

SEMÁNTICA

SEMÁNTICA

- Repaso clase anterior
 - Sintaxis, definición
 - Elementos de la sintaxis
 - Maneras de definirla
 - BNF
 - EBNF
 - Diagramas de flujo
- Semántica
 - Semántica estática
 - Semántica dinámica
- Procesamiento de los programas
 - Intérpretes
 - Compiladores



Sintáxis

■ Pensar:

Como definir una gramática para una expresión con operandos del tipo identificador y números y, que además, quede en ella reflejado el orden de prioridades de las operaciones



REPASO CLASE ANTERIOR

○ Gramática para expresiones usando BNF

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

$N=\{<exp>, <term>, <elem>, <iden>, <letra>\}....$

$T=\{0,1,2,3,4,5,6,7,8,9,a,b,c,.....\}$

$S=\{<exp>\}$

$P=\{<exp> ::= <term> | <term> + <exp> | <term> - <exp>$

$<term> ::= <elem> | <elem> * <term> | <elem> / <term>$

$<elem> ::= <iden> | <núm> | (<exp>)$

$<iden> ::= <letra> | <letra> <sec>$

$<sec> ::= <letra> | <digito> | <digito> <sec> | <letra> <sec>$

$<letra> ::= a|b|c|.....$

$<núm> ::= <digito> | <digito> <núm>$

$<digito> ::= 0|1|2|.....$

}



REPASO CLASE ANTERIOR

○ Gramática para expresiones usando EBNF

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

$N=\{<exp>, <term>, <elem>, <iden>, <letra>\}....$

$T=\{0,1,2,3,4,5,6,7,8,9,a,b,c,.....\}$

$S=\{<exp>\}$

$P=\{<exp> ::= <term>\{(+|-)<term>\}^*$

$<term> ::= <elem>\{(*|/)<elem>\}^*$

$<elem> ::= (<iden> | <núm>)$

$<iden> ::= <letra>\{(<letra> | <dígito>)\}^*$

$<letra> ::= (a|b|c..)$

$<núm> ::= <dígito>\{<dígito>\}^*$

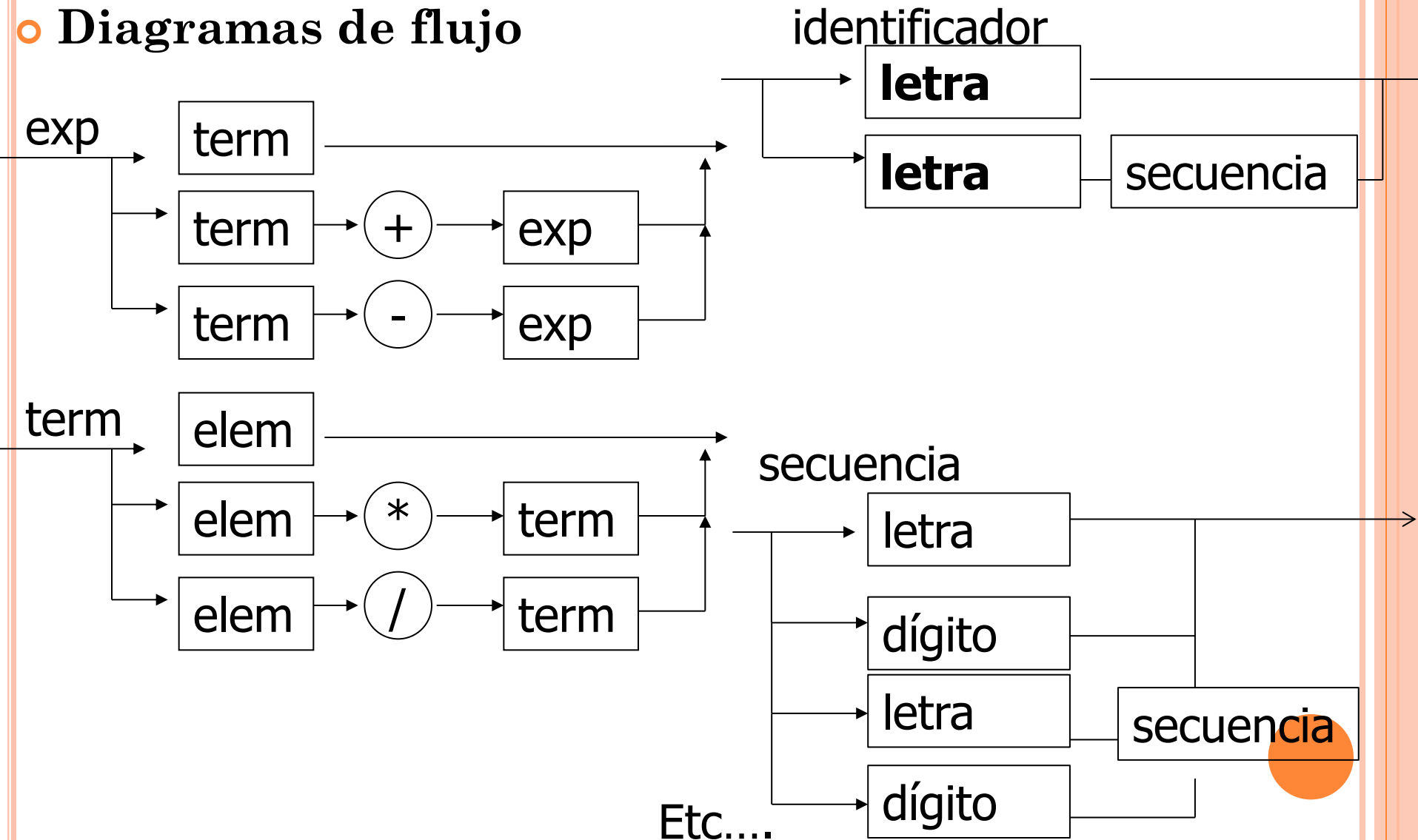
$<dígito> ::= (0|1|2..)$

}



REPASO CLASE ANTERIOR

Diagramas de flujo



SEMÁNTICA

La ***semántica*** describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático

- **Ejemplos:**

- `int vector [10];`
- `if (a<b) max=a; else max=b;`

- **Tipos de semántica**

- Estática
- Dinámica



1

```
main.c
1  #include <stdio.h>
2
3  int main()
4  {
5      int a, resul;
6      char cadena;
7
8      cadena = 'h';
9      resul = a + x;
10     printf(resul);
11     return 0;
12 }
13
```

2

```
main.c
1  #include <stdio.h>
2
3  int main()
4  {
5      int a, resul;
6      char cadena;
7
8      cadena = 'h';
9      resul = a + cadena;
10     printf(resul);
11     return 0;
12 }
```

3

```
main.c
1  #include <stdio.h>
2
3  int main()
4  {
5      int a;
6      int a, b;
7
8      b = a * 2;
9      printf(b);
10     return 0;
11 }
```

1. La variable x no fué declarada.
2. En una expresión se combinan diferentes tipos de datos y no hay reglas que lo permitan o lo resuelvan.
3. Se declararon 2 variables con el mismo nombre en un mismo entorno.

SEMÁNTICA

Semántica estática

- No está relacionado con el significado del programa, está relacionado con las formas validas.
- Se las llama así porque el análisis para el chequeo puede hacerse en compilación.
- Para describir la sintaxis y la semántica estática formalmente sirven las denominadas gramáticas de atributos, inventadas por Knuth en 1968.
- Generalmente las gramáticas sensibles al contexto resuelven los aspectos de la semántica estática.



SEMÁNTICA

Semántica estática - Gramática de atributos

- A las construcciones del lenguaje se le asocia información a través de los llamados “**atributos**” asociados a los símbolos de la gramática correspondiente
- Los valores de los atributos se calculan mediante las llamadas “**ecuaciones o reglas semánticas**” asociadas a las producciones gramaticales.
- La evaluación de las reglas semánticas puede:
 - Generar Código.
 - Insertar información en la Tabla de Símbolos.
 - Realizar el Chequeo Semántico.
 - Dar mensajes de error, etc.




SEMÁNTICA

Semántica estática - Gramática de atributos

Los atributos están directamente relacionados con los símbolos gramaticales (terminales y no terminales)

La forma, general de expresar las gramáticas con atributos se escriben en forma tabular. Ej:

Regla gramatical	Reglas semánticas
Regla 1	Ecuaciones de atributo asociadas
.	.
.	.
Regla n	Ecuaciones de atributo asociadas



SEMÁNTICA

Semántica estática - Gramática de atributos

- Ej. Gramática simple para una declaración de variable en el lenguaje C. Atributo **at**

Regla gramatical

decl* \rightarrow *tipo lista-var

***tipo* \rightarrow int**

***tipo* \rightarrow float**

lista-var* \rightarrow *id

***lista-var*₁ \rightarrow *id*, *lista-var*₂**

Reglas semánticas

lista-var.at* = *tipo.at

***tipo.at* = int**

***tipo.at* = float**

id.at* = *lista-var.at

Añadetipo(id.entrada, lista-var.at)

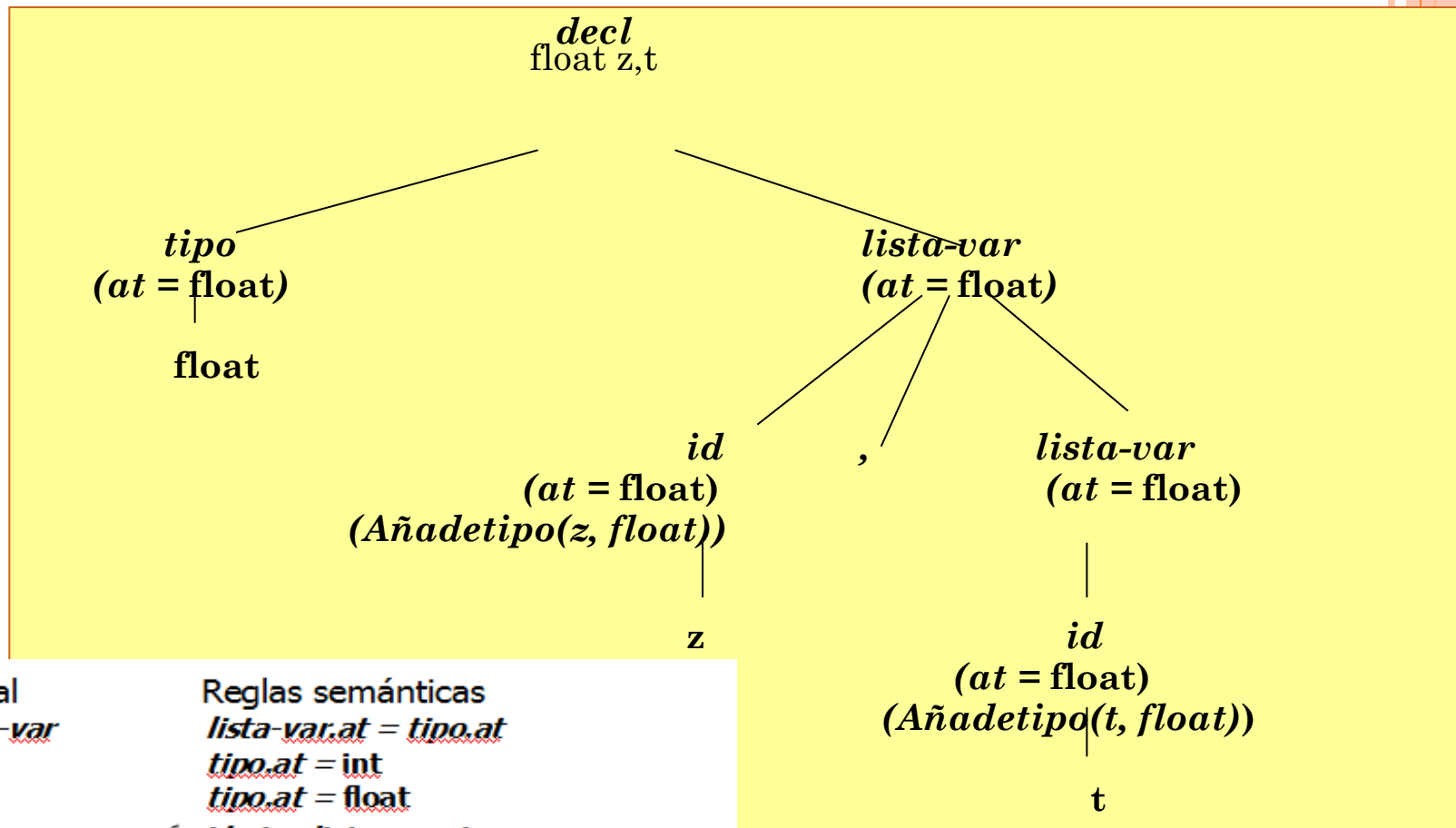
id.at* = *lista-var.at

Añadetipo(id.entrada, lista-var₁.at)

lista-var*₂.at = *lista-var.at



- El árbol sintáctico que muestra los cálculos de atributo para la declaración: "float z,t"



Regla gramatical
 $decl \rightarrow tipo \ lista-var$
 $tipo \rightarrow int$
 $tipo \rightarrow float$
 $lista-var \rightarrow id$

$lista-var_1 \rightarrow id, lista-var_2$

Reglas semánticas
 $lista-var.at = tipo.at$
 $tipo.at = int$
 $tipo.at = float$

$id.at = lista-var.at$
 $Añadirtipo(id.entrada, lista-var.at)$
 $id.at = lista-var.at$
 $Añadirtipo(id.entrada, lista-var_1.at)$
 $lista-var_2.at = lista-var.at$

SEMÁNTICA

Semántica dinámica.

- Es la que describe el efecto de ejecutar las diferentes construcciones en el lenguaje de programación.
- Su efecto se describe durante la ejecución del programa.
- Los programas solo se pueden ejecutar si son correctos para la sintáxis y para la semántica estática.



SEMÁNTICA

¿Cómo se describe la semántica?

- No es fácil
- No existen herramientas estándar como en el caso de la sintaxis (diagramas sintácticos y BNF)
- Hay diferentes soluciones formales:
 - Semántica axiomática
 - Semántica denotacional
- Semántica operacional



SEMÁNTICA

○ Semántica axiomática

- Considera al programa como “**una máquina de estados**”.
- La notación empleada es el “**cálculo de predicados**”.
- Se desarrolló para probar la corrección de los programas.
- Los constructores de un lenguajes de programación se formalizan describiendo como su ejecución provoca un cambio de estado.



SEMÁNTICA

Semántica axiomática

- Un estado se describe con un predicado que describe los valores de las variables en ese estado
- Existe un **estado anterior** y un **estado posterior** a la ejecución del constructor.
- Cada sentencia se precede y se continúa con una expresión lógica que describe las restricciones y relaciones entre los datos.
 - Precondición
 - Poscondición

Ejemplo: a/b $\begin{array}{c} a \quad \underline{\quad b} \\ r \quad c \end{array}$

Precondición: $\{b \text{ distinto de cero}\}$

Sentencia: expresión que divide a por b

Postcondición: $\{a=b*c+r \quad \text{y} \quad r<b\}$



SEMÁNTICA

○ Semántica denotacional

- Se basa en la teoría de funciones recursivas
- Se diferencia de la axiomática por la forma que describe los estados, la axiomática lo describe a través de los predicados, la denotacional a través de funciones.
- Se define una correspondencia entre los constructores sintácticos y sus significados
- Ejemplo: números binarios

Producción: $\langle \text{Nbin} \rangle ::= 0 \mid 1 \mid \langle \text{Nbin} \rangle \ 0 \mid \langle \text{Nbin} \rangle \ 1$

$\text{FNbin}(0)=0$ $\text{FNbin}(\langle \text{Nbin} \rangle \ 0)= 2 * \text{FNbin}(\langle \text{Nbin} \rangle)$

$\text{FNbin}(1)=1$ $\text{FNbin}(\langle \text{Nbin} \rangle \ 1)= 2 * \text{FNbin}(\langle \text{Nbin} \rangle) + 1$

SEMÁNTICA

110

$\text{FNbin}(\langle \text{Nbin} \rangle 0)$

$2 * \text{FNbin}(\langle \text{Nbin} \rangle 1)$

$2 * [2 * \text{FNbin}(1) + 1]$

$2 * [2 * 1 + 1]$

$2 * [3]$

6

Producción: $\langle \text{Nbin} \rangle ::= 0 | 1 | \langle \text{Nbin} \rangle 0 \mid \langle \text{Nbin} \rangle 1$

$\text{FNbin}(0) = 0$ $\text{FNbin}(\langle \text{Nbin} \rangle 0) = 2 * \text{FNbin}(\langle \text{Nbin} \rangle)$

$\text{FNbin}(1) = 1$ $\text{FNbin}(\langle \text{Nbin} \rangle 1) = 2 * \text{FNbin}(\langle \text{Nbin} \rangle) + 1$

SEMÁNTICA

Semántica Operacional

- El significado de un programa se describe mediante otro lenguaje de bajo nivel implementado sobre una máquina abstracta
- Los cambios que se producen en el estado de la máquina cuando se ejecuta una sentencia del lenguaje de programación definen su significado
- Es un método informal
- Es el más utilizado en los libros de texto
- PL/1 fue el primero que la utilizó



SEMÁNTICA

Semántica Operacional

Ejemplo:

Lenguajes

```
for i := pri to ul do  
begin  
.....  
end
```

Máquina abstracta

```
i := pri  
lazo if i > ul goto sal  
.....  
i := i + 1  
goto lazo  
sal .....
```



PROCESAMIENTO DE UN LENGUAJE

TRADUCCIÓN

- Las computadoras ejecutan lenguajes de bajo nivel llamado “lenguaje de máquina”.
- Un poco de historia...
 - Programar en código de máquina
- Uso de código mnemotécnico (abreviatura con el propósito de la instrucción). “Lenguaje Ensamblador” y “Programa Ensamblador”

```
SUM #10, #11, #13  
SUM #13, #12, #13  
DIV #13, 3, #13  
FIN
```
- Aparición de los “Lenguajes de alto nivel”



PROCESAMIENTO DE UN LENGUAJE

TRADUCCIÓN

procesadores
80x86 de Intel

```
.data
msg:
.string "Hello, World!\n"
len:
.long . - msg
.text
.globl _start
_start:
push $len
push $msg
push $1
movl $0x4, %eax
call _syscall
addl $12, %esp
push $0
movl $0x1, %eax
call _syscall
_syscall:
int $0x80
ret
```

Procesadores de
la familia
Motorola 68000

```
start:
move.l #msg, -(a7)
move.w #9, -(a7)
trap #1
addq.l #6, a7
move.w #1, -(a7)
trap #1
addq.l #2, a7
clr -(a7)
trap #1
msg: dc.b "Hello,
World!", 10, 13, 0
```

‘Hola mundo!’
en dos familias de
procesadores



PROCESAMIENTO DE UN LENGUAJE TRADUCCIÓN

**Ejemplo de lenguaje máquina para el microprocesador 68000:
suma de dos enteros:**

Dirección	Código Binario	Código Ensamblador	Alto Nivel
\$1000	0011101000111000	MOVE.W \$1200,D5	Z=X+Y
\$1002	0001001000000000		
\$1004	1101101001111000	ADD.W \$1202,D5	
\$1006	0001001000000010		
\$1008	0011000111000101	MOVE.W \$D5,\$1204	
\$100A	0001001000000100		

Evolución hasta llegar a lenguajes de alto nivel

PROCESAMIENTO DE UN LENGUAJE

INTERPRETACIÓN Y COMPILACIÓN

- ¿Cómo los programas escritos en lenguajes de alto nivel pueden ser ejecutados sobre una computadora cuyo lenguaje es muy diferente y de muy bajo nivel?.
- Alternativas de traducción:
 - **Interpretación**
 - **Compilación**



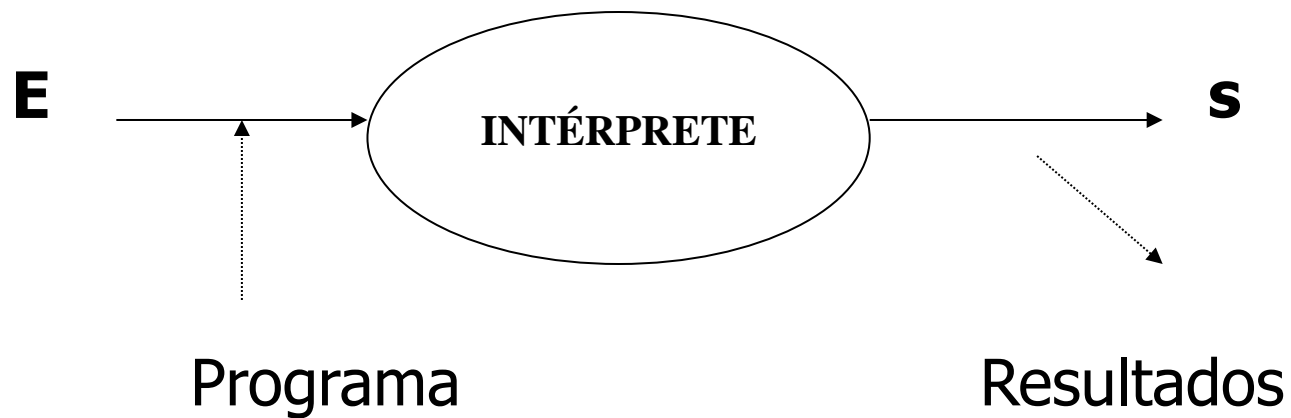
INTERPRETACIÓN

Intérprete:

- Lee,
- Analiza
- Decodifica y
- Ejecuta una a una las sentencias de un programa escrito en un lenguaje de programación.
- Ej: Lisp, Smalltalk, Basic, Python, etc.)
- Por cada posible acción hay un subprograma que ejecuta esa acción.
- La interpretación se realiza llamando a estos subprogramas en la secuencia adecuada.



INTERPRETACIÓN

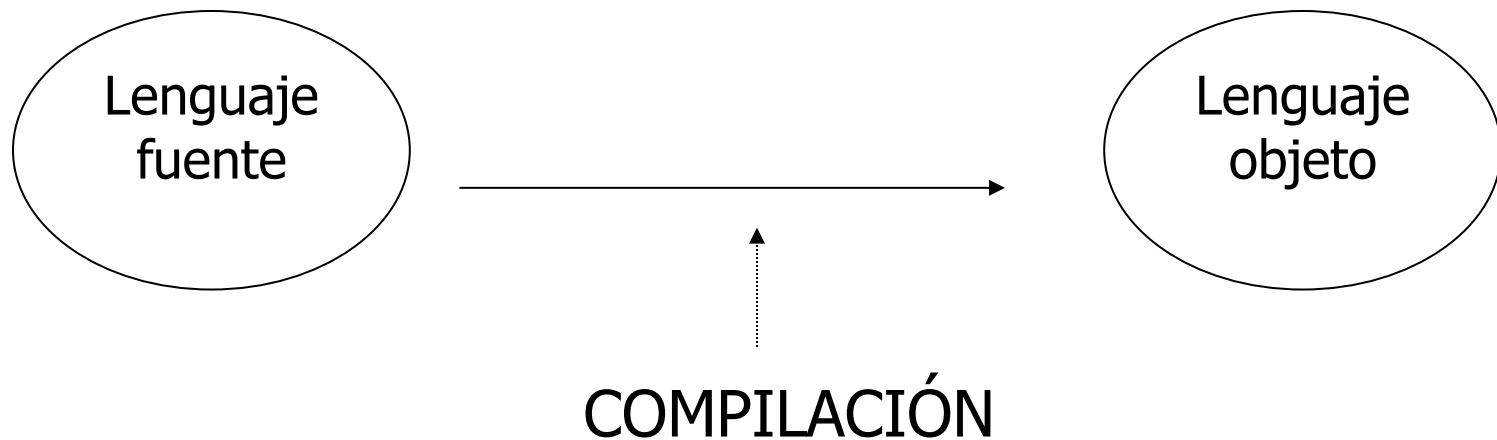


- Un intérprete ejecuta repetidamente la siguiente secuencia de acciones:
 - Obtiene la próxima sentencia
 - Determina la acción a ejecutar
 - Ejecuta la acción



COMPILACIÓN

Los programas escritos en un lenguaje de alto nivel se traducen a una versión en lenguaje de máquina antes de ser ejecutados.



TRADUCCIÓN

Comparación entre Traductor e Intérprete

- **Forma en cómo ejecuta:**
 - *Intérprete:*
 - Ejecuta el programa de entrada directamente
 - *Compilador:*
 - Produce un programa equivalente en lenguaje objeto
- **Forma en qué orden ejecuta:**
 - *Intérprete:*
 - Sigue el orden lógico de ejecución
 - *Compilador:*
 - Sigue el orden físico de las sentencias



TRADUCCIÓN

○ **Tiempo de ejecución:**

- ***Intérprete:***

- Por cada sentencia se realiza el proceso de decodificación para determinar las operaciones a ejecutar y sus operandos.
- Si la sentencia está en un proceso iterativo, se realizará la tarea tantas veces como sea requerido
- La velocidad de proceso se puede ver afectada

- ***Compilador:***

- No repetir lazos, se decodifica una sola vez

○ **Eficiencia:**

- ***Intérprete:***

- Más lento en ejecución

- ***Compilador:***

- Más rápido desde el punto de vista del hard



TRADUCCIÓN

- **Espacio ocupado:**

- *Intérprete:*

- Ocupa menos espacio, cada sentencia se deja en la forma original

- *Compilador:*

- Una sentencia puede ocupar cientos de sentencias de máquina

- **Detección de errores:**

- *Intérprete:*

- Las sentencias del código fuente pueden ser relacionadas directamente con la que se esta ejecutando.

- *Compilador:*

- Cualquier referencia al código fuente se pierde en el código objeto

TRADUCCIÓN

Combinación de ambas técnicas:

- Los compiladores y los interpretes se diferencian en la forma que ellos reportan los errores de ejecución.
- Algunos ambientes de programación contienen las dos versiones **interpretación y compilación**.
 - Utilizan el *intérprete* en la etapa de desarrollo, facilitando el diagnóstico de errores.
 - Luego que el programa ha sido validado se *compila* para generar código mas eficiente.



TRADUCCIÓN

Combinación de ambas técnicas

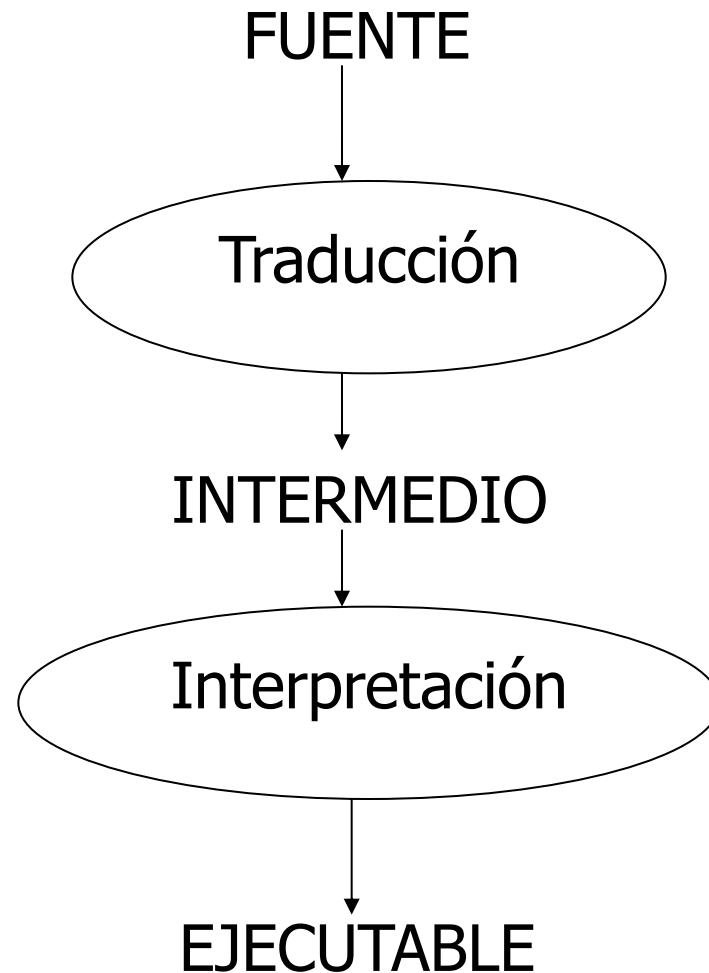
Otro forma de combinarlos:

- Traducción a un código intermedio que luego se interpretará.
- Sirve para generar **código portable**, es decir, código fácil de transferir a diferentes máquinas.
- **Ejemplos:** Java, genera un código intermedio llamado “bytecodes”, que luego es interpretado por la máquina cliente.



TRADUCCIÓN

- **Combinación de ambas técnicas:**



COMPILADORES

- Al compilar los programas la ejecución de los mismos es más rápida. Ej. de programas que se compilan: C, Ada, Pascal, etc.
- Los compiladores pueden ejecutarse en un solo paso o en dos pasos.
- En ambos casos cumplen con varias etapas, las principales son

- **Análisis**

- Análisis léxico (Scanner)
- Análisis sintáctico (Parser)
- Análisis semántico (Semántica estática)

- **Síntesis**

- Optimización del código
- Generación del código


← ***Generación de
código intermedio***



COMPILADORES

- **Análisis del programa fuente**

- **Análisis léxico (Scanner):**

- Es el que lleva mas tiempo
 - Hace el análisis a nivel de palabra
 - Divide el programa en sus elementos constitutivos: identificadores, delimitadores, símbolos especiales, números, palabras clave, delimitadores, comentarios, etc.
 - Analiza el tipo de cada token
 - Filtra comentarios y separadores como: espacios en blanco, tabulaciones, etc.
 - Genera errores si la entrada no coincide con ninguna categoría léxica
 - Convierte a representación interna los números en punto fijo o punto flotante
 - Poner los identificadores en la tabla de símbolos
 - Reemplaza cada símbolo por su entrada en la tabla
 - El resultado de este paso será el descubrimiento de los items léxicos o tokens.
- 

COMPILADORES

- **Scanner**

x:=a+b*c;
y:=3+b*c;

Analizador
Léxico

TOKENS

(id,"x")	(op,":=")	(id,"a")
(op,"+")	(id,"b")	(op,"*")
(id,"c")	(punt,";")	
(id,"y")	(op,":=")	(num,"3")
(op,"+")	(id,"b")	(op,"*")
(id,"c")	(punt,";")	

COMPILADORES

- **Análisis sintáctico (Parser):**

- El análisis se realiza a nivel de sentencia.
- Se identifican las estructuras; sentencias, declaraciones, expresiones, etc. ayudándose con los tokens.
- El analizador sintáctico se alterna con el análisis semántico. Usualmente se utilizan técnicas basadas en gramáticas formales.
- Aplica una gramática para construir el árbol sintáctico del programa.



COMPILADORES

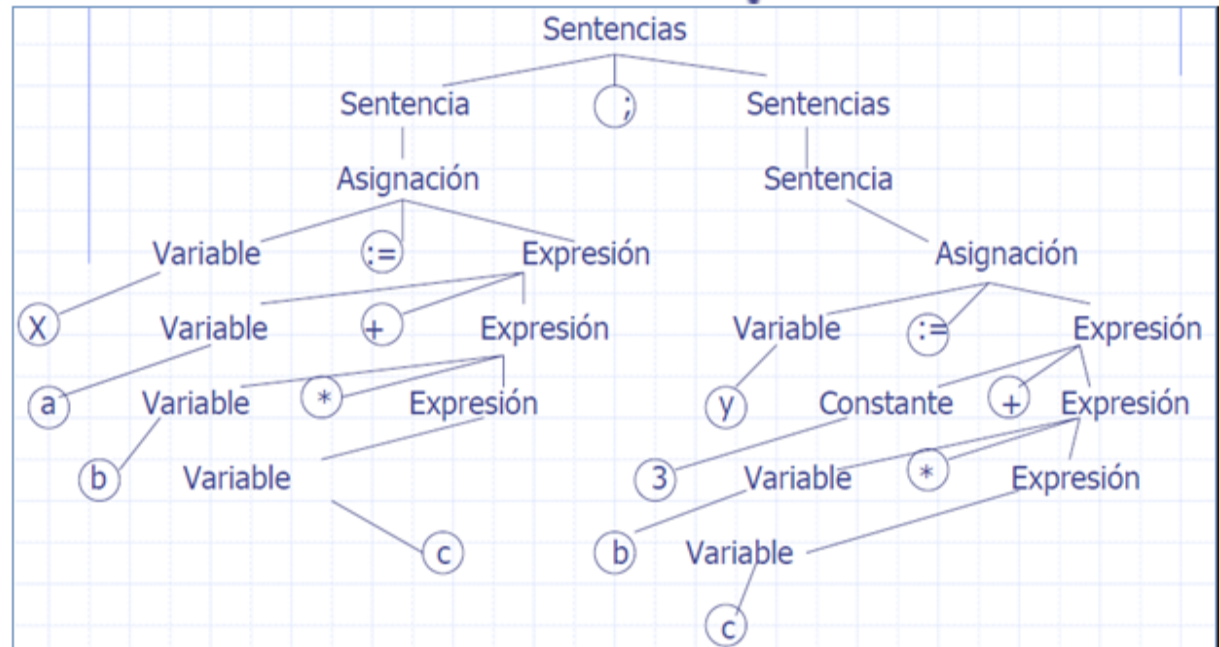
- Parser**

x := a + b * c;
y := 3 + b * c;




TOKENS		
(id, "x")	(op, ":=")	(id, "a")
(op, "+")	(id, "b")	(op, "*")
(id, "c")	(punct, ";")	
(id, "y")	(op, ":=")	(num, "3")
(op, "+")	(id, "b")	(op, "*")
(id, "c")	(punct, ";")	

Analizador
Sintáctico



COMPILADORES

Análisis semántica (semántica estática):

- Es la fase medular
 - Es la mas importante
 - Las estructuras sintácticas reconocidas por el analizador sintáctico son procesadas y la estructura del código ejecutable toma forma.
 - Se realiza la comprobación de tipos
 - Se agrega la información implícita (variables no declaradas)
 - Se agrega a la tabla de símbolos los descriptores de tipos, etc. a la vez que se hacen consultas para realizar comprobaciones.
 - Se hacen las comprobaciones de nombres. Ej: toda variable debe estar declarada.
 - Es el nexo entre el análisis y la síntesis
- 

COMPILADORES

Generación de código intermedio:

- Características de esta representación
 - Debe ser fácil de producir
 - Debe ser fácil de traducir al programa objeto

Ejemplo: Un formato de código intermedio es el **código de tres direcciones**.

Forma: $A := B \text{ op } C$, donde A, B, C son operandos y op es un operador binario

Se permiten condicionales simples y saltos.

while (a > 0) and (b < (a * 4 - 5)) do a := b * a - 10;

L1: if (a > 0) goto L2

goto L3

L2: t1 := a * 4

t2 := t1 - 5

if (b < t2) goto L4

goto L3

L4: t1 := b * a

t2 := t1 - 10

a := t2

goto L1

L3:



COMPILADORES

- **Síntesis:**

- En esta etapa se construye el programa ejecutable.
- Se genera el código necesario y se optimiza el programa generado.
- Si hay traducción separada de módulos, es en esta etapa cuando se linkedita.
- Se realiza el proceso de optimización. Optativo



COMPILADORES

○ Optimización (ejemplo):

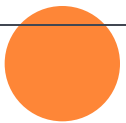
Posibles optimizaciones locales:

- Cuando hay dos saltos seguidos se puede quedar uno solo

El ejemplo anterior quedaría así:

L1: if (a<=0) goto L3	t1:=b*a
t1:=a*4	t2:=t1-10
t2:=t1-5	a:=t2
if (b >= t2) goto L3	goto L1
	L3:

```
L1: if (a>0) goto L2
      goto L3
L2: t1:=a*4
      t2:=t1-5
      if (b < t2) goto L4
      goto L3
L4: t1:=b*a
      t2:=t1-10
      a:=t2
      goto L1
L3: .....
```



COMPILADORES

