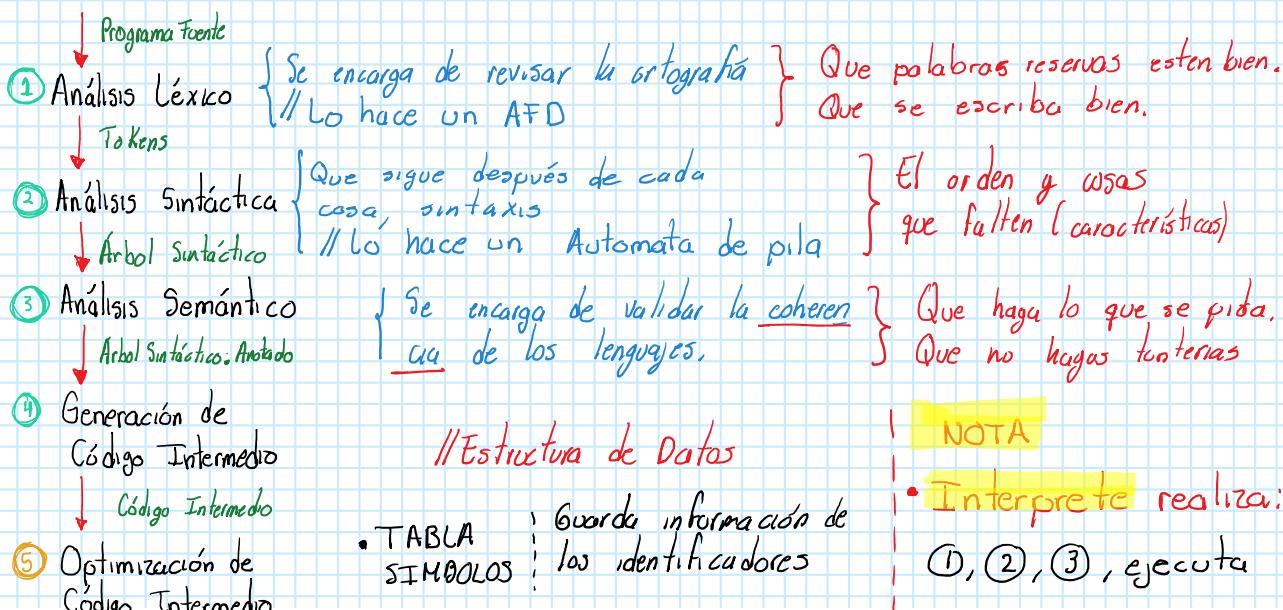
► Sistema de procesamiento del lenguaje



∴ Al llamar lo siguiente
gcc *.c // Se llama al sistema de procesamiento

NOTAS

► Fases de compilación:



- ⑤ Optimización de Código Intermedio
↓
Código Intermedio
- ⑥ Generación de Código Objeto
↓
Código Objeto
- ⑦ Optimización de Código Objeto
↓
Código Objeto

NOTA DE ETAPAS

► Front End o
Etapa de análisis

- TABLA SÍMBOLOS | Guarda información de los identificadores
- TABLA DE TIPOS | Tipos nativos del lenguaje y definidos por el programador
 //Arreglo no es nativo
 //Apuntadores tampoco
- Manejador de errores

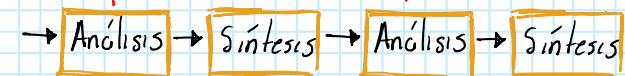
► Back End o
Etapa de síntesis

- ①, ②, ③, ejecuta
- Ensamblador realiza:
①, ②, ③, ⑥, ejecuta

• Código Objeto \neq Código maquina

Com.Fuent \rightarrow Com.Fuent
Asamblly \rightarrow Code.OBJ.

• Compilador de 2 etapas



var x,y : integer // Escrito en Pascal

Ejemplo

```

int a[4];
int i,j;
:
q[i] = a[i-1] + q[i-2];
  
```

① Análisis Léxico

```

<PR, int>
<ID, a>
<[ >
<NUM, 4>
<]>
<;>
  
```

→ Línea 2

```

<PR, int>
<ID, i>
<, >
<ID, j>
<;>
  
```

→ Línea 3

```

<ID, a>
<[ >
<ID, i>
<]>
<= >   <+ >
<ID, a> <ID, a>
<[ >   <[ >
<ID, i> <ID, i>
<~ >   <~ >
<NUM, 1> <NUM, 2>
<]>   <]>
, , .
  
```

TABLA SÍMBOLOS			
Posición	Identificador	Tipo	Dirección
0	a	1	∅
1	i	0	16
2	j	0	20

TABLA DE TIPOS			
ID	Tipo	TipoBase	Tam
0	int	—	4
1	array	int	16 ²

- ① Son 4 bytes del int
- ② Es por 4 espacios ($4 \times 4 = 16$)

$\langle \text{num}, 1 \rangle$ $\langle \text{NUM}, 2 \rangle$
 $\langle \text{J} \rangle$ $\langle \text{?} \rangle$
 $\langle ; \rangle$

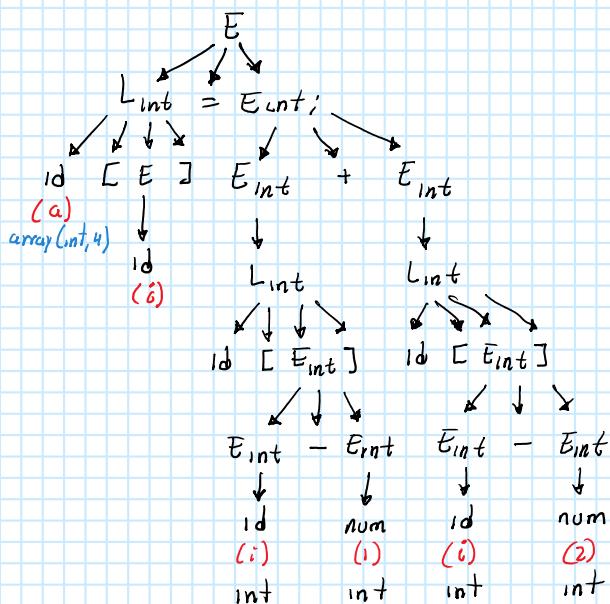
• Gramática

$$E \rightarrow L = E;$$

$$| E + E | E - E$$

$$| \text{ID} | \text{id} | \text{num}$$

$$L \rightarrow \text{id}[E]$$



• Árbol sintáctico

• Código de 3 direcciones | Código Intermedio

$$\begin{aligned}
 t_0 &= i - 1 \\
 t_1 &= t_0 \times 4 \\
 t_2 &= a[t_1] \\
 t_3 &= i - 2 \\
 t_4 &= t_3 \times 4 \\
 t_5 &= a[t_4] \\
 t_6 &= t_2 + t_5 \\
 t_7 &= i \times 4 \\
 \therefore a[t_7] &= t_6
 \end{aligned}$$

• Assemblage Language

```

----- // Resultado
| LD RO, i
| LD RI, #1
| SUB RO, RO, RI
| LD R1, #4
| MUL R0, RO, R1
| LD R2, a(RO) // t_2 = a(t_0)
| LD R3, i
| LD R4, #2
| SUB R5, R3, R4
| MUL R5, R5, R1
| LD R6, a(R5)
| ADD R2, R2, R6
| MUL R3, R3, R1
| ST a(R3), R2

```

► Tarea 2

$\text{float } x, y, z;$
 $\rightarrow x = 30 * y + z;$

► función imaginaria
into float / num

Gramática | $E \rightarrow E + E | E * E | \text{id} = E; | \text{id}$
| num

① Análisis Léxico

< PR, float >
 < ID, x >
 < , >
 < ID, y >
 < , >
 < ID, z >
 < ; >

 < ID, x >
 < = >
 < NUM, 30 >
 < * >
 < ID, y >
 < + >
 < ID, z >
 < ; >

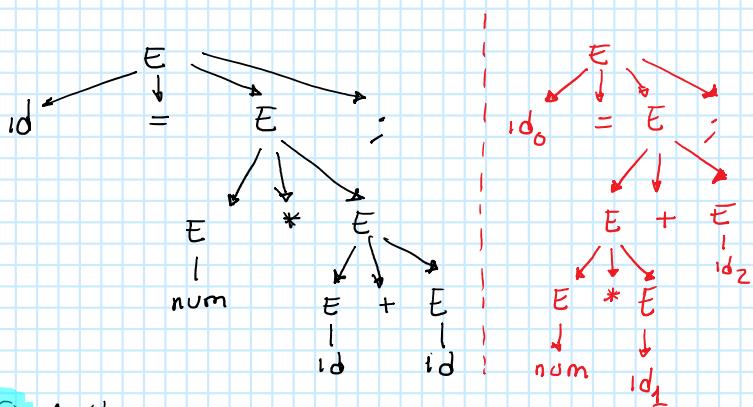
Posición	Identificador	Tipo	Direcciones
0	x	0	0
1	y	0	4
2	z	0	8

TABLA TIPOS

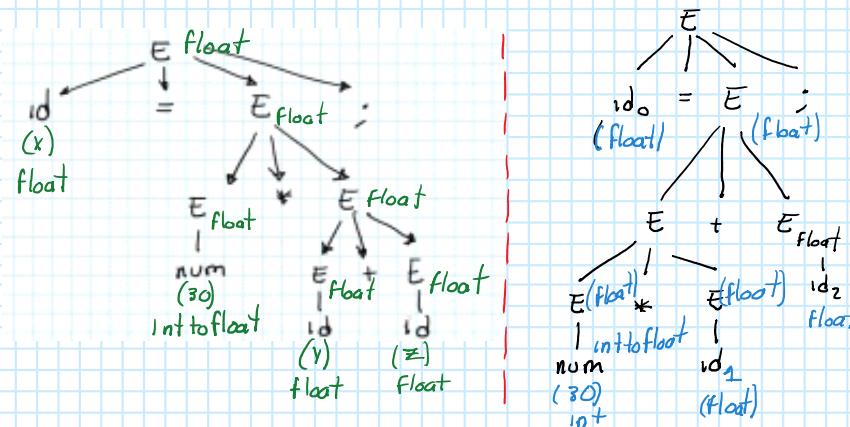
ID	TIPO	TIPO BASE	SIZE
0	float	—	4

NOTA: Poner la posición de la tabla

② Análisis | Árbol Sintáctico



③ Análisis Semántico



④ Generación Código Intermedio

	OP	RES	ARG1	ARG2
$t_0 = \text{int to float}(30)$	LD	R ₀	30	
	LD	R ₁		ID ₁

$$\begin{aligned}
 t_0 &= \text{int}(\text{to float}(30)) \\
 t_1 &= t_0 * \cancel{Id_1} \\
 t_2 &= t_1 + \cancel{Id_2} \\
 y &= t_2 \\
 \cancel{Id_0} &\cancel{\equiv} \cancel{t_2}
 \end{aligned}$$

'	LD	R_0	30	
'	LD	R_1	\cancel{y}	$\rightarrow Id_1$
'	MULT	R_0	R_0	R_1
'	LD	R_2	\cancel{z}	$\rightarrow Id_2$
'	ADD	R_0	R_0	R_2
'	ST	\cancel{X}	R_0	
'		Id_0		

// Optimización

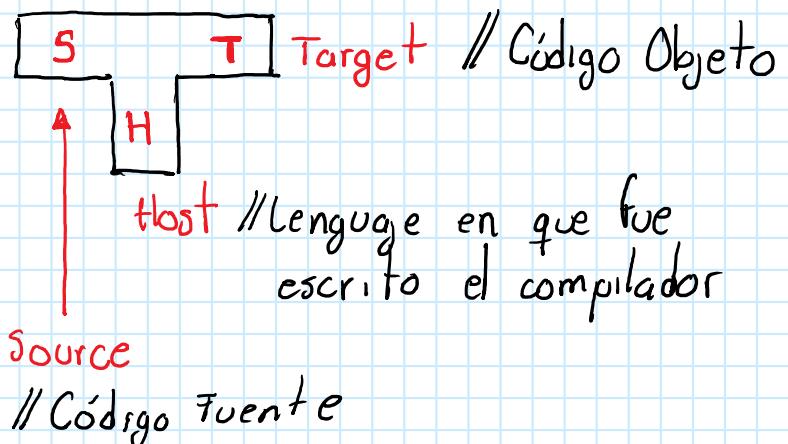
$$t_0 = 30.0 * Id_1$$

$$Id_0 = t_0 + Id_2$$

OP	RES	ARG1	ARG2
LDF	R_0	#30.0	
LDF	R_1	Id_1	
MULT	R_0	R_0	R_1
LDF	R_1	Id_2	
ADD	R_0	R_0	R_1
STF	Id_0	R_0	

→ Diagrama T

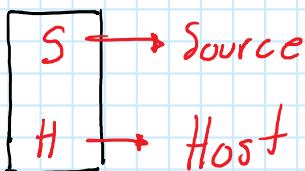
- Define al compilador mediante 3 lenguajes:



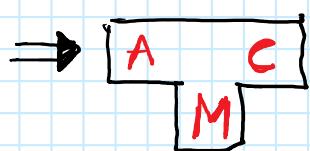
• La maquina virtual



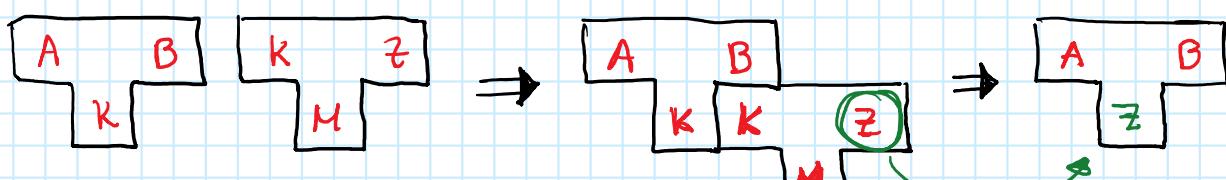
- Define el interprete



// Ejercicio 1



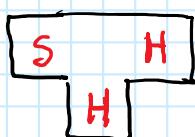
// Ejercicio 2



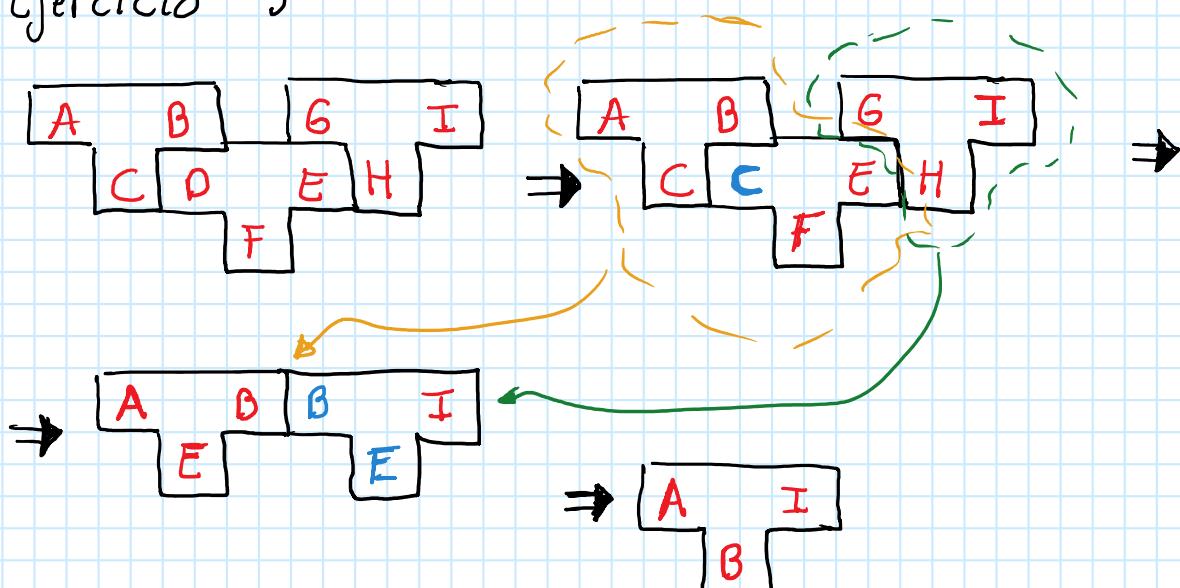


- // Ponemos el host con el entrada (Letra k)
- // Posterior, salida del 2º compilador se vuelve host del final
- // Compilador cruzado → Diferentes arquitecturas

NOTA:
Compilador
común



// Ejercicio 3

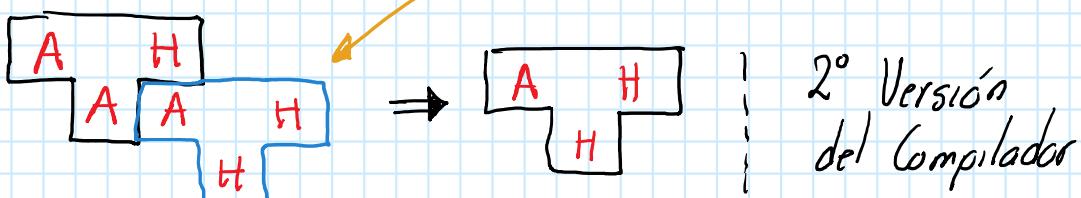


► Puesta en marcha de un Compilador

- Unix → Escrito en lenguaje ensamblador
- Lenguaje → Se desarrolló en ensamblador C

∴ Unix se reescribió en lenguaje C

- ① Escribir un compilador ineficiente y lento en cualquier otro lenguaje. Ejemplo: Ensamblador
- ② Con el compilador lento e ineficiente, compilar uno nuevo que agregue funcionalidad.
- ③ Hacer el paso 2 hasta optimizar.



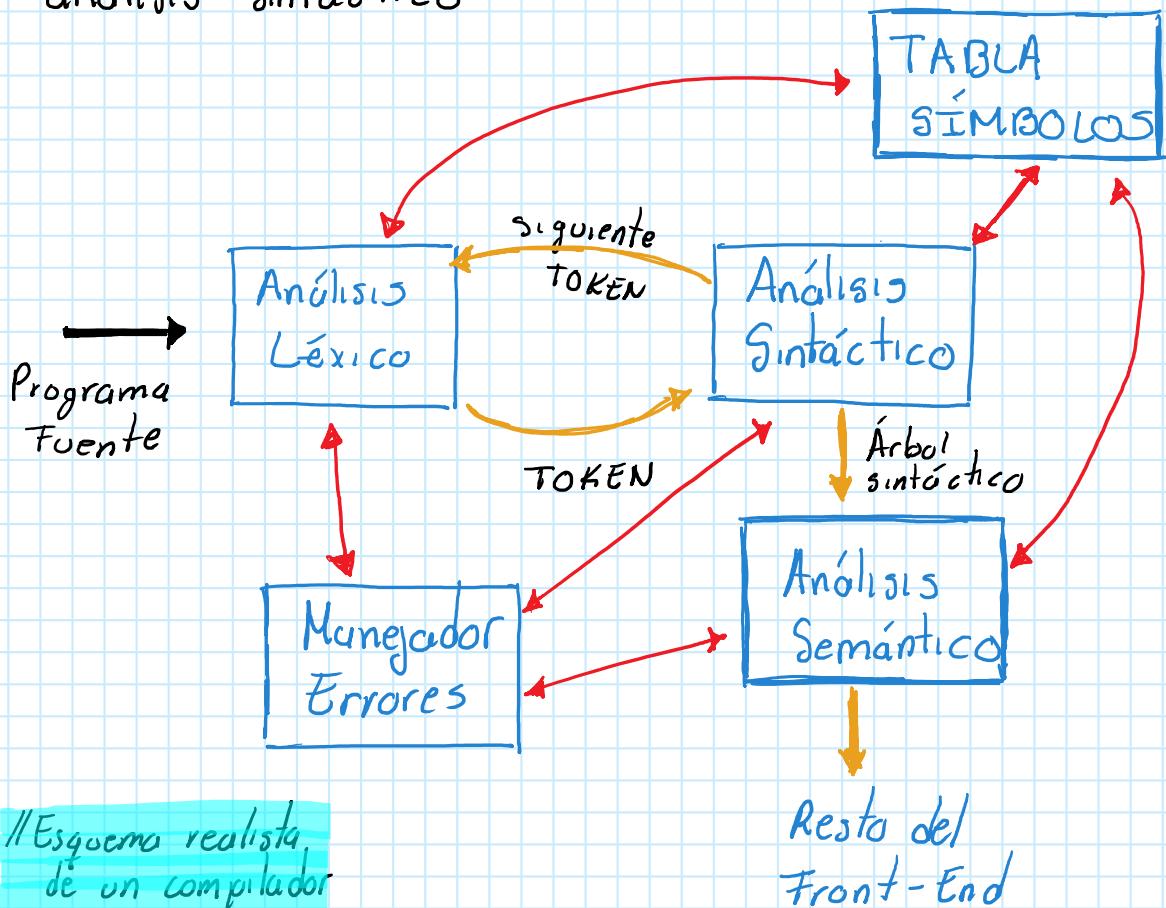
NOTA:
histórica

`C++ C` // Antes se escribía en C++
 C se trabajaba en C y
 se daba el C.Obj.

2.1 Análisis Léxico

Monday, August 12, 2019 2:32 PM

- El que entra en ejecución es el análisis sintáctico



► Tareas Análisis Léxico

- 1.- Leer el programa fuente
- 2.- Devolver tokens
- 3.- Detectar errores léxicos
- 4.- Recuperarse de los errores léxicos
- 5.- Ignorar los espacios en blanco // Depende del lenguaje.
- 6.- Ignorar comentarios // Sólo si no tiene preprocesador

NOTA • Un **error léxico** es cualquier símbolo extraño no definido en el lenguaje.

no se define en el lenguaje.

// Errores Léxicos

1] Palabras reservadas mal escritas // No existe en C

2] Identificadores mal formados // 1 variable;

3] Identificadores que excedan longitud

NOTA: La mayor longitud de variables en C = **31**

4] Constantes numéricas mal escritas // Directamente los núm.

// 1.45.45

↑ 4 Tiene 2 puntos

5] Constantes numéricas fuera de rango

// Pasarse del límite de un short o char

6] Constantes de cadenas mal formados

→ " Una tiene 2 comillas, y otra una '

→ 'holca' // No es un carácter

g] Comentarios que no finalizan

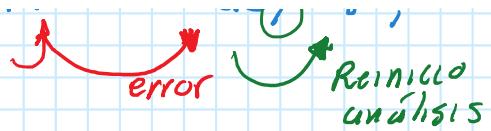
// Porque todo el programa es un comentario enorme.

// Recuperación de errores Léxicos

① Modo pánico // El más común.

Una vez encontrado un error se omite hasta encontrar un token nuevo:





- ② Insertar un carácter faltante en la entrada;
 - ③ Eliminar un carácter de la entrada
 - ④ Trasponer 2 caracteres de la entrada
- } No tan usadas

- Analizador Léxico

Wednesday, August 14, 2019 1:33 PM

► Proceso para hacer Analizador Léxico
o Scanner o Lexer

- 1) Definir los tokens del lenguaje:
- 2) Generar las expresiones regulares para todos los tokens
// Pueden existir varias exp. regulares para definir un token (Lenguaje Regular)
- 3) Obtener el Automata Determinista (AFD) que reconozca todas las expresiones regulares,
- 4) Minimizar el automata (AFD)

// Ejemplo:

$(a|b)^* abb$; - Cadenas que terminan con abb

- 5) Programar el automata

NOTA:

→ Token | • Es la unidad mínima léxica que tiene significado para el lenguaje de programación.

| • Se compone de una clase léxica, o de un valor opcional

< CLASE LÉXICA, valor >

```
struct token {  
    short clase;  
    char *valor;
```

} Como se podría
hacer en "C"

```

    |     donri  use;
    |     char  *valor;
    |
    | }                               } nacer en

```

- Clases de Token
- 1) Palabras Reservadas
 - 2) Identificadores
 - 3) Constantes numéricas
 - i) Enteros
 - ii) Imaginarios
 - iii) Reales
 - 4) Operadores
 - i) Aritméticos
 - ii) Lógicos
 - iii) Relacionales
 - iv) Bit
 - v) Asignación
 - 5) Símbolos de puntuación
 $,$ $:$ $:$ $,$ $,$ ${}$ $$
 - 6) Constantes cadena
 - 7) Comentarios
 $//$ $/* */$
 - 8) Espacios en blanco

► Expresión Regular

- Alfabeto (Σ) • Es un conjunto finito de símbolos
- Lenguaje • Conjunto de cadenas producido por un alfabeto el cual tiene reglas definidas.
 ↳ "Reglas Definidas" → Gramática
- Expresión Regular • Definen un patrón de búsqueda, para buscar las cadenas que pertenecen a un lenguaje

Regular ; buscar las cadenas que pertenecen a un lenguaje.

1) $E \stackrel{\Delta}{=} \text{Expresión regular que define}$

$$L(E) = \{E\} \quad // \text{Cadena vacía}$$

2) Si $a \in \Sigma$ entonces

a es una expresión regular y $L(a) = \{a\}$

► Operadores

• Sean " r " y " s " 2 expresiones regulares

1) $(r) | (s)$ // Disyunción $\rightarrow L\{(r) | (s)\} = \{r, s\}$
 \hookrightarrow uno u otro

2) $(r)(s)$ // Concatenación $\rightarrow L\{(r)(s)\} = \{rs\}$
 \hookrightarrow Unir las expresiones regulares

3) $(r)^*$ // Cerradura Kleene $\rightarrow L\{(r)^*\} = \{\epsilon, r, rr, \dots r^n\}$

4) $\underline{(r)}^+$ // Cerradura Positiva $\rightarrow L(r^+) = \{r, rr, \dots r^n\}$

5) $(r)? = (r) | \epsilon$ // Puede o no estar (O sea opcional)

6) $[...]$ $\neq [a-z]$ $\neq [0-9]$ $\neq [()\{\}]$

7) $\sim(r)$ Negación // Todo lo que no sea r
 \hookrightarrow // No está restringido por el alfabeto

$\sim(abb) \rightarrow aba, aab, aaa$

\hookrightarrow // No son lo mismo que "abb"

8) r^* // Mención

8) $[^{\dots}]$ // Negación

↪ $[^{\text{ab}}]$ // Que no se ab

NOTA: \neg y \bar{x} son lo mismo

9) (r) // Expresión regular que reconoce el lenguaje de "r" → $L((r)) = \{ r \}$

Amenazas | ① Corchetes no se usan para agrupar

$[(a|b)^*abb(a|bb)^*]$

| ② No usar símbolo disyunción dentro de corchetes

$[a|b|c|d]$

| ③ Prohibido uso de comas

► Como trabajar expresiones regulares

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

:

$d_n \rightarrow r_n$

• Cada d_i es diferente a todas las anteriores // Ninguna se llama igual

• Cada r_i puede estar formada de símbolos de $\Sigma \cup \{d_1, \dots, d_{i-1}\}$

• Nombre de expresiones regulares anteriores.

Letra → letra | NO EXISTE
Recursividad | RECURSIVIDAD

// Obtener la expresión regular para los identificadores

// Obtener la expresión regular para los identificadores válidos del lenguaje C.

$$\text{letra} \rightarrow [a - z A - Z]$$

$$\text{letra_} \rightarrow _{-} | \text{letra}$$

$$\text{digito} \rightarrow [0 - 9]$$

$$\bullet \text{id} \rightarrow \text{letra_} (\text{letra_} | \text{digito})^*$$

// Expresión regular para el lenguaje sobre el alfabeto a-b donde se tiene número par de "a" y número par de "b"

$$\text{first} \rightarrow aa | bb$$

$$\text{second} \rightarrow ab | ba$$

$$\text{general} \rightarrow ((\text{first})^* | ((\text{second}))^*)^*$$

→ Caso Correcto | $(aa|bb|(ab|ba)(aa|bb))^*(ab|ba)^*$

// Encontrar la expresión regular para los números flotantes en lenguaje C que incluye los de notación exponencial

$$\bullet \text{number} \rightarrow [0 - 9]$$

$$\text{signos} \rightarrow (+|-)$$

~~$$\text{final} \rightarrow (\text{number})^+ (.) ((\text{number})^+ (f) | (e) (\text{signos}) (\text{number})^+ (f))$$~~

• Caso | • digito $\rightarrow [0 - 9]$

debería estar

- Caso Correcto
 - dígito $\rightarrow [0-9]$
 - entera $\rightarrow (\text{dígito})^+$
 - exponente $\rightarrow [Ee][+ -]?\text{entero}$ Podría estar
 - flotante $\rightarrow (\text{entero}.\text{entero}?)\text{entero?exponente}$
 - | entero?.entero exponente?
 - | entero.entero?exponente?) [ff]

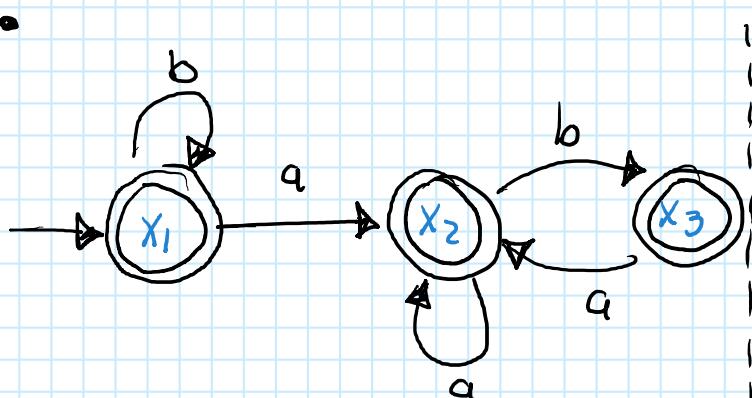
- // Encontrar la exp. regular para la palabra reservada "select" en cualquier combinación de minúsculas y mayúsculas.
- final $\rightarrow [Ss][Ee][Ll][Ee][Cc][Tt]$

- // La expresión regular que da todos los números enteros que no empiecen con 0 menos el cero

$$\text{final} \rightarrow [1-9][0-9]^* | 0$$

NOTA: $a^+? = a^*$

- // Todas las cadenas de "a" y "b" y que no contienen a la subcadena abb



NOTA 2: Negar automa
(Invierte finales)

- // Automata que genera expresión regular

$$\begin{array}{l} x_1 \rightarrow bx_1 + ax_2 + \epsilon \quad (1) \\ x_2 \rightarrow ax_2 + bx_3 + \epsilon \quad (2) \\ x_3 \rightarrow ax_2 + \epsilon \quad (3) \end{array}$$

Conviirtiendo
Automata a
Expresión Regular

// Sustituyendo (3) en (2)

$$\begin{aligned} x_2 &\rightarrow ax_2 + b(ax_2 + \epsilon) + \epsilon \\ &\rightarrow ax_2 + bax_2 + b + \epsilon \\ &\rightarrow (a + ba)x_2 + b + \epsilon \\ x_2 &\rightarrow (a + ba)^*(b + \epsilon) \quad (4) \end{aligned}$$

// LEY ARDEN

$$\begin{aligned} x &= rx + s \\ x &= r^* s \end{aligned}$$

// Sustituyendo (4) en (1)

$$\begin{aligned} x_1 &\rightarrow bx_1 + a((a + ba)^*(b + \epsilon)) + \epsilon \\ &\rightarrow b^*(a(a + ba)^*(b + \epsilon) + \epsilon) \\ \therefore \text{expReg} &\rightarrow b^*(a(a + ba)^*(b)? \mid \epsilon) \end{aligned}$$

AFND

Monday, August 19, 2019 1:14 PM

► Autómata Finito No Determinístico

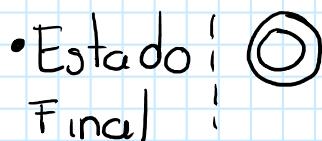
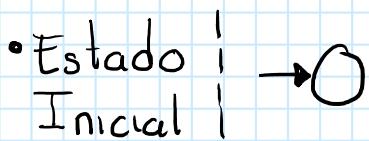
- $N(Q, \Sigma, S, \Delta, F)$ | $\subseteq //$ Subconjunto
- $S \subseteq Q$ - $\Delta: 2^Q \times \Sigma \rightarrow 2^Q$
- $F \subseteq Q$

► Autómata Finito Determinista

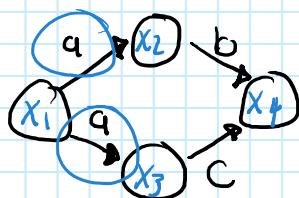
- $s \triangleq$ Estado Inicial // Sólo uno | • $D(Q, \Sigma, s, \delta, F)$
- $F \triangleq$ Estados Finales | - $S \subseteq Q$ - $\delta: Q \times \Sigma \rightarrow Q$
- | - $F \subseteq Q$

// Representaciones

→ Diagramas de estados

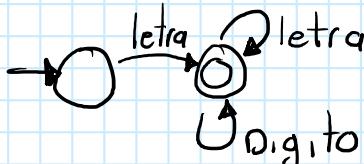


• No Determinista |



Debido a que
se va hacia
2 con la misma

• Determinista |



letra $\rightarrow [a-z]$

dígito $\rightarrow [0-9]$

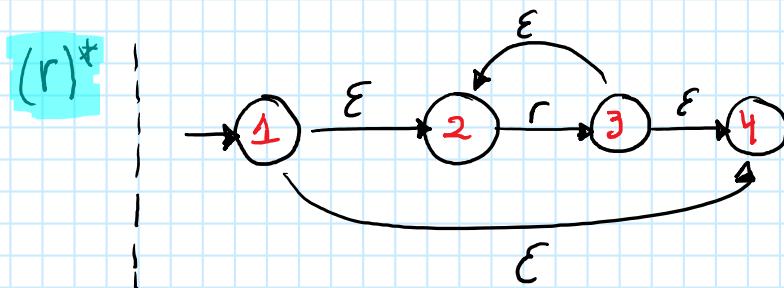
→ Tabla de estados

Letra	Dígito
→ n	+

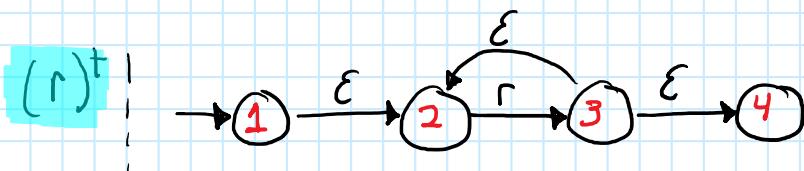
→ // Estado inicial

	Letra	Dígito	
0	0	1	→ // Estado inicial
1	2	1	* // Estado aceptación
2	2	3	
3*	2	4	

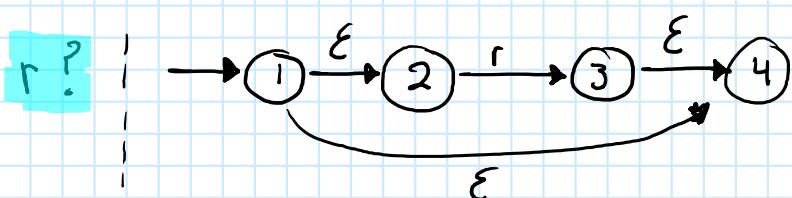
→ Pasar de Exp.Reg → AFD



- $(r)^* \Rightarrow (\cdot r)^* \quad (r \cdot)^* \Rightarrow (\cdot r)^*$
- $(r)^* \cdot \quad (r \cdot)^* \cdot$



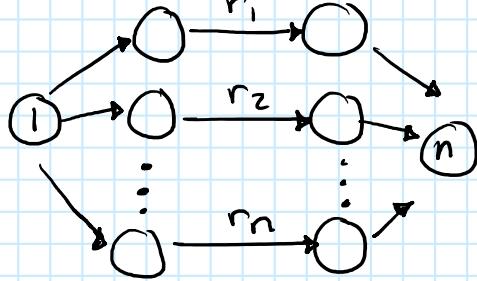
- $(r)^+ \Rightarrow (\cdot r)^+$
- $(r \cdot)^+ \Rightarrow (\cdot r)^+$



- $(r)? \Rightarrow (\cdot r)? \quad (r \cdot)? \Rightarrow r? \cdot$
- $r? \cdot$



$(r_1 | r_2 | \dots | r_n)$



- $(r_1 | r_2 | r_n) \Rightarrow (\cdot r_1 | r_2 | \dots | r_n)$
 $(r_1 | \cdot r_2 | \dots | r_n)$
 \vdots
 $(r_1 | r_2 | \dots | \cdot r_n)$

$$(\dots | r_i | \dots) \Rightarrow (\dots | r_i | \dots) \bullet$$

→ Automata que reconoce la expresión regular $(a|b)^*abb$

$$\bullet (a|b)^*abb$$

$$\left. \begin{array}{l} \rightarrow \underline{(a|b)^*abb} \\ \rightarrow (a|\cdot b)^*abb \\ \rightarrow (a|b)^*\cdot abb \end{array} \right\} q_0 // \text{Todas las posibles opciones}$$

$$\rightarrow \text{goto}(q_0, a) = \left\{ \begin{array}{l} \cancel{(a \cdot b)^*abb} \text{ se cancela} \\ (a|b)^*a \cdot bb \\ \rightarrow (\cdot a|b)^*abb \xrightarrow{*} \\ (a|\cdot b)^*abb \\ \rightarrow (a|b)^*\cdot abb \end{array} \right\} = q_1$$

// Tarea Mural

$$\rightarrow \text{goto } (q_0, b) = \left\{ \begin{array}{l} \cancel{(a|b)^* abb} // \text{Se repite} \\ (a|b)^* abb \\ (\cdot a|b)^* abb \\ (a|b)^* \cdot abb \end{array} \right\} = q_0$$

Se repite

$$\rightarrow \text{goto } (q_1, a) \left\{ \begin{array}{l} (\cdot a|b)^* abb \\ (a|b)^* abb \leftarrow \\ (a|b)^* \cdot abb \\ (a|b)^* a \cdot abb \end{array} \right\} = q_1$$

$$\rightarrow \text{goto } (q_1, b) \left\{ \begin{array}{l} (a|b)^* abb \\ (\cdot a|b)^* abb \leftarrow \\ (a|b)^* \cdot abb \leftarrow \\ (a|b)^* ab \cdot b \end{array} \right\} = q_2$$

$$\rightarrow \text{goto } (q_2, a) = \left\{ \begin{array}{l} (\cdot a|b)^* abb \\ (a|b)^* abb \\ (a|b)^* \cdot abb \\ (a|b)^* a \cdot abb \end{array} \right\} = q_1$$

$$\rightarrow \text{goto } (q_2, b) = \left\{ \begin{array}{l} (\cdot a|b)^* abb \leftarrow \\ (a|b)^* abb \leftarrow \\ (\cdot a|b)^* \cdot abb \leftarrow \end{array} \right\}$$

$$\begin{array}{l}
 (a|b)^* \cdot abb \leftarrow \\
 - (a|b)^* abb \cdot f = q_3 \quad \text{No se pasa}
 \end{array}$$

$$\rightarrow \text{goto}(q_3, a) = \left\{ \begin{array}{l} (\cdot a|b)^* abb \\ (a|b)^* abb \\ (a|b)^* \cdot abb \\ (a|b)^* a \cdot bb \end{array} \right\} = q_1$$

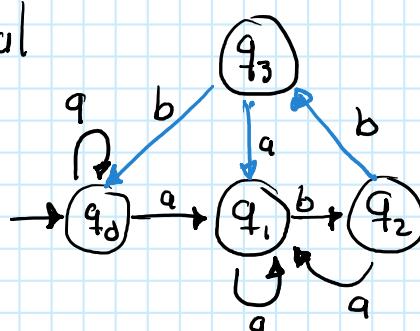
$$\rightarrow \text{goto}(q_3, b) = \left\{ \begin{array}{l} (\cdot a|b)^* abb \\ (a|b)^* abb \\ (a|b)^* \cdot abb \end{array} \right\} = q_0$$

~~$(a|b)^* abb$~~

// Tabla de transiciones

Estado \rightarrow Aquellos estados donde al menos hay un $\rightarrow \circlearrowright$
 Aceptación \rightarrow elemento final

	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_0



AFND

Wednesday, August 21, 2019 1:48 PM

$$\rightarrow (a|b)^* \left((c|d)(a|b)^*(c|d)(a|b)^* \right)^*$$

$$q_0 = \left\{ (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \right. \\ \left. (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \right\}$$

// Entramos a la cerradura enorme

$$(a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \leftarrow$$

$$(a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^*$$

// Omitiendo el estrella

$$(a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \cdot \}$$

$$\rightarrow \text{goto}(q_0, a) = \left\{ \begin{array}{l} (a|b)^* (\dots)^* \quad // \text{Comentario} \\ (a|b)^* (\dots)^* \\ (\dots)^* ((c|d)(\dots))^* \\ (\dots)^* ((c|d)(\dots))^* \\ \dots \end{array} \right\} = q_0$$

$$\rightarrow \text{goto}(q_0, b) = q_0 \quad // \text{Comentario}$$

Comentario: Al ser dentro de la misma disyunción llegamos al mismo estado

$$\rightarrow \text{goto}(q_0, c) = \left\{ \begin{array}{l} (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \\ (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \\ (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \\ (a|b)^* ((c|d)(a|b)^*(c|d)(a|b)^*)^* \end{array} \right\} =$$

$$(a|b)^*(c|d)(a|b)^*(c|d)(a|b)^*)^* \} = \\ = q_1$$

$\rightarrow \text{goto}(q_0, d) = q_1$ // Es el mismo por la disyunción

$\rightarrow \text{goto}(q_1, a) = q_1$ // Es el mismo por la disyunción

$\rightarrow \text{goto}(q_1, b) = q_1$ // Es el mismo por la disyunción

$\rightarrow \text{goto}(q_1, c) = \{ (a|b)^*((c|d)(a|b)^*(c|d)(\neg a|b)^*)^* \\ (a|b)^*((c|d)(a|b)^*(c|d)(a|\neg b)^*)^* \\ (a|b)^*((\neg c|d)(a|b)^*(c|d)(a|b)^*)^* \\ (a|b)^*((c|\neg d)(a|b)^*(c|d)(a|b)^*)^* \} = q_2$

// Incluir Final

$\rightarrow \text{goto}(q_1, d) = q_2$

$\rightarrow \text{goto}(q_2, a) = q_2$

$\rightarrow \text{goto}(q_2, b) = q_2$

$\rightarrow \text{goto}(q_2, c) = q_1$

$\rightarrow \text{goto}(q_2, d) = q_1$

// Tabla de transiciones

	a	b	c	d
q_0^*	q_0	q_0	q_2	q_1
q_1	q_1	q_1	q_2	q_2
q_2^*	q_2	q_2	q_1	q_1

$$\rightarrow (aa|bb)^*((ab|ba)(aa|bb)^*(ab|ba)(aa|bb)^*)^*$$

$$q_0 = \{ (aa|bb)^*(...)^*$$

$$\xrightarrow{\quad} (aa|bb)^*(...)^*$$

// Entramos al segundo

$$\xrightarrow{\quad} (...)^*((ab|ba)...)^*$$

$$\xrightarrow{\quad} (...)^*((ab|ba)...)^*$$

$$\xrightarrow{\quad} (...)^*(...)^* \} \xrightarrow{\quad} \text{Final}$$

$$\underline{\text{goto}}(q_0, a) = \{ (a \cdot a | bb)^*(...)^*$$

$$\xrightarrow{\quad} (...)^*((a \cdot b | bb)(...)^* ...)^* \} = q_1$$

$$\text{goto}(q_0, b) = \{ (aa | b \cdot b)^*(...)^*$$

$$\xrightarrow{\quad} (...)^*((ab | b \cdot a)(...)^* ...)^* \} = q_2$$

$$\text{goto}(q_1, a) = \{ (aa | bb)^*(...)^*$$

$$(aa | bb)^*(...)^*$$

:

$$\} = q_0$$

$$\underline{\text{goto}}(q_1, b) = \{ (...)^*((aa | bb)^* ...)^*$$

$$(...)^*((aa | bb)^* ...)^*$$

$$(...)^*((...) (...)^*(ab | ba) ...)^*$$

$$(...)^*((...) (...)^*(ab | ba) ...)^* \} =$$

$$= q_3$$

$$\text{goto}(q_1, b) = \{ (...)^*((aa | bb)^* ...)^*$$

$$\text{goto}(q_2, a) = \{ (\dots)^* ((\dots)(\cdot aa | bb)^* \dots)^* \\ (\dots)^* ((\dots)(aa | \cdot bb)^* \dots)^* \\ \vdots \\ \} = q_3$$

$$\text{goto}(q_2, b) = \{ (aa|bb)^* (\dots)^* \}$$

$$\text{goto}(q_3, a) = \{ ((\dots)^* ((\dots)(a \cdot q_1 | bb)^* \dots)^* \\ (\dots)^* ((\dots)(\dots)^* (a \cdot b | ba)^* \dots)^* \}$$

= q_4

$$\text{goto}(q_3, b) = \{ (\dots)^* ((\dots)(aa|bb)^* \dots)^* \\ (\dots)^* ((\dots)^* (\dots)^* (ab|ba)^* \dots)^* \}$$

$= q_5$

$$\text{goto}(q_4, a) = q_3$$

$$\text{goto } (q_4, b) = \left\{ \begin{array}{l} (\dots)^* ((\dots)(\dots)^* (\dots)(aa|bb)^*)^* \\ (\dots)^* ((\dots)(\dots)^* (\dots)(aa|bb)^*)^* \\ (\dots)^* ((ab|ba)(\dots) \dots)^* - \\ (\dots)^* ((abl|ba) \dots)^* \\ (\dots)^* (\dots)^* \bullet \text{Final} \end{array} \right\} = q_6$$

$$goto(q_5, a) = \{ (\dots)^* ((\dots) (\dots)^* (\dots) (\dots) (aa/bb)^* \}^*$$

$$\text{goto}(q_5, a) = \left\{ \begin{array}{l} (\dots)^* ((\dots)(\dots)^* (\dots)(aa/bb)^*)^* \\ (\dots)^* ((\dots)(\dots)^* (\dots)(a\bar{a}/\bar{b}b)^*)^* \\ \vdots \\ \vdots \end{array} \right\} = q_6$$

$$\text{goto}(q_5, b) = \{\} = q_3$$

$$\text{goto}(q_6, a) = \left\{ \begin{array}{l} (\dots)^* ((\dots)^* ((\dots)(aa/bb)^*)^* \\ (\dots)^* ((a\bar{b}/\bar{b}a)(\dots)^* (\dots)^*)^* \end{array} \right\} = q_7$$

$$\text{goto}(q_6, b) = \left\{ (\dots)^* ((\dots)(aa/b\cdot b)^*)^* \right\} = q_8$$

$$\text{goto}(q_7, a) = \{\dots\} = q_6$$

$$\text{goto}(q_7, b) = \{\dots\} = q_3$$

$$\text{goto}(q_8, a) = \{\dots\} = q_3$$

$$\text{goto}(q_8, b) = \{\dots\} = q_6$$

// Tabla de transiciones

Valdez Vergas

	a	b
$\rightarrow q_0^*$	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_4	q_5
q_4	q_3	q_6
q_5	q_6	q_3
q_6^*	q_7	q_8
q_7	q_6	q_3
q_8	q_3	q_6

AFND

Monday, August 26, 2019 1:12 PM

1. Obtener el automata de los números enteros y los n.º decimales, donde la parte entera es opcional:

$$\text{dígito} \rightarrow [0-9]$$

$$\text{entero} \rightarrow (\text{dígito})^+$$

$$\text{decimal} \rightarrow (\text{dígito})^*.(\text{dígito})^+$$

// Automata

$$\text{entero} \rightarrow \bullet (\text{dígito})^+$$

$$\text{decimal} \rightarrow \bullet (\text{dígito})^*.(\text{dígito})^+$$

$$q_0 = \{ \begin{array}{l} \text{entero} \rightarrow (\bullet \text{dígito})^+ \\ \text{decimal} \rightarrow (\bullet \text{dígito})^* \bullet (\dots)^+ \\ \text{decimal} \rightarrow (\dots)^* \bullet (\text{dígito})^+ \end{array} \}$$

$$\text{goto}(q_0, \text{dígito}) = \{ \begin{array}{l} \text{entero} \rightarrow (\text{dígito})^+ \\ \text{entero} \rightarrow (\text{dígito})^+ \bullet \\ \text{decimal} \rightarrow (\text{dígito})^* \bullet (\dots)^+ \\ \text{decimal} \rightarrow (\text{dígito})^* \bullet (\text{dígito})^+ \end{array} \} = q_1$$

$$\text{goto}(q_0, \cdot) = \{ \text{decimal} \rightarrow (\text{dígito})^* \bullet (\text{dígito})^+ \} = q_2$$

$$\text{goto}(q_1, \text{dígito}) = q_1$$

$$\text{goto}(q_1, \cdot) = q_2$$

$$\text{goto}(q_2, \text{digito}) = \{ (\text{digito})^* \cdot (\text{digito})^+ \cdot \\ (\text{digito})^* \cdot (\cdot \text{digito})^+ \} = q_3$$

* //Aceptación

$$\text{goto}(q_2, \cdot) = \emptyset \quad // \text{Conjunto que contiene al}$$

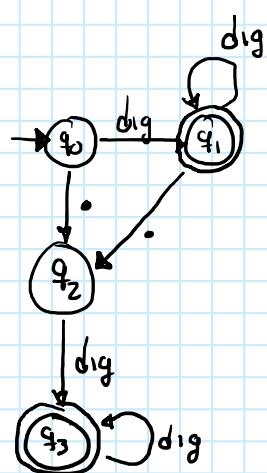
$$\text{conjunto vacío.}$$

$$\text{goto}(q_3, \text{digito}) = q_3$$

$$\text{goto}(q_3, \cdot) = \emptyset$$

//Tabla de transiciones

	<u>digito</u>	<u>punto</u>
$\rightarrow q_0$	q_1	q_2
q_1^*	q_1	q_2
q_2	q_3	\emptyset
q_3^*	q_3	\emptyset



- Automata que reconoce las palabras reservadas int, float, char.

$I \rightarrow \text{int}$ $F \rightarrow \text{float}$ $C \rightarrow \text{char}$	$q_0 = \{ I \rightarrow \cdot \text{int} \cdot \\ F \rightarrow \cdot \text{float} \cdot \\ C \rightarrow \cdot \text{char} \}$
---	---

$$\text{goto}(q_0, i) = \{ I \rightarrow i \cdot \text{nt} \} = q_1$$

$$\text{goto}(q_0, F) = \{ F \rightarrow f \cdot \text{loat} \} = q_2$$

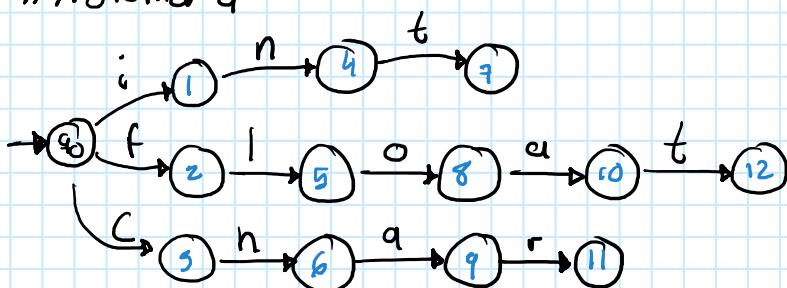
$$\text{goto}(q_0, c) = \{ C \rightarrow c \cdot \text{har} \} = q_3$$

$$\text{goto}(q_1, n) = \{ I \rightarrow \text{in} \cdot \text{t} \} = q_4$$

$$\text{goto}(q_2, l) = \{ F \rightarrow \text{f} \cdot \text{loat} \} = q_5$$

$\text{goto}(q_2, l) = \{ F \rightarrow \text{float} \} = q_5$
 $\text{goto}(q_3, h) = \{ C \rightarrow \text{char} \} = q_6$
 $\text{goto}(q_4, t) = \{ I \rightarrow \text{int} \cdot \} = q_7^*$
 $\text{goto}(q_5, o) = \{ F \rightarrow \text{float} \} = q_8$
 $\text{goto}(q_6, a) = \{ C \rightarrow \text{char} \} = q_9$
 $\text{goto}(q_8, q) = \{ F \rightarrow \text{float} \} = q_{10}$
 $\text{goto}(q_9, r) = \{ C \rightarrow \text{char} \cdot \} = q_{11}^*$
 $\text{goto}(q_{10}, +) = \{ F \rightarrow \text{float} \cdot \} = q_{12}^*$

// Autómata



$$\Sigma = \{ a, c, f, h, i, l, n, o, r, t \}$$

- Operadores lógicos, relacionales y de asignación

$\text{Id} \rightarrow \underbrace{[a-z A-Z]}_{\text{letra}} \underbrace{[0-9]}_{\text{digito}}^*$

Relacionales → $> | > | < | <= | != | ==$
 Asignación → $= | -= | += | *= | /= | \% =$

$m\text{Relacion} \rightarrow [< >] (=)? \mid [=!] =$

$m\text{Asigna} \rightarrow [+ - * / \%] ? =$

Lógicos → $\& \& \mid || \mid !$

Lógicos $\rightarrow \& \& | || | !$

$$q_0 = \{ Id \rightarrow \cdot [a-zA-Z] [a-zA-Z_0-9]^* \\ R \rightarrow \cdot \&\& | || | ! \\ R \rightarrow \&\& | \cdot || | ! \\ R \rightarrow \&\& | || | \cdot ! \\ A \rightarrow (\cdot [+-* \%])? = \\ A \rightarrow ([+-* \%])? \cdot = \\ L \rightarrow \cdot [<>] (=)? | = = | ! = \\ L \rightarrow [<>] (=)? | = = | \cdot ! = \}$$

$$\text{goto}(q_0, [a-zA-Z]) = \{ Id \rightarrow [a-zA-Z] (\cdot [a-zA-Z_0-9])^* \\ Id \rightarrow [a-zA-Z] ([\dots])^* \cdot \} = q_1^*$$

$$\text{goto}(q_0, [+-* \%]) = \{ A \rightarrow ([+-* \%])? \cdot = \} = q_2$$

$$\text{goto}(q_0, =) = \{ A \rightarrow ([+-* \%])? \cdot = \\ L \rightarrow [<>] (=)? | = \cdot = | ! = \} = q_3$$

$$\text{goto}(q_0, [<>]) = \{ L \rightarrow [<>] (=)? | = = | ! = \\ L \rightarrow ([<>] (=)? | = = | ! =) \cdot \} = q_4^*$$

$$\text{goto}(q_0, !) = \{ L \rightarrow [<>] (=)? | = = | ! \cdot = \\ R \rightarrow (\&\& | || | !) \cdot \} = q_5^*$$

$$\text{goto}(q_0, \&) = \{ R \rightarrow (\& \& | || | !) \} = q_6$$

$$\text{goto}(q_0, |) = \{ R \rightarrow (\&\& | \cdot | | ;) \} = q_7$$

$$\text{goto}(q_1, [a-zA-Z_0-9]) = q_1$$

$$\text{goto}(q_2, =) = \{ A \rightarrow ([+-* \%])? \cdot = \} = q_8^*$$

$$\text{goto}(q_3, =) = \{ L \rightarrow ([<>] (=)? | = = | ! =) \cdot \} = q_9^*$$

$\text{goto}(q_4, =) = q_9$

$\text{goto}(q_5, =) = q_9$

$\text{goto}(q_6, \&) = \{ (\& | 1 | ! | !) \cdot \} = q_{10}^*$

$\text{goto}(q_2, 1) = q_{10}$

AFND

Wednesday, August 28, 2019

1:08 PM Valdez Uraga

- Obtener el autómata para reconocer las direcciones IP.

// Expresión Regular

• Cadenas	128.205.1.1	• No	256.0.0.3. <u>4</u>
(Ejemplos)	0.0.0.0	válidas	

Decenas $\rightarrow [1-9]^? [0-9]$

Cien $\rightarrow 1 [0-9][0-9]$

DosC $\rightarrow 2 ([0-4][0-9] | 5[0-5])$

octeto $\rightarrow (\text{cero} | \text{uno} | \text{dos})$

IP $\rightarrow \text{octeto}.\text{octeto}.\text{octeto}.\text{octeto}$

$$q_0 = \{ \begin{array}{l} \text{cero} \rightarrow \cdot [1-9]^? [0-9] \\ \text{cero} \rightarrow [1-9]^? \cdot [0-9] \\ \text{uno} \rightarrow \cdot 1 [0-9][0-9] \\ \text{dos} \rightarrow \cdot 2 ([0-4][0-9] | 5[0-5]) \end{array}$$

$$\text{goto}(q_0, 2) = \{ \begin{array}{l} \text{dos} \rightarrow 2 ([0-4][0-9] | 5[0-5]) \\ \text{dos} \rightarrow 2 ([0-4][0-9] | \cdot 5[0-5]) \\ \text{cero} \rightarrow 2 \cdot [0-9] \\ \text{cero} \rightarrow 2 \cdot \} = q_1^* \}$$

$$\text{goto}(q_0, 1) = \{ \begin{array}{l} \text{cero} \rightarrow 1 \cdot [0-9] \\ \text{cero} \rightarrow 1 \cdot \\ \text{uno} \rightarrow 1 \cdot [0-9][0-9] \end{array} \} q_2^*$$

cero⁻¹

uno $\rightarrow 1 \cdot [0-9][0-9] \} q_2^*$

goto($q_0, [3-9]$) = { cero $\rightarrow ([3-9])? \cdot [0-9]$
cero $\rightarrow [1-9]? [3-9] \cdot \} = q_3^*$

goto($q_0, 0$) = { cero $\rightarrow [1-9]? 0 \cdot \} = q_4^*$

goto($q_1, [0-4]$) = { dos $\rightarrow 2 [0-4] \cdot [0-9]$
cero $\rightarrow 2 [0-4] \cdot \} = q_5^*$

goto($q_1, 5$) = { dos $\rightarrow 25 \cdot [0-5]$
cero $\rightarrow 25 \cdot \} = q_6^*$

goto($q_1, [6-9]$) = { cero $\rightarrow 2 [6-9] \cdot \} = q_7^*$

goto($q_2, [0-9]$) = { cero $\rightarrow 1 [0-9] \cdot$
uno $\rightarrow 1 [0-9] \cdot [0-9] \} = q_8^*$

goto($q_3, [0-9]$) = { cero $\rightarrow ([1-9]? [0-9] \cdot \} = q_9^*$

goto($q_5, [0-9]$) = { dos $\rightarrow 2 [0-4] [0-9] \cdot \} = q_{10}^*$

goto($q_6, [0-5]$) = q_{10}^*

// Código en Lex

%{

#include < stdio.h >

#include < stdlib.h >

%}

cero [1-9]? [0-9]

uno 1 [0-9] [0-9]

dos 2 [0-4][0-9] | 5 [0-5]

octeto { cero } | { uno } | { dos }

%option noyywrqp | Sólo si marca error

% // Sección de exp. Regulares

```
{ octeto ("") { octeto } "" { octeto } ":" { octeto }
```

```
{ printf (" La dirección IP es %s \n", yytext); }
```

% // Sección de código de usuarios

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

O cualquier carácter
menos salto linea

" . " opción
alternativa

// Instalación

- sudo apt-get install flex

- flex archivo.
.lex
.flex

- gcc lex.yy.c -o ejecutable -lfl

- ./ejecutable

NOTA

- Lex retorna 0, no puedes retornar ese valor

- Token una clase Léxica y un valor opcional

```
%}
```

```
#include <stdio.h>  
#include "token.h"  
#include <string.h>
```

```
%}
```

```
%option yylineno
```

```
id [a-zA-Z_][a-zA-Z0-9_]*
```

Ejemplo de
código

id [a-zA-Z_][a-zA-Z0-9_]*

dig [0-9]

%%

```
int { return INT; }
float { return FLOAT; }
{ id } { strcpy(value, yytext);
          return ID;
      }
```

[\t\n] { } // Esto sirve para ignorar
• { printf("Error léxico en la línea %i\n", yylineno); }

%%

```
int yywrap()
    return 1;
```

```
}
```

```
int main(int argc, char **argv){
```

```
}
```

Mínimizar

Monday, September 2, 2019

1:10 PM



- Si 2 estados están separados no se pueden juntar
- Si un estado se separa no se pueden volver a juntar

Ejercicio *Aceptación

Estado	a	b
α { A }	B	C
	B	D
	B	C
	B	E
β { E* }	B	D

//Dividir en 2 grupos

Estado	a	b
α { A }	α	α
	α	α
	α	α
	α	β
β { E* }	α	α

① Estados Iguales juntas, diferentes separadas

Estado	a	b
α { A }	α	α
	α	β
	α	α
	α	γ
β { E }	α	β

Nota: Repetimos proceso

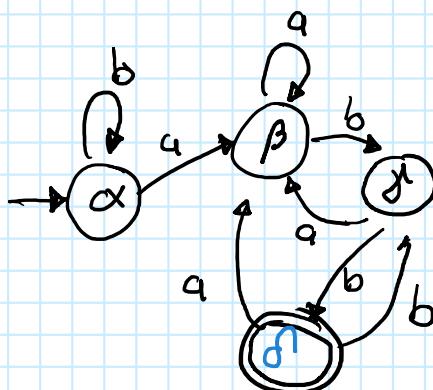
② Estados Iguales juntos, diferentes separados

Estado	a	b	
α	A	B	α
C	β	α	
β	B	β	δ
D	β	S	
E	P	δ	

Nota: Repetimos proceso

Nota 2: No se puede dividir más

∴ Automat Minimo



// Tabla pero números

	a	b
$\rightarrow 0$	1 0	
1	1 2	
2	1 3	
3^*	1 2	

RECUERDA

- $c[0] = '\backslash 0'$; \Rightarrow Fin de cadena
- Unión vs Estructura
- Unión sólo existe uno de sus datos
- Estructura todos existen.

// Declaración Typedef

```

typedef union _estruc {
    // Datos
} estruc;
  
```

// Instancia
estruc value;

```

%{
#include <stdio.h>
%}

%$ uno /* Estado inclusivo */
%# dos /* Estado exclusivo */

%%
```

• Estados de LEX (Exclusivos)

- INITIAL
- COMMENT
- STRING

Diagrama

```

<uno>(aa|bb) {printf ("%s\n", yytext);}
<uno> {
    a*b {printf ("%s\n", yytext);}
    a?cbb{printf ("%s\n", yytext);}
    ab {printf ("%s\n", yytext);}
    // Marcaría error ab ya que está contenido en a*b :: si queremos hacer el caso particular
    d {BEGIN(dos);}
}
```

```

<dos> | (ab)+ {printf ("%s\n", yytext);}
        cd* | " + " {printf ("%s\n", yytext);}
        e {BEGIN(INITIAL)}
```

```

<INITIAL> {
    [A-Z]+ {BEGIN(uno);}
    [a-z]+ {BEGIN(dos);}
}
```

```

[ \n\t]+ {}
    {printf("Error léxico con %s \r")}
```

Línea 1
· { printf("Error Léxico con %s \r"); }

// ^ a que empiece con a
// a \$ que termine con a

% %

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

REPASO

Wednesday, September 11, 2019 1:31 PM

► Ejercicio

	a	b		a	b		a	b
q_0	q_1	q_2	q_0	α	α	q_0	β	β
q_1	q_2	q_3	q_1	α	β	q_1	β	γ
q_2	q_1	q_3	q_2	α	β	q_2	β	γ
q_3^*	q_4	q_5	q_3^*	β	β	q_3^*	δ	δ
q_4^*	ϕ	ϕ	q_4^*	ϕ	ϕ	q_4^*	ϕ	ϕ
q_5^*	ϕ	ϕ	q_5^*	ϕ	ϕ	q_5^*	ϕ	ϕ

EXAMEN Teo

Tuesday, September 17, 2019 1:02 PM

- Traductor : Programa que transforma lenguaje de entrada a otro de salida

► Ensamblador

Programa en
L. ensamblador



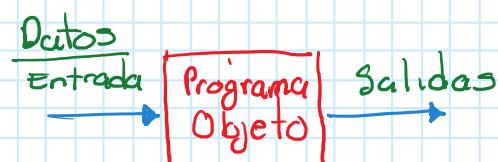
Programa en
L. Maquina

► Compilador

Programa
Alto nivel



Código
Objeto



► Interpretante

Programa
fuente

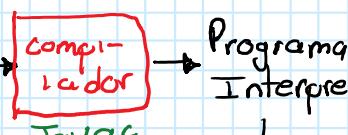


Resultados

Datos

► Maquina Virtual

Source
Code



Programa
Interprete

Datos



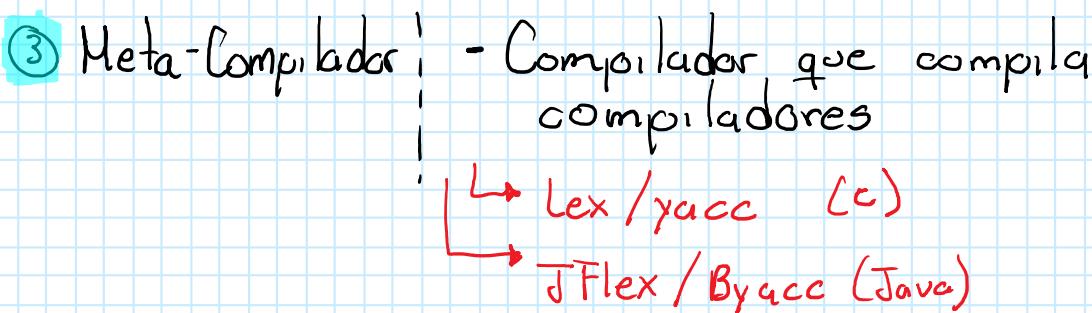
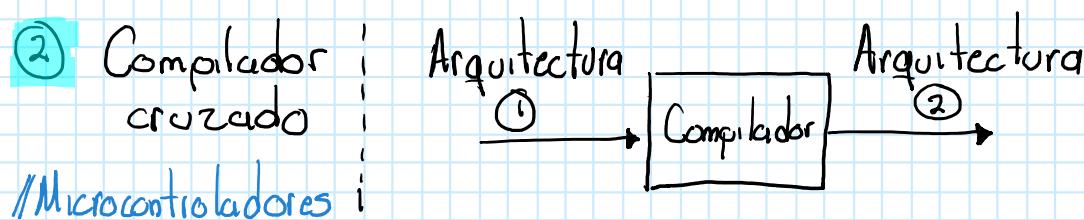
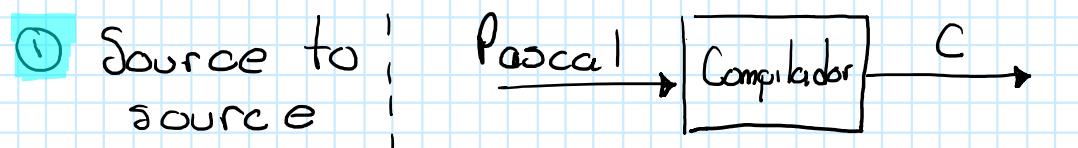
Salida

- | | |
|--------------|--|
| • Compilador | <ul style="list-style-type: none">- Programa Objeto Independiente- Errores en tiempo de compilación- Oculta código fuente- Se pueden crear virus <p>// No necesariamente ejecutable</p> |
|--------------|--|

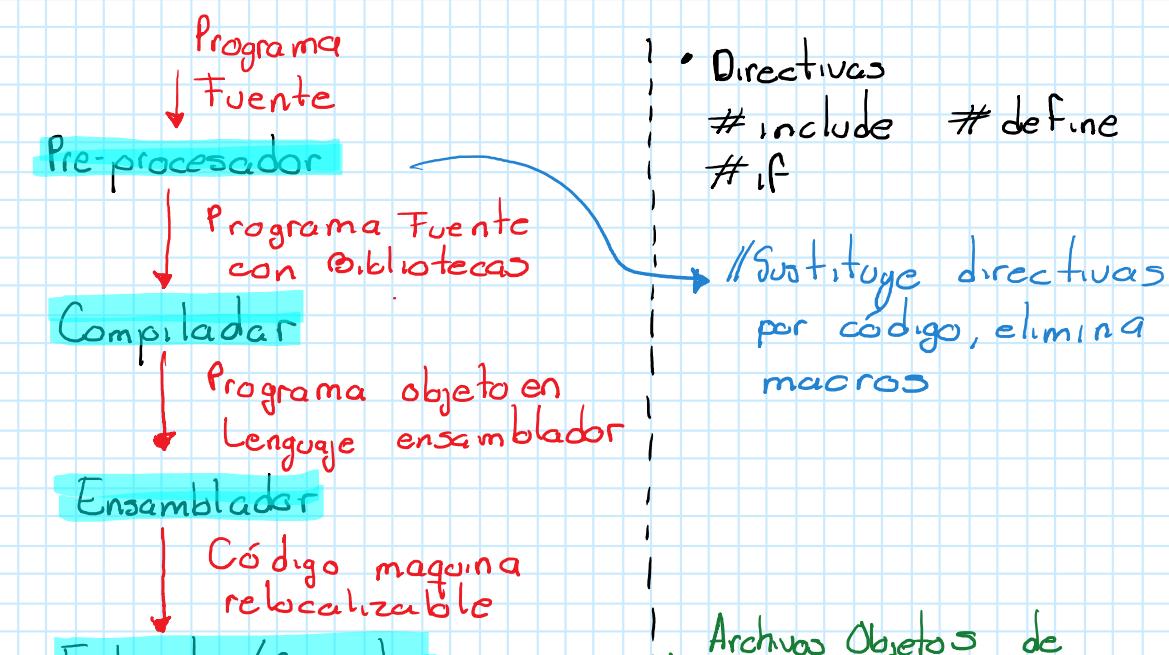
- | | |
|-----------------|--|
| • Interpretante | <ul style="list-style-type: none">- No genera programa objeto- Detecta errores en tiempo de ejecución |
|-----------------|--|

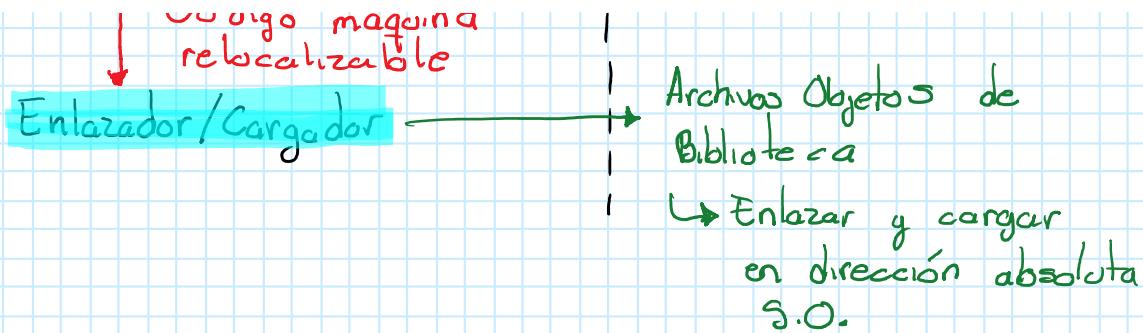
- Maquina Virtual
 - * Es más lento que un compilador
 - * Es más rápido que interprete
 - Es portable (Necesario Interpretar)
 - Detecta errores en Comp y ejecución
 - Es más robusto

► Tipos de Compiladores



► Sistema de Procesamiento del Lenguaje





► Fases de Compilación

► Front-End o
Etapa de análisis

- ↓ Programa Fuente
- ① Análisis Léxico → Revise la ortografía // La hace un AFD
 - ↓ Tokens
 - ① Palabras reservadas bien
 - ② Que se escriba bien
- ② Análisis Sintáctico → Que sigue después de cada cosa // Automata de pila
 - ↓ Árbol Sintáctico
- ③ Análisis Semántico → Validar la coherencia de los lenguajes
 - ↓ Árbol Sintáctico Anotado
 - ① Que no haga tonterías
- ④ Generación de Código Intermedio

- ↓ Código Intermedio
- ⑤ Optimización de Código Intermedio
 - ↓ Código Intermedio
- ⑥ Generación de Código Objeto
 - ↓ Código Objeto
- ⑦ Optimización de Código Objeto
 - ↓ Código Objeto

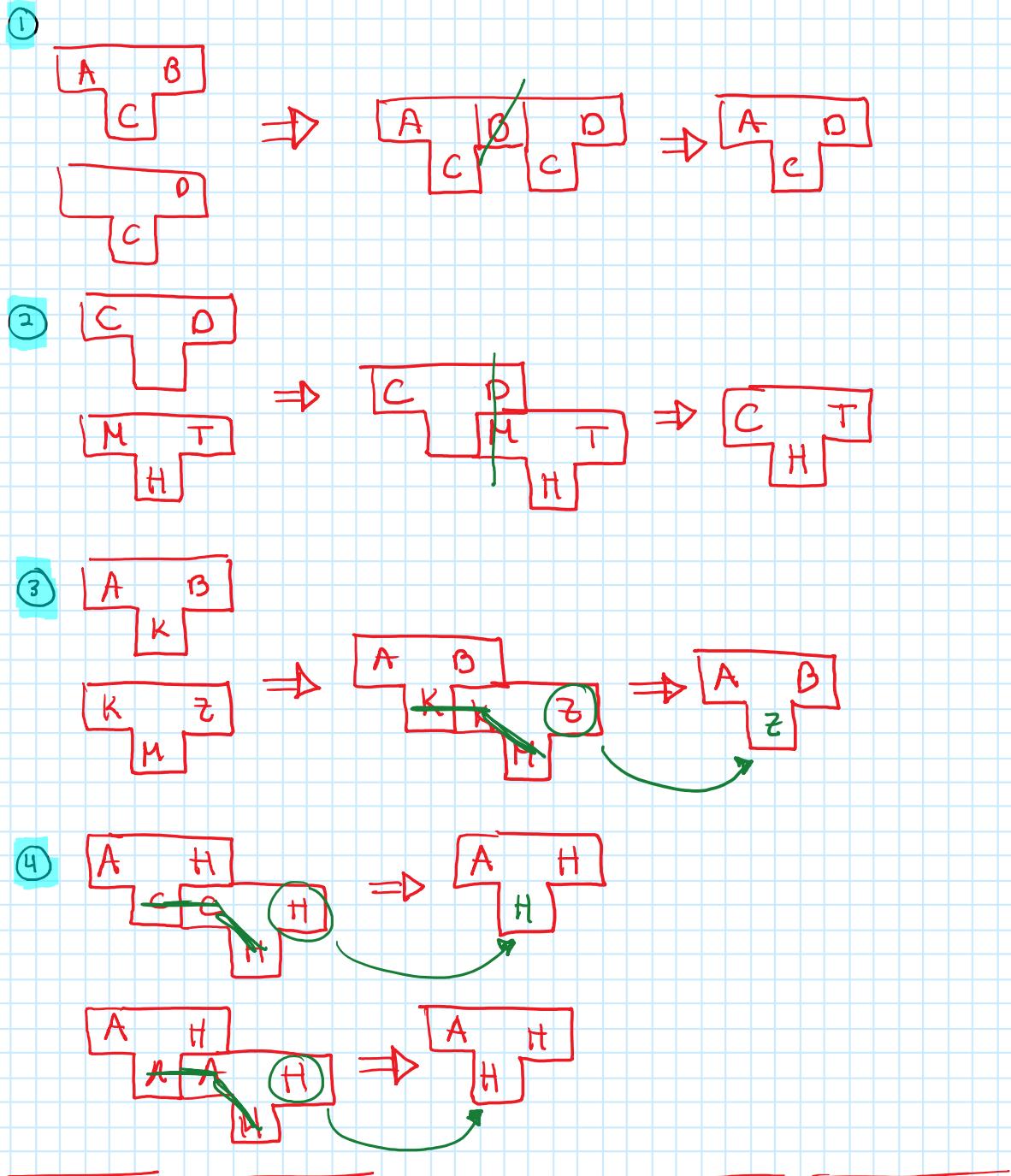
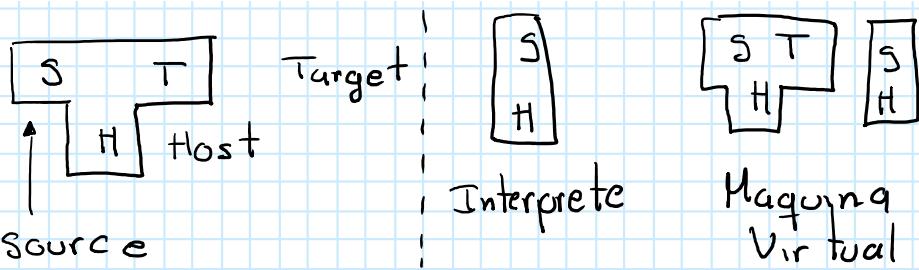
► Back-End o
Etapa de síntesis

- ↓ Archivos Objetos de Biblioteca
- ↓ Enlazar y cargar en dirección absoluta S.O.

► Extra, Estructura Datos

- Tabla Símbolos
 - ↳ Información ID
- Tabla Tipos
 - ↳ Tipos nativos y definidos por el programa // Arreglos / Apuntadores
- Manejador de errores

► Diagrama T



Análisis Léxico

- ① Leer programa fuente
- ② Devolver tokens
- ③ Detectar errores léxicos
- ④ Recuperarse errores

(4) Recuperarse errores

(5) Ignorar espacios en blanco y comentarios

↓
Cualquier símbolo que no pertenece al lenguaje

► Tipos de errores Léxicos

① Palabras reservadas
(mal escritas)

② Identificadores | 1 variable
(mal formados)

③ Identificadores | C = 31 caracteres
(exceden longitud)

④ Constantes numéricas | 3,14.15
(mal escritas)

⑤ Constantes numéricas | short = 333000
(Fuera rango)

⑥ Constantes de cadenas | "holá"
(Mal escritas)

⑦ Comentarios que no finalizan | /* Todo es un comentario

► Recuperación errores

• Modo Pánico { Encontrado un error, se omite hasta token nuevo

• Insertar/Eliminar carácter en la entrada

• Transponer 2 caracteres

► Analizador Léxico / Lexer

- ① Definir tokens
- ② Expresiones R. para tokens
- ③ Obtener AFD que reconozca E. Regulares
- ④ Minimizar AFD
- ⑤ Programar el automata

- Token | • Unidad mínima léxica que tiene un significado en el lenguaje
 - | • Se compone de
 - | < CLASE Valor >
 - | LÉXICA,

- Clases de Token |
 - | ① Palabras reservadas
 - | ② Identificadores
 - | ③ Constantes num.
 - | ④ Operadores
 - | ⑤ Símbolos de pun
 - | ⑥ Cont. Cadenas
 - | ⑦ Comentarios
 - | ⑧ Espacios Blanco

EXAMEN PRÁCTICO

Wednesday, September 18, 2019 2:28 AM

► Expresiones Regulares

- Es un patrón de búsqueda para verificar si una cadena pertenece a un lenguaje.

① Identificadores en C

letra → [a-zA-Z]

letra_ → _ | letra

dígito → [0-9]

id → letra_ (letra_|dígito)*

② Número par de "a" y "b"

first → bb|aa

second → ab|ba

general → (first | second)(first)* (second))*

③ Números flotantes

dígito → [0-9]

entero → (dígito)+

exponente → [E e][+ -]? entero

flotante → (entero"."? entero)? exponente

| entero?"." entero exponente?

| entero. entero? exponente?) [Ff]

// Alternativ a

$[O-q]^* \left(\cdot \cdot [O-q]^+ \right)^? \left([Ee][+-][O-q]^+ \right)^? [FF]$

II Ejercicios

1) letras que no son vocales

$$[b - df - h_j - \bar{n} p - tv - z B - DF - HJ - \bar{N} P - TV - z]$$

② Hexadecimals

$$[O-qA-F_{a-f}]^+$$

3 Octale 5

[0-7]

④ Operadores Aritméticos

$$[t - * / \%]$$

a) Select, from, where → mayúscula/minúsculas

Aux 1 → [e E]

$$A_{\text{UX2}} \rightarrow [r \ R]$$

final $\rightarrow [sS] Aux, [lL] Aux, [cC][+T]$

| from | where

* En su forma correcta

b) secuencia → $[\wedge \{ \}]^*$ | Todos los caracteres
menos los mencionados

d) Orden Léxico Gráfico Ascendente

$$c_4 d_4 \mapsto [aA]^* [bB]^* \dots [zZ]^*$$

e) Comentarios largos en C que no contengan

(e) Comentarios largos en C que no contengan /* intermedio CTECHAR

espace $\rightarrow ["\text{ " } \backslash n \backslash t"]$

cadena $\rightarrow [i"\text{ " }\backslash n \backslash t]$

coment $\rightarrow /* \text{ espacio}^* (\text{cadena}^* \text{ espacio}^*)^* */$

(f) Números Binarios múltiplos de 4

16 8 4 2 1 ; NOTA 0 es múltiplo de 4
 Multiplos de 4 0 0 ; $\therefore \text{Aux} \rightarrow [10]$
 bin $\rightarrow \text{Aux}^* 00$

A F D

$(a|b)^* ((c|d)(a|b)^* (c|d)(a|b)^*)^*$

$q_0 = \{ (a|b)^* (\dots)^* \leftarrow$
 $(a|b)^* (\dots)^* \leftarrow$
 $(\dots)^* ((c|d)(\dots)^* (\dots)^*)^* \leftarrow$
 $(\dots)^* ((c|d)(\dots)^* (\dots)^*)^*$
 $(\dots)^* (\dots)^* \cdot \} \text{ /Final}$

• $\text{goto}(q_0, a) = \{ (a|b)^* (\dots)^*$
 $(a|b)^* (\dots)^*$
 $(\dots)^* ((c|d)(\dots)^* (\dots)^*)^* \}$

$$\begin{aligned}
 & (\dots)^* ((\cdot c \mid d) (\dots)^* (\dots)(\dots)^*)^* \\
 & (\dots)^* ((c \mid d) (\dots)^* (\dots)(\dots)^*)^* \\
 & (\dots)^* (\dots)^* \cdot \} = q_0 \text{ //Final}
 \end{aligned}$$

• $\text{goto}(q_0, b) = q_0 \text{ //Final}$

$$\begin{aligned}
 \text{goto}(q_0, c) = & \left\{ (\dots)^* ((c \mid d) (\cdot a \mid b)^* (\dots)(\dots)^*)^* \right. \\
 & (\dots)^* ((c \mid d) (a \mid b)^* (\dots)(\dots)^*)^* \\
 & (\dots)^* ((\dots) (\dots)^* ((c \mid d) (\dots)^*)^* \\
 & \left. (\dots)^* ((\dots) (\dots)^* ((c \mid d) (\dots)^*)^*)^* \right\} = q_1
 \end{aligned}$$

$\text{goto}(q_0, d) = q_1$

$\text{goto}(q_1, a) = q_1$

$\text{goto}(q_1, b) = q_1$

$$\begin{aligned}
 \text{goto}(q_1, c) = & \left\{ (\dots)^* ((\dots) (\dots)^* (\dots) (\cdot a \mid b)^*)^* \right. \\
 & (\dots)^* ((\dots) (\dots)^* (\dots) (a \mid b)^*)^* \\
 & (\dots)^* ((c \mid d) (\dots)^* (\dots) (\dots)^*)^* \\
 & \left. (\dots)^* ((c \mid d) (\dots)^* (\dots) (\dots)^*)^* \right\} \\
 & (\dots)^* (\dots)^* \cdot \} = q_2 \text{ //Final}
 \end{aligned}$$

$\text{goto}(q_2, a) = q_2$

$\text{goto}(q_2, b) = q_2$

$\text{goto}(q_2, c) = q_1$

$\text{goto}(q_2, d) = q_1$

II Tabla de transiciones

	a	b	c	d
$\rightarrow q_0^*$	q_0	q_0	q_2	q_1
q_1	q_1	q_1	q_2	q_2
q_2^*	q_2	q_2	q_1	q_1

► Minimizar Autómato

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
$\beta\{E^*\}$	B	D

\Rightarrow

	a	b
A	α	α
B	α	α
C	α	α
D	α	β
$\beta\{E^*\}$	α	α

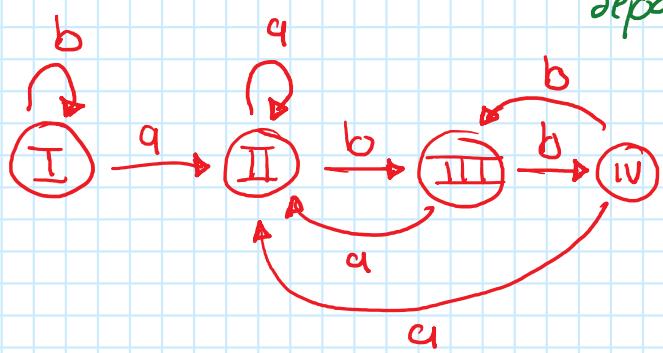
	a	b
I	I	I
II	I	II
III	I	III
$\text{IV}\{E^*\}$	I	II

\Rightarrow

	a	b
A	II	I
C	II	I
B	II	II
D	II	II
$\text{IV}\{E^*\}$	II	III

↑ Separado al ser final

depende do seu final



EXAMEN PRÁCTICO

Wednesday, September 18, 2019 11:31 AM

a) $ab(a|b)^*b$

$$q_0 = \{ \cdot ab(a|b)^*b \}$$

$$\text{goto}(q_0, a) = \{ a \cdot b (a|b)^* b \} = q_1$$

$$\begin{aligned} \text{goto}(q_1, b) &= \{ ab(\cdot a|b)^* b \\ &\quad ab(a \cdot b)^* b \\ &\quad ab(a|b)^* \cdot b \} = q_2 \end{aligned}$$

$$\begin{aligned} \text{goto}(q_2, a) &= \{ ab(\cdot a|b)^* b \\ &\quad ab(a \cdot b)^* b \\ &\quad ab(a|b)^* \cdot b \} = q_2 \end{aligned}$$

$$\begin{aligned} \text{goto}(q_2, b) &= ab(a \cdot b)^* b \\ &\quad ab(\cdot a|b)^* b \\ &\quad ab(a|b)^* \cdot b \\ &\quad ab(\dots)^* b \circ // \text{Final } \{ q_3 \} \end{aligned}$$

$$\text{goto}(q_3, a) = q_2$$

$$\text{goto}(q_3, b) = q_3$$

// Tabla de transiciones

	a	b	
q0	q1	-	
q1	-	q2	
q2	q2	q3	

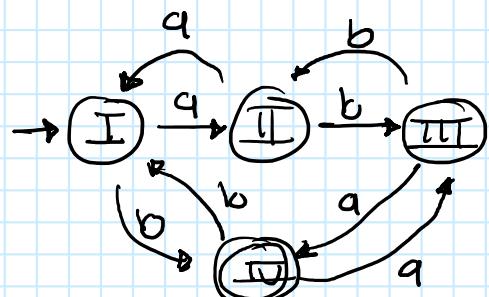
q_2	-	q_2	-
q_3	-	q_3	-
q_2	-	q_3	-
q_3	-	q_2	-

b) Minimizar

q_0	-	a	-	b	-
α	0	α	P		
α	1	α	α		
β	2	β	α		
α	3	α	α		
α	4	α	P		
α	5	α	α		
α	6	α	α		
α	7	α	α		
β	8*	α	α		
α	9*	α	α		
α	10*	α	α		

I	-	a	-	b	-
I	0		II		IV
II	6		II		IV
III	1		I		III
IV	5		I		III
I	7		I		III
III	3		IV		II
IV	2*		III		I
II	4*		II		I
III	8*		II		I
IV	4*		III		I
I	8*		III		I

	-	a	-	b	-
I	II		IV		
II	I		III		
III	IV		II		
IV*	III		I		



c) entero $\rightarrow [0-9]^+$

exponente $\rightarrow [Ee][+-]? \text{ entero}$

flotante $\rightarrow (\text{entero} \cdot ? \text{ entero} ? \text{ exponente} [FF])$

| entero? . entero exponente? [FF]

| entero. entero? exponente? [FF]

| entero. entero? exponente? [F F])

EXAMEN_SIMUL

Monday, September 23, 2019 9:14 AM

1

- a) Preprocesador → Programa sin macros
- b) Compilador → Programa en Len. Ensamblador
- c) Ensamblador → Programa en Len. Maquina
- d) Entrelazador/Cargador → Programa ejecutable
- e) Análisis Léxico → Tokens
- f) A. Sintáctico → Árbol Sintáctico
- g) A. Semántico → Árbol Sintáctico Anotado
- h) Gen. Code Mid → Código Intermedio
- i) Gen. Code Obj → Código Objeto

2

- a) Compilador cruzado: Es aquel que compila programas para una arquitectura distinta a la del origen
- b) Compilador "source to source": Es aquel que como entrada tiene un programa de alto nivel y de salida tiene igual otro de alto nivel
- c) Metacompileador: Es aquel que compila "Compiladores"
- d) JIT: Traduce un programa al mismo tiempo que lo ejecuta (Just in Time).

③ Fases que componen el Back-End y su otro nombre

También conocido como "Etapa de Síntesis"

- Fases que lo comparten :
- ① Optimización Code Mid
 - ② Generación Obj Code
 - ③ Optimización Obj Code

④ Notación Octal | [0-7]

Dígitos hexadecimales [0-9 A-F a-f]

- letras minúsculas que no son vocales : [b-d f-h j-n p-t r-z]

Forma exponencial en C :

$$\begin{aligned} d &\rightarrow [0-9] \\ \text{ex} &\rightarrow [\text{eE}] [+-]? \text{d}^+ \\ \text{f} &\rightarrow (\text{d})^* . (\text{d})^+ (\text{ex})? | (\text{d})^+ . (\text{d})^* (\text{ex})? \\ &| (\text{d})^+ \text{ex} \end{aligned}$$

Para forma compleja :

$$\begin{aligned} \text{c} &\rightarrow (\text{f})? [+-] \text{f}[\text{I};] \quad \uparrow \\ &| \text{f} [+-] ((\text{f})[\text{I};])? \end{aligned}$$

⑤ Fases del Front End y el nombre

• También conocido como Etapa de Análisis

- Fases que lo conforman :
- ① Análisis Léxico
 - ② Análisis Sintáctico
 - ③ Análisis Semántico
 - ④ Generación Código Intermedio

⑥ Errores Léxicos

```

int man(int argc, char** argv) {
    int qa, bc, 4;
    floata b, c;
    a = 'b' + 3276;
    b = 3.5E++6;
    a++;
    return;
}

```

| ① Palabras reservadas
 | "man" y "floata"
 | ② Identificadores mal
 | escritos
 | "qa" y "bc,4"
 | ③ Número mal escrito
 | "3.5E++6"

// Second

```

int main(...)

intt a, bc,4;
float b, qcd;

a ...
b = 3.5E-6.3
a +-+-
return ...

```

- | ① Palabra reservada a
 | "intt"
 | ② Identificadores
 | "bc,4" y "qcd"
 | ③ Cf. Numérica
 | "3,5 E -6.3"
 | ④ Operador *
 | mal escrito
 | +-+-

⑧ Análisis léxico y reporte errores

intt a, bc,4;

<error, intt> → Modo pánico → Continuamos

<ID, a>

<error, bc,4> → Modo pánico → Continuamos

<;>

⋮

TABLA SÍMBOLOS

Pos.	ID	Tipo	Dir
0	a	d?	d?

TABLA DE TIPOS

TIPO	Tipo	TipoBase	Tam
------	------	----------	-----

Pos.	ID	Tipo	Dir	IO	Tipo	TipoBase	Tam
O	a	d?	d?				

⑨ Tabla comparativa Compilador, Interpretar y MV

Compilador	Interpretar	Maquina Virtual
• Se detectan errores en tiempo de compilación	• Se detectan errores en tiempo de ejecución	• Se detectan errores en tiempo de Comp. y Ejecución
• Se oculta el código fuente	• No se puede ocultar el C.F.	• Se puede ocultar código fuente
• Genera código objeto	• Ejecuta el programa fuente	• Genera un programa intermedio y lo interpreta
• Traduce programa en Len: Ensum	• Traduce programa de Alto nivel	• Traduce programa de alto nivel
• Un programa compilado no es portable	• Si es portable (necesita interpretar)	• Si es portable

⑤ AFD

a) $ab(a|b)^* b$

$$q_0 = \{ \cdot ab(a|b)^* b \}$$

$$\text{goto}(q_0, a) = \{ a \cdot b (a|b)^* b \} q_1$$

$$\text{goto}(q_0, b) = \emptyset$$

$$\text{goto}(q_1, a) = \emptyset$$

$$\text{goto}(q_1, b) = \left\{ ab (\cdot a | b)^* b, ab (a | \cdot b)^* b, ab (a | b)^* \cdot b \right\} = q_2$$

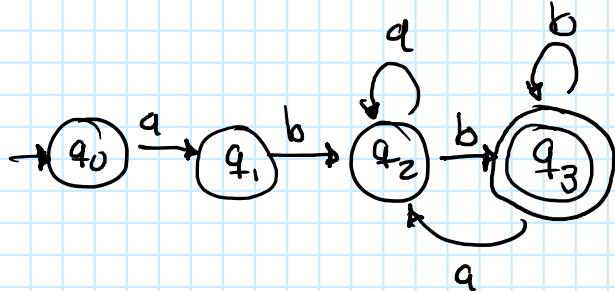
$$\text{goto}(q_2, a) = q_2$$

- $\text{goto}(q_2, b) = \left\{ ab (a | \cdot b)^* b, ab (\cdot a | b)^* b, ab (a | b)^* \cdot b \right\} = q_3$

$$\text{goto}(q_3, a) = q_2$$

$$\text{goto}(q_3, b) = q_3$$

TABLA		
	a	b
$\rightarrow q_0$	q_1	-
q_1	-	q_2
q_2	q_2	q_3
(q_3)	q_2	q_3



b) $x ((a)^* c | b)$

$$q_0 = \left\{ x ((a)^* c | b) \right\}$$

$$\begin{aligned} \text{goto}(q_0, x) = & \left\{ x ((\cdot a)^* c | b), \right. \\ & x ((a)^* \cdot c | b), \\ & \left. x ((a)^* c | \cdot b) \right\} = q_1 \end{aligned}$$

$$x((a)^* c) \cdot b \} = q_1$$

$$\text{goto}(q_0, a) = \emptyset \quad \text{goto}(q_0, b) = \emptyset$$

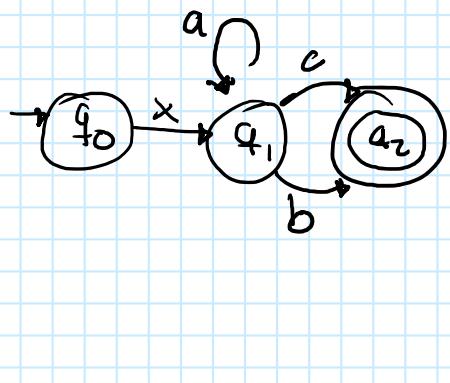
$$\text{goto}(q_0, c) = \emptyset \quad \text{goto}(q_1, x) = \emptyset$$

$$\text{goto}(q_1, a) = q_1$$

$$\text{goto}(q_1, c) = \{ x((a)^* c) \cdot b \} = \{ q_2 \}$$

$$\text{goto}(q_1, b) = \{ q_2 \}$$

$$\text{goto}(q_2, a) = \emptyset \quad \text{goto}(q_2, b) = \emptyset \quad \text{goto}(q_2, c) = \emptyset$$



TABLA

	X	a	b	c
q0	-	-	-	-
q1	-	q1	q2	q2
q2	-	-	-	-

a) $(a)? b^*$

$$q_0 = \{ \cdot (a)? b^* \leftarrow$$

CUIDADO

$$(a)? \cdot b^*$$

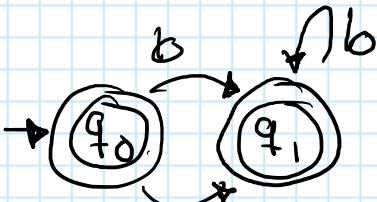
$$(a)? b^* \cdot \}$$

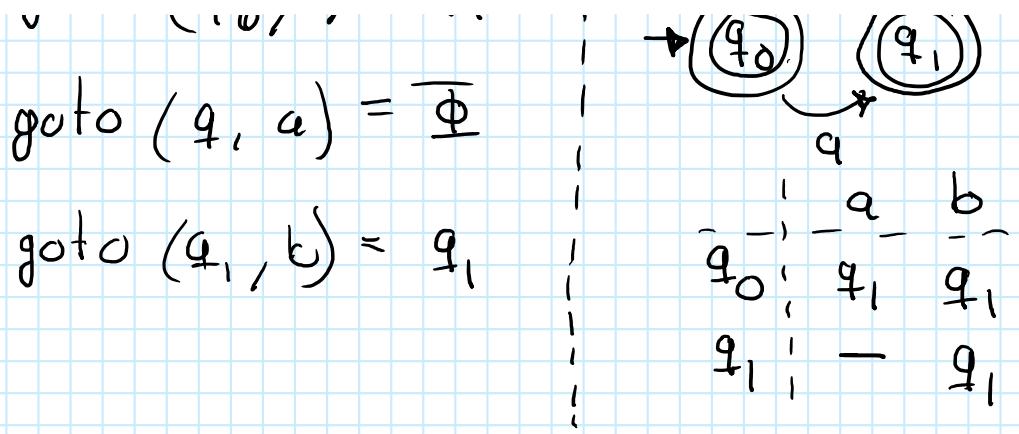
goto(q0, a) = { (a)? (-b)*

$$(a)? (b)^* \cdot } = q_1$$

$$\text{goto}(q_0, b) = q_1$$

$$\dots L_n / n \dots = \overline{\Phi}$$





b) $bq((a)^* 0 | b)^* (a)^+ b$

$$q_0 = \{ \cdot ba((a)^* \dots) \}$$

$\text{goto}(q_0, a) = \underline{\Phi} \quad \text{goto}(q_0, 0) = \underline{\Phi}$
 $\text{goto}(q_0, b) = \{ b \cdot q ((a)^* 0 | b)^* (a)^+ b \} \quad q_1$
 $\text{goto}(q_1, a) = \{ bq ((\cdot a)^* 0 | b)^* (a)^+ b \leftarrow$
 $bq ((a)^* \cdot 0 | b)^* (a)^+ b$
 $\rightarrow bq ((a)^* 0 \cdot b)^* (a)^+ b \quad /$
 $bq ((a)^* 0 | b)^* (\cdot a)^+ b \quad \} = q_2$
 $\text{goto}(q_2, a) = \{ bq ((\cdot a)^* 0 | b)^* (a)^+ b \leftarrow$
 $bq ((a)^* \cdot 0 | b)^* (a)^+ b$
No se puede ~~$bq ((a)^* a \cdot b)^* (q)^+ b$~~
 $\underbrace{(a)^* 0 \cdot b}_{\text{b}} \rightarrow bq ((a)^* 0 | b)^* (a)^+ b \leftarrow$
 $bq ((a)^* 0 | b)^* (a)^+ \cdot b \quad \} = q_3$

$$\text{goto}(q_2, b) = q_2 \quad \text{goto}(q_2, 0) = q_2$$

$$\text{goto}(q_3, a) = q_3$$

$$\text{answ} \sqsubset L = \{ L - 1 \cdot 1^* \cdot 0 \cdot 1 \cdot 1^* \cdot 1 \cdot 1 + L - 2 = q \}$$

$$y_010(4_3, 4) = \overline{q}_3$$

$$\text{goto}(q_3, b) = \{ ba((a)^* 0 | b)^* (a)^+ b \} = q_4$$

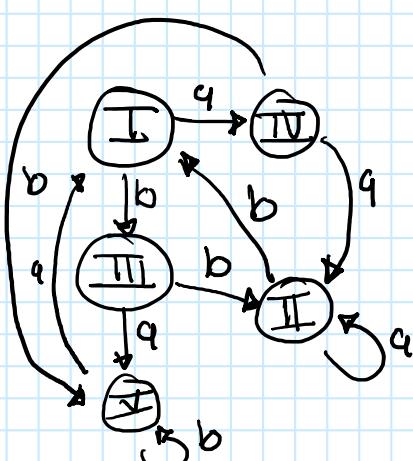
$$\text{goto}(q_3, 0) = q_2$$

(10) Minimizar AFD

	<u>a</u>	<u>b</u>		<u>a</u>	<u>b</u>	
I	0	1	5	0	I	I
II	1	6	2	1	J	III
III	2	0	2	3	II	I
IV	3	2	6	1	I	I
V	4	7	5	5	II	II
VI	5	2	6	6	I	I
VII	6	6	4	7	I	III
VIII	7	6	2	2	I	IV

	<u>a</u>	<u>b</u>		<u>a</u>	<u>b</u>	
I	0	1	III	II		I
II	4	6	III	II		II
III	6	I	IV	I		III
IV	3	5	IV	I		IV
V	5	I	IV	I		V
VI	1	7	I	IV		VI
VII	7	I	I	IV		VII
VIII	IV	2	I	II		VIII

	<u>a</u>	<u>b</u>	
I	0	IV	III
II	4	IV	III
III	3	V	II
IV	5	IV	II
V	I	II	V
VI	7	II	IV
VII	I	II	IV
VIII	6	II	I
IX	II	I	V
X	2	I	IV
XI	1	I	V



$$d \rightarrow [0-a]$$

$$\exp \rightarrow [eE][+-](d)^+$$

$$\text{float} \rightarrow (d)^*. (d)^+ \exp? | (d)^+. (d)^+ \exp? | (d)^+ \exp$$

$d \rightarrow [0-a]$

$\exp \rightarrow [e \in] [+ -] d +$

$\text{float} \rightarrow (d)^* \cdot (d)^+ \exp? |$

$(d)^+ \cdot (d)^* \exp? |$

$d^+ \exp$

EXAMEN SIMUL 2

Wednesday, September 25, 2019 11:48 AM

$$ba((a)^* \cup b)^* (a)^+ b$$

$$q_0 = \{ \cdot ba(\dots)^* (a)^+ b \}$$

$$\text{goto}(q_0, a) = \emptyset \quad \text{goto}(q_0, \cup) = \emptyset$$

$$\rightarrow \text{goto}(q_0, b) = \{ b \cdot a (\dots)^* (a)^+ b \} = q_1$$

$$\text{goto}(q_1, b) = \emptyset \quad \text{goto}(q_1, a) = \emptyset$$

$$\rightarrow \text{goto}(q_1, a) = \{ ba((\cdot a)^* \cup b)^* (a)^+ b$$

$$\xrightarrow{\quad} ba((a)^* \cdot \cup b)^* (a)^+ b$$

$$ba((a)^* \cup \cdot b)^* (a)^+ b$$

$$\rightarrow ba((a)^* \cup b)^* (\cdot a)^+ b \} = q_2$$

$$\rightarrow \text{goto}(q_2, a) = \{ ba((\cdot a)^* \cup b)^* (a)^+ b$$

$$ba((a)^* \cdot \cup b)^* (a)^+ b$$

$$ba((a)^* \cup b)^* (\cdot a)^+ b$$

$$ba((a)^* \cup b)^* (a)^+ \cdot b \} = q_3$$

$$\text{goto}(q_2, b) = q_2$$

$$\text{goto}(q_2, \cup) = q_2$$

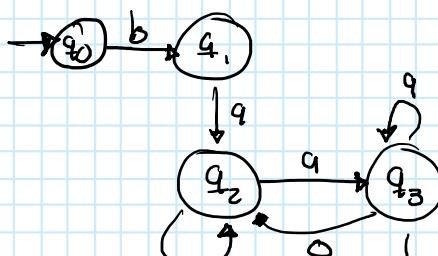
$$\text{goto}(q_3, a) = q_3$$

$$\text{goto}(q_3, \cup) = q_2$$

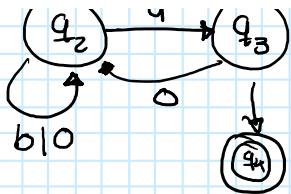
$$\text{goto}(q_3, b) = \{ ba((a)^* \cup b)^* (a)^+ b \cdot \} = \boxed{q_4}$$

TABLA

	a	b	\cup	\cdot
q_0	-	q_1	-	-
q_1	q_2	-	-	-
q_2	q_3	q_2	q_2	-
q_3	q_3	q_4	q_2	-
q_4	-	-	-	-



$\overline{q_3} \mid q_3 \mid \underline{q_4} \mid \underline{q_2} \mid$



- a) Preprocesador → Programa sin macro
- b) Compilador → Programa l. Ensamblador
- c) Ensamblador → Programa l. Maquina
- d) Encabezador/Cargador → Programa ejecutable
- e) Análisis Léxico → Tokens
- f) A. Sintáctico → A. Sintáctico
- g) A Semántico → A. S. Anotado
- h) Gen. Code Mid → Código Intermedio
- i) Gen. Code Obj → Code Obj

- Cruzado ; Es aquel que compila un programa para una arquitectura distinta a la del origen
- Source to Source ; Aquel que como entrada tiene un lenguaje de alto nivel y de salida un programa igual en lenguaje de alto nivel
- Metacompileador ; Es el compilador que genera compiladores
- JIT ; Es aquel que al compilar también se ejecuta (Just in time)
- Hexadecimales ; [0-9 a-fA-F]
- Minúsculas vocales ; [b-d f-h j-n p-t v-z]

- Decimales | $d \rightarrow [0-9]$
- exponencial | $\exp \rightarrow [Ee][+-]d^+$
- | $\text{Float} \rightarrow (d)^*.(d)^+ \exp?$
- | $(d)^+.(d)^* \exp?$
- | $d^+ \exp$

- front-end → Etapa de análisis

- ① Análisis Léxico
- ② Análisis Sintáctico
- ③ Análisis Semántico
- ④ Gene. Code. Inter

- Back-end → Etapa de síntesis

- ① Op. Code. Inter
- ② Gene. Code Obj
- ③ Op. Code Obj

Compilador

Errores en T.
Compilación

Oculta Source
Code

Genera Código
Objeto

El programa
compilado no es
portable

Interprete

Errores en T
de ejecución

No oculta
Source Code

Ejecuta program
de alto nivel

Es portable pero
se necesita
interprete

NV

Errores en
Compila y
ejecución

Oculta
Source Code

Genera code mid
y lo ejecuta

Es portable

Ensamblador

Errores al
ensamblar

Oculta
Source Code

Genera Código
Maquina

No es port
able