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>+<exp>|<term> - <exp>

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

<elem>::=<iden>|<núm>

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

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

<letra>::=a|b|c|…

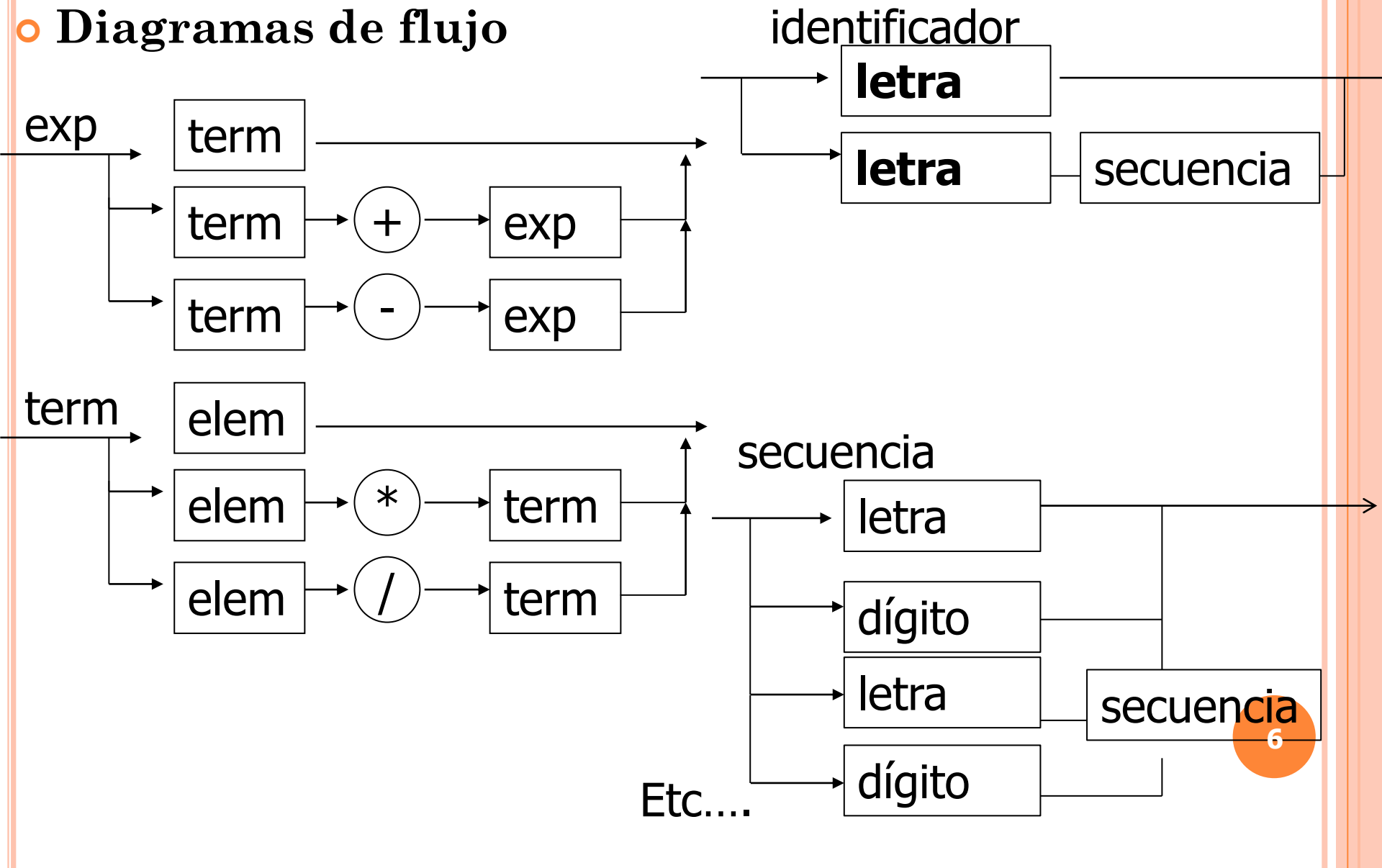
<núm>::=<digito>|<digito><núm>

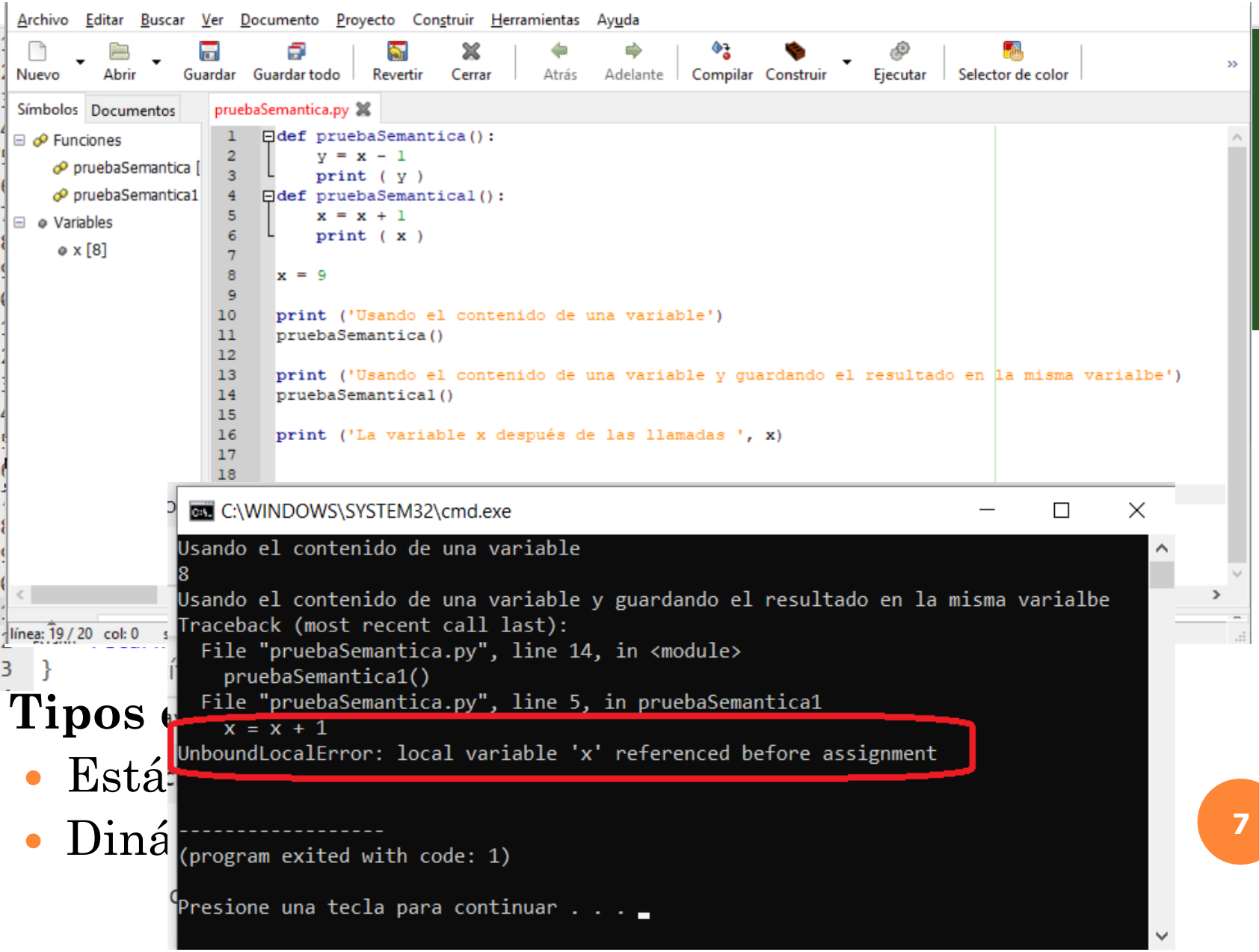
<digito>::=0|1|2|….

}

Repaso Clase Anterior

- **Diagramas de flujo**





- **Tipos e**
  - Está
  - Diná

1

main.c

```c
#include <stdio.h>

int main()
{
    int a, resul;
    char cadena;

    cadena = 'h';
    resul = a + x;
    printf(resul);
    return 0;
}
```

2

main.c

```c
#include <stdio.h>

int main()
{
    int a, resul;
    char cadena;

    cadena = 'h';
    resul = a + cadena;
    printf(resul);
    return 0;
}
```

main.c

```c
#include <stdio.h>

int main()
{
    int a;
    int a, b;

    b = a * 2
    printf(b);
    return 0;
}
```

3

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 (con la sintaxis)**
- 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 les **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

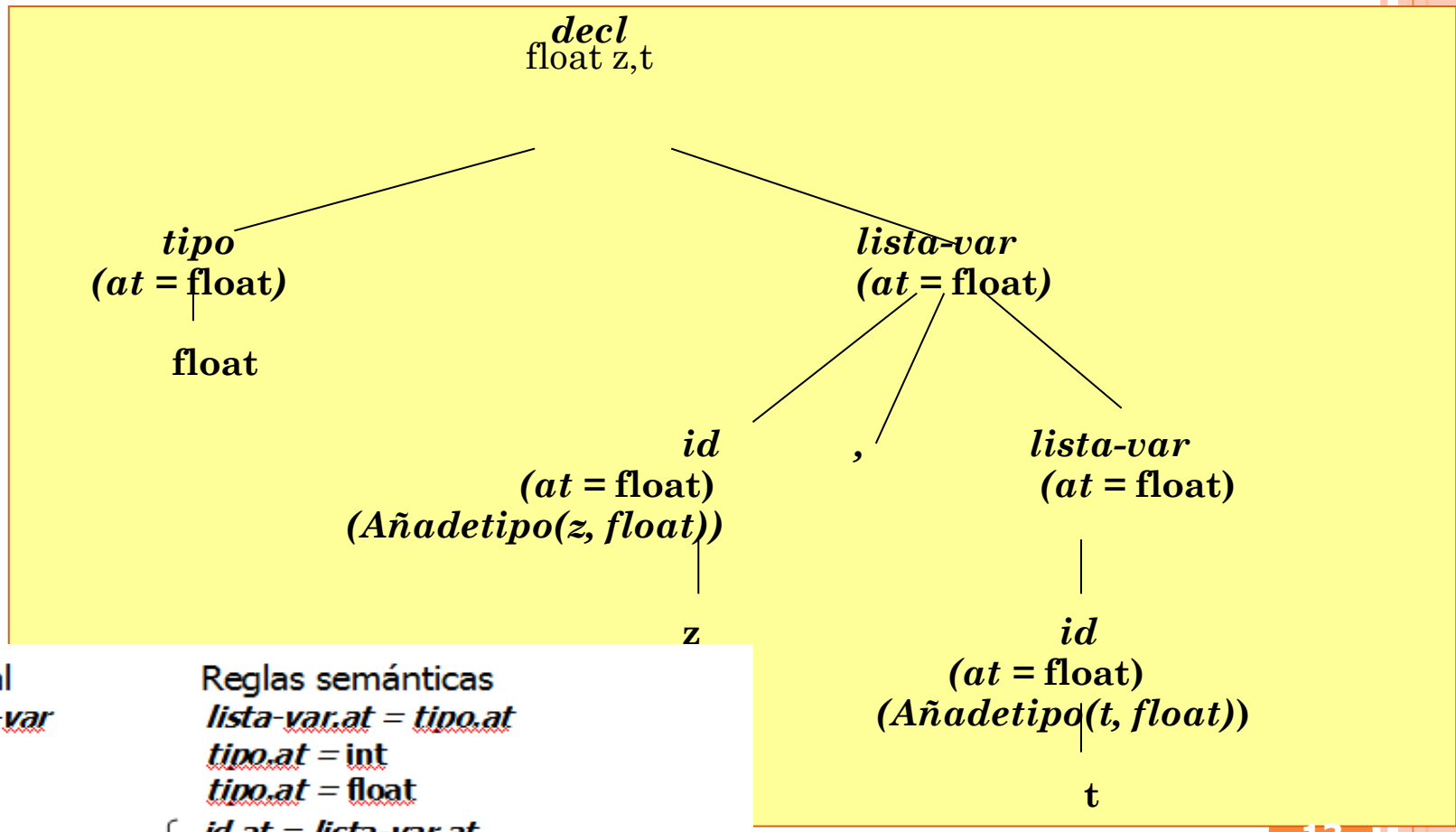
SEMÁNTICA

Semántica estática - Gramática de atributos

- Ej. Gramática simple para una declaración de variables sólo tipo *int* y *float* en el **lenguaje C**. **Atributo at**

Regla gramatical	Reglas semánticas
decl → tipo lista-var	***lista-var.at = tipo.at***
tipo→ int	***tipo.at = int***
tipo→ float	***tipo.at = float***
lista-var→ id	***id.at = lista-var.at*** ***Añadetipo(id.entrada, lista-var.at)***
$lista\text{-}var_1 \rightarrow id, lista\text{-}var_2$	***id.at = lista-var.at*** $Añadetipo(id.entrada, lista\text{-}var_1.at)$ $lista\text{-}var_2.at = lista\text{-}var.at$

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



Regla gramatical	Reglas semánticas
decl → *tipo lista-var*	*lista-var.at* = *tipo.at*
tipo → int	*tipo.at* = int
tipo → float	*tipo.at* = float
lista-var → *id*	*id.at* = *lista-var.at* *Añadetipo(id.entrada, lista-var.at)*
$lista\text{-}var_1 \rightarrow id, lista\text{-}var_2$	*id.at* = *lista-var.at* *Añadetipo(id.entrada, lista-var*$_1$*.at)* $lista\text{-}var_2.at = lista\text{-}var.at$

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 **sólo** se **pueden ejecutar** si son **correctos** para la **sintáxis** y para la **semántica estática**.

Analizar en C
int notas[10];

Semántica

¿Cómo se describe la semántica?

- No es fácil escribirla
- No existen herramientas estándar (fáciles y claras) como en el caso de la sintáxis (diagramas sintácticos y BNF)
- Hay diferentes soluciones formales complejas:
  - 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

  a | b
  r | c

  Precondición: {b distinto de cero}

  Sentencia: expresión que divide a por b

  Postcondición: {a=b*c+r y r<b}

Semántica

110 es un 6

FNbin(<Nbin>**0**)

2* FNbin(<Nbin>**1**)

2*[2*FNbin(**1**)+1]

2*[2*1+1]

2*[3]

6 **resultado de las funciones**

Producción definir binarios: <Nbin>::=0|1|<Nbin> 0 | <Nbin> 1

FNbin(0)=0 FNbin(<Nbin> 0)= 2 * FNbin(<Nbin>)

FNbin(1)=1 FNbin(<Nbin> 1)= 2 * FNbin(<Nbin>) + 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: en Pascal

Lenguajes

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

Máquina abstracta

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

Procesamiento de un lenguaje Traducción

Procesadores de

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		

call _syscall
_syscall:
int $0x80
ret

FIN

- A alto nivel"

Procesamiento de un lenguaje Traducción: 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 y Compilación**

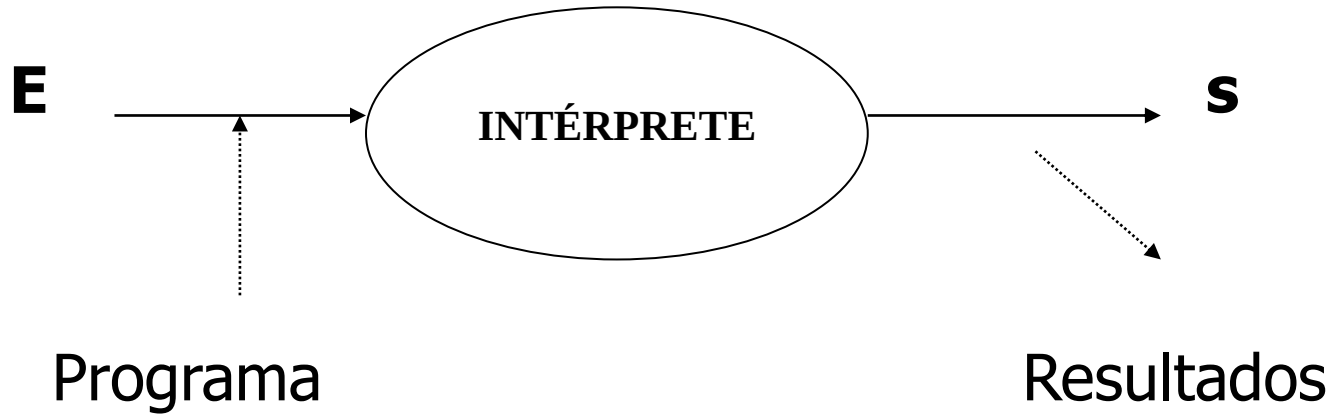
Interpretación

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, Ruby, PHP, Perl..)
- Por **cada** posible **acción** hay **un subprograma** en **lenguaje de máquina** que ejecuta esa acción.
- La interpretación se realiza **llamando** a estos **subprogramas** en la **secuencia adecuada**.

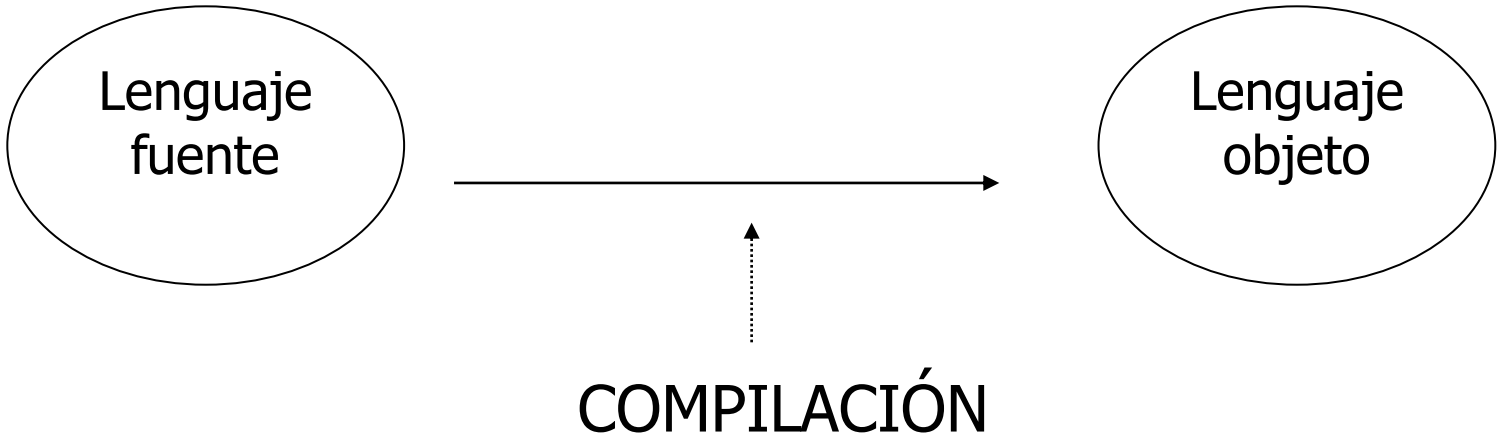
Interpretación



- 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 Compilador e Intérprete

- **Forma en cómo ejecuta:**
  - ***Intérprete:***
    - Ejecuta el programa de entrada directamente **(Durante)**
  - ***Compilador:***
    - Produce un programa equivalente en lenguaje objeto **(Antes)**
- **Forma en qué orden ejecuta:**
  - ***Intérprete:***
    - Sigue el orden lógico de ejecución **(por donde pasa)**
  - ***Compilador:***
    - Sigue el orden físico de las sentencias **(todo)**

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:***
    - Genera código de máquina para cada sentencia. No repite lazos, se decodifica una sola vez.

- **Eficiencia:**
  - ***Intérprete:***
    - Más lento en ejecución
  - ***Compilador:***
    - Más rápido ejecutar desde el punto de vista del hard, pero tarda más en compilar

Traducción

- **Espacio ocupado:**
  - ***Intérprete:***
    - Ocupa menos espacio de memoria.
    - Cada sentencia se deja en la forma original y las instrucciones necesarias para ejecutarlas se almacenan en los subprogramas del interprete en memoria
  - ***Compilador:***
    - Una sentencia puede ocupar decenas o centenas 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 técnicas:

- *Primero interpreto y luego compilo*
- *Primero compilo y luego interpreto*

La interpretación pura y la compilación pura son dos extremos, en la práctica muchos lenguajes combinan ambas técnicas.

Traducción

Combinación de ambas técnicas:

Primero interpreto y luego compilo

- **Los compiladores y los interpretes se diferencian en la forma que ellos reportan los errores de ejecución. (se basa en esto)**
- 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

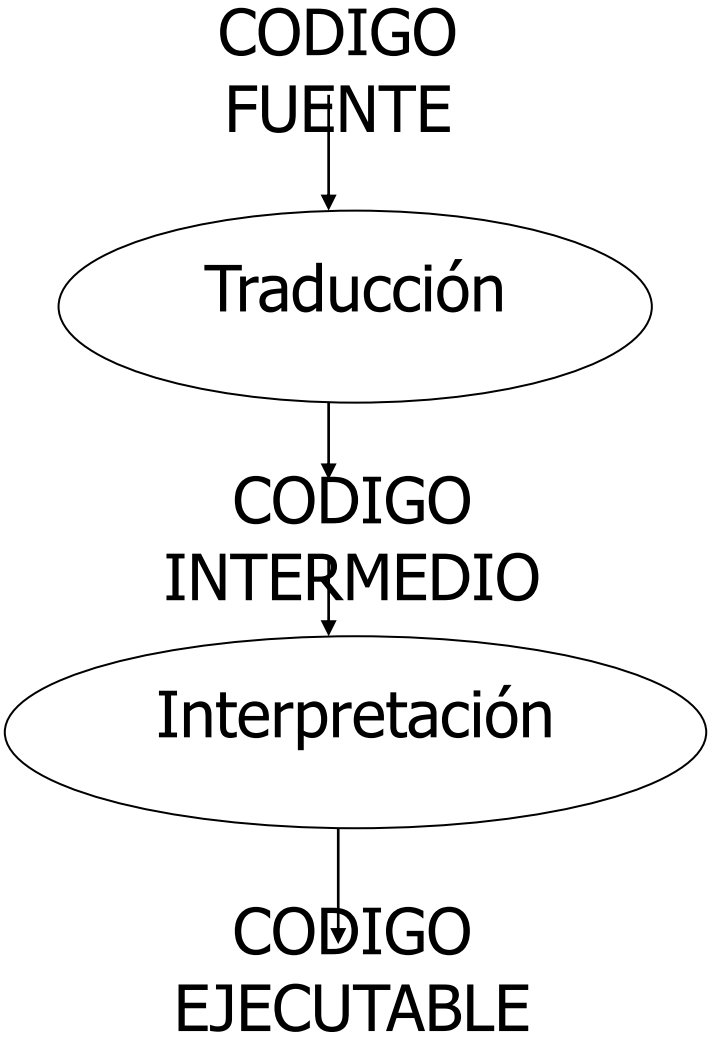
Primero compilo y luego interpreto

Otra 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 estas técnicas:**



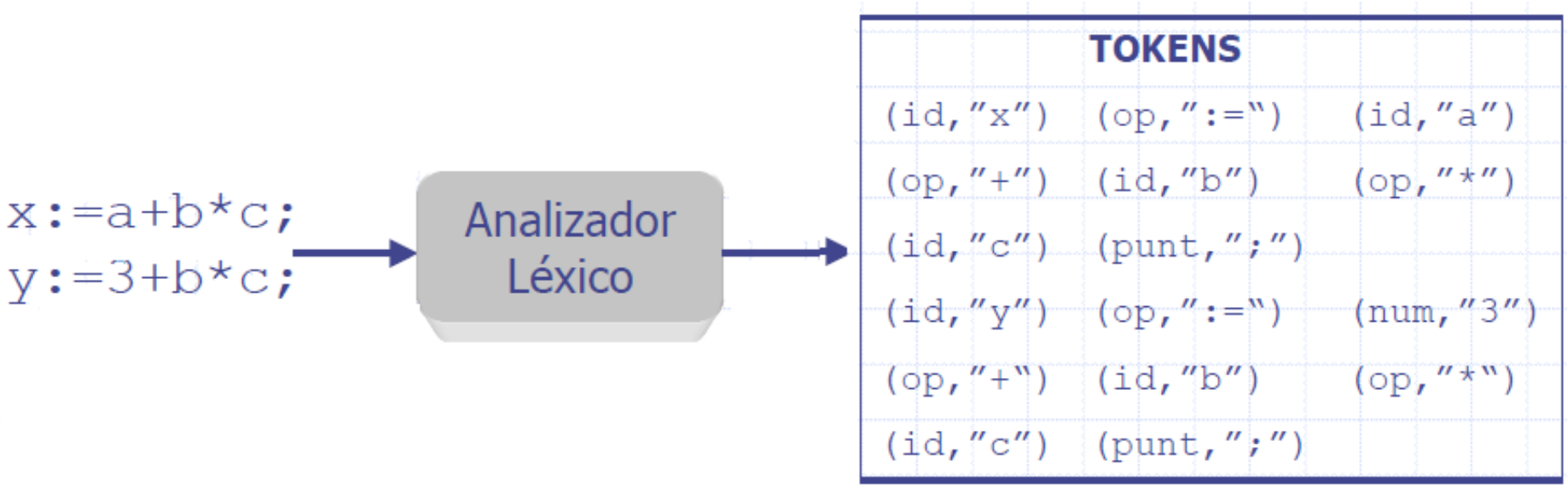
Compiladores – Cómo funcionan

- **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 **ejecutare** en **un** sólo **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

- **Etapa de Análisis del programa fuente**
  - **Análisis léxico (Scanner):**
    - Es el que lleva mas tiempo
    - Hace el análisis a nivel de palabra (**LEXEMA**)
    - Divide el programa es sus elementos constitutivos: identificadores, delimitadores, símbolos especiales, números, palabras clave, delimitadores, comentarios, etc.

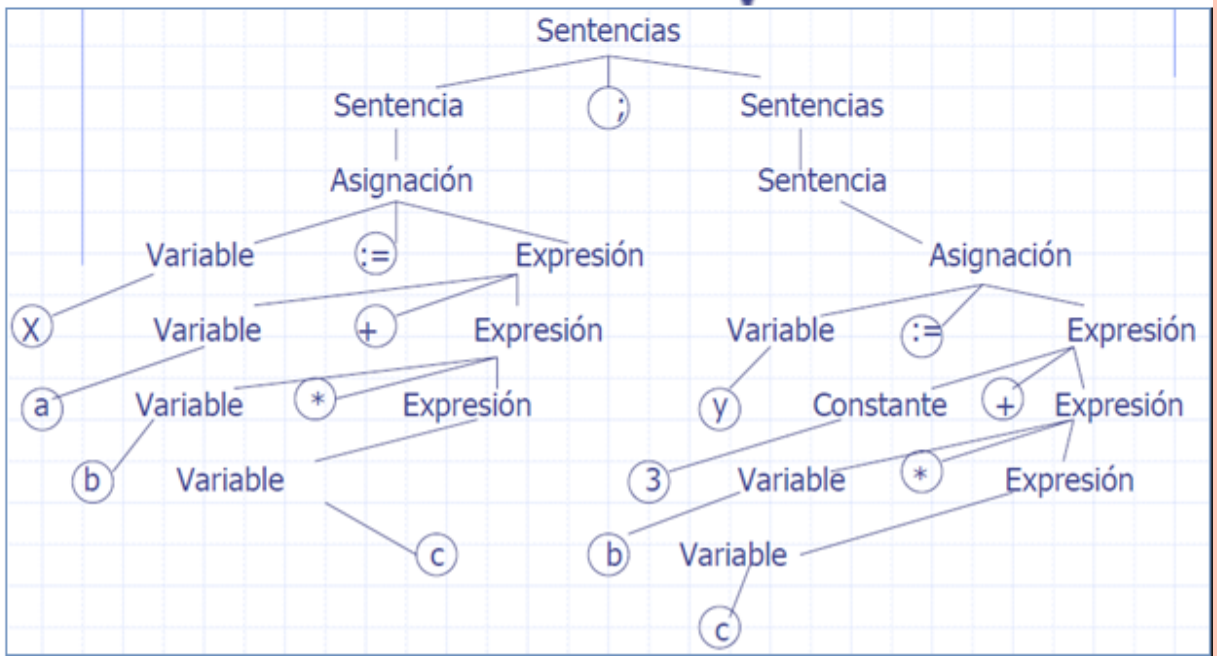


COMPILADORES

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

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,";")	

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:

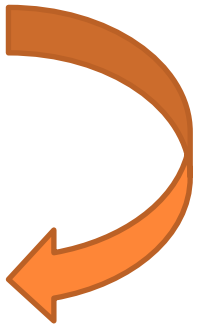
- Características de esta representación
  - Debe ser **fácil** de **producir**
  - Debe ser **fácil** de **traducir** al programa objeto

Ejemplo: Un formato de código intermedio es el **código de tres direcciones**.
Forma: *A*:= *B 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

- **Etapa de Síntesis:**
  - **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 2 saltos seguidos se puede quedar 1 solo

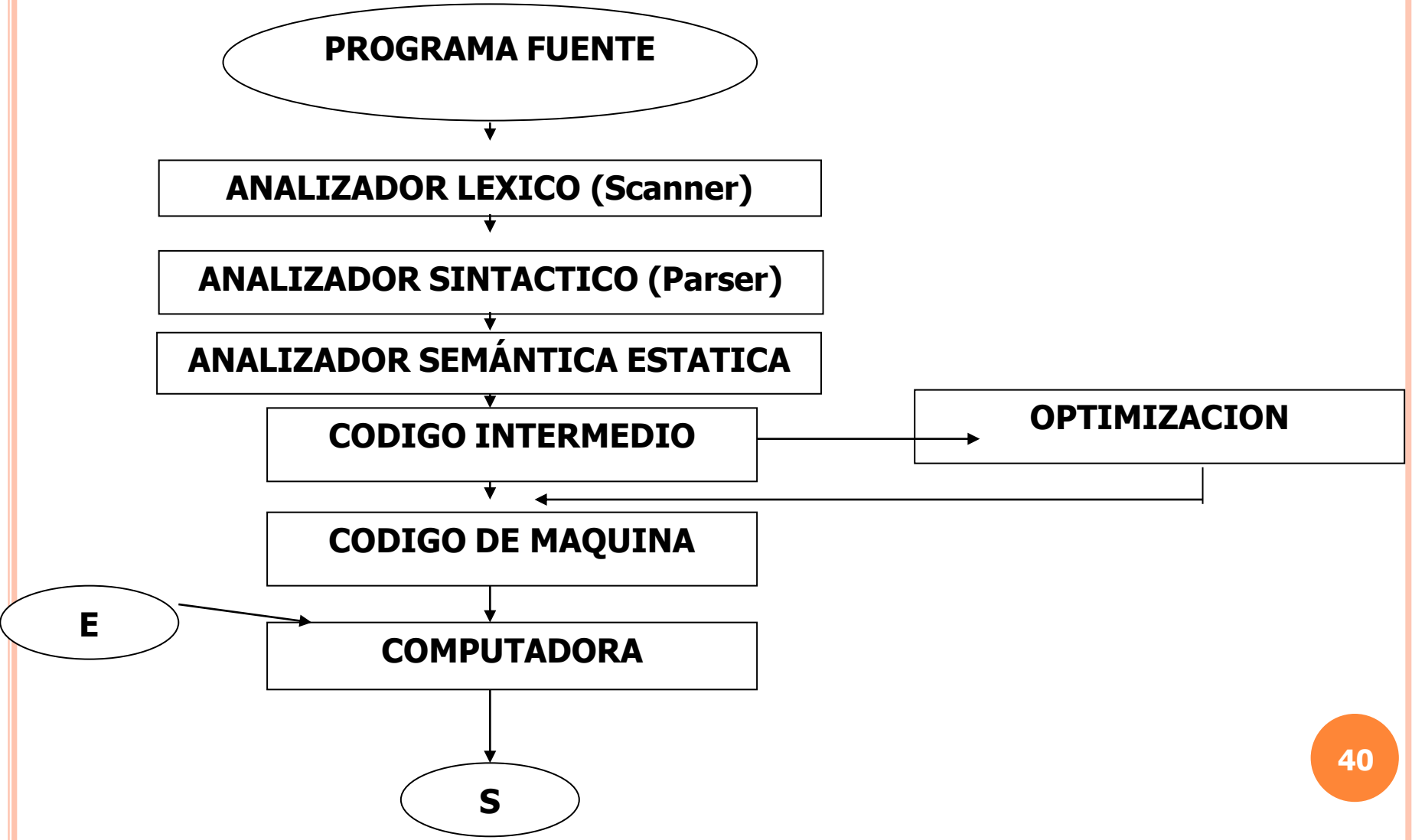
Ejemplo anterior

```
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: .....
```

Luego optimización

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

COMPILADORES



PRÓXIMA CLASE

SEMÁNTICA OPERACIONAL.

- Ligadura. Descriptores. Momentos de ligadura. Estabilidad.
- Variables. Arquitectura Von Newman. Atributos. Momentos y estabilidad. Nombre: características. Alcance: visibilidad, reglas. Tipo: definición, clasificación. L-valor: tiempo de vida, alocación. R-valor: constantes, inicialización. Alias