



Universidad  
de Huelva

# Tema 1

## Introducción

1.1 Conceptos

1.2 Un poco de historia

1.3 Estructura de un compilador

1.4 Teoría de lenguajes formales

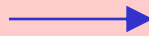
### 1.1 Conceptos

1.2 Un poco de historia

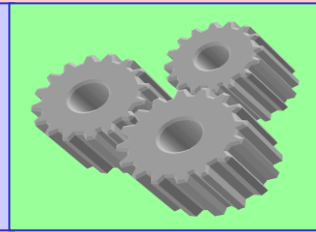
1.3 Estructura de un compilador

1.4 Teoría de lenguajes formales

Lenguaje fuente



Procesador  
de lenguaje



- Procesador de lenguaje: aplicación que toma como entrada un conjunto de archivos descritos en un cierto lenguaje formal (lenguaje fuente)
- Traductor: procesador de lenguaje cuya función consiste en re-escribir los archivos de entrada en otro lenguaje formal (lenguaje objeto)



- Ensamblador: traductor de lenguaje ensamblador a código máquina
- Compilador: traductor de lenguaje de alto nivel a ensamblador o a código máquina

- Intérprete: Analiza el código fuente y lo ejecuta sin generar un código objeto.
- Emulador: Intérprete en el que el código fuente es el código máquina de una plataforma diferente.
- Compiladores vs intérpretes:
  - Los programas compilados son mucho más rápidos
  - Los compiladores consumen muchos recursos
  - Los intérpretes utilizan código independiente de la plataforma
  - Los intérpretes pueden ejecutar código generado en tiempo de ejecución.

- **Preprocesador**: pequeño compilador dedicado a expandir las macros e incluir ficheros. (*include, define ...*)
- **Enlazador (linker)**: programa que enlaza los diferentes módulos compilados para generar el ejecutable.
- **Compilador cruzado**: compilador que genera código para una máquina diferente a la utilizada para compilar.
- **Autocompilador**: compilador escrito en el mismo lenguaje que va a compilar.
- **Metacompilador**: herramienta de ayuda al diseño de compiladores. *JAVA CC*
- **Descompilador**: programa que produce una representación de alto nivel a partir del código máquina. (*Ing. Inverse*)

1.1 Conceptos

**1.2 Un poco de historia**

1.3 Estructura de un compilador

1.4 Teoría de lenguajes formales



- 1940's:
  - Primeros ordenadores.
  - Programación directa en código máquina.
  - Nace el ensamblador (traducido manualmente)
  - Se automatiza la etapa de ensamblado
- 1950's
  - (1950) John Backus dirige en IBM un proyecto de lenguaje algebraico
  - (1954-1958) Nace el FORTRAN (FORmulae TRANslator)
  - Primer compilador de FORTRAN para IBM modelo 704

- 1950's (mediados):
  - Noam Chomsky publica sus estudios sobre la estructura de los lenguajes y las gramáticas formales
- 1950's (finales):
  - F.L. Bauer (Univ. Munich) dirige un proyecto de lenguaje formal
  - Se crea un comité conjunto con la Association for Computing Machinery en el que participa Backus
  - Se define IAL (International Algebraic Language)
  - Posteriormente se denomina ALGOL 58 (ALGOrithmic Language)

- Características de ALGOL 58
  - Definición de la notación BNF (Backus-Naur Form)
  - Declaración de tipos de variables
  - Estructura modular
  - Funciones recursivas
  - Paso de parámetros por valor o por nombre
  - Precursor de Pascal, Ada o Modula

- 1960's: Primeras técnicas de desarrollo de compiladores
  - 1958: Strong y otros proponen dividir el compilador en dos fases: front-end y back-end y promueven la creación de un lenguaje intermedio universal (UNCOL – UNiversal Computer Oriented Language).
  - 1959: Rabin y Scott proponen utilizar autómatas deterministas para el análisis léxico
  - 1961: Primer analizador sintáctico descendente recursivo
  - Durante toda la década se estudian intensamente las gramáticas LL.

- 1970's:
  - Se estudian las gramáticas LR
  - Se definen los métodos de análisis ascendente SLR y LALR
  - Se crean numerosas herramientas de ayuda al diseño de compiladores
  - A mediados de la década aparecen las herramientas lex y yacc que se convierten en estándares de facto al distribuirse con UNIX.
  - La Free Software Foundation distribuye estas herramientas bajo los nombres de flex y bison

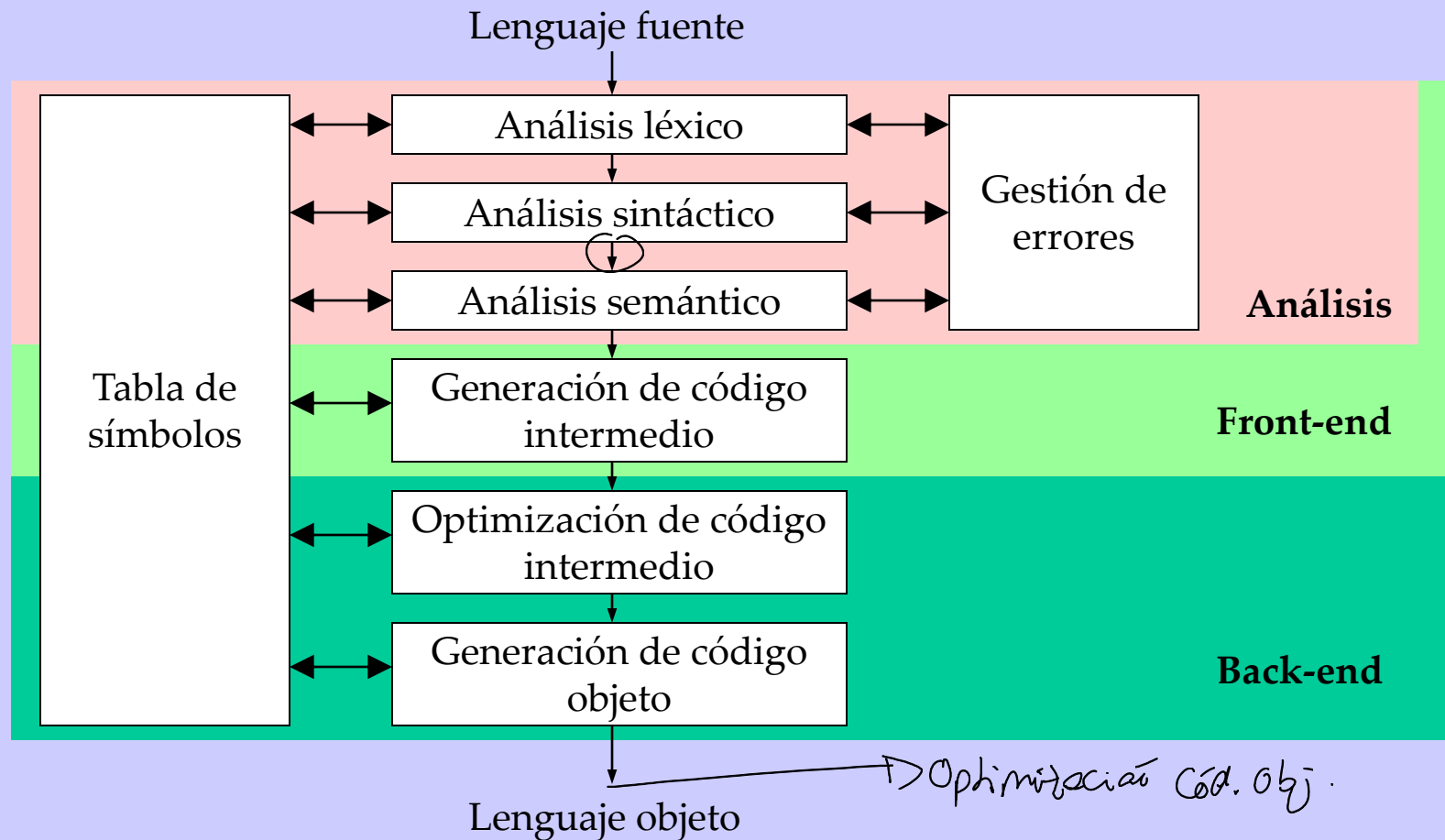
- Generación de código:
  - Inicialmente la gestión de la memoria era estática
  - Cuando aparecen las funciones recursivas se desarrolla la pila como forma de gestionar la memoria
  - La gestión de memoria dinámica conduce a la definición del montículo (heap) como zona de gestión de esta memoria
  - Surgen nuevos problemas como las referencias perdidas o la recogida de basura
- Últimos avances:
  - Optimización de código
  - Máquinas virtuales y auge de los lenguajes interpretados

1.1 Conceptos

1.2 Un poco de historia

**1.3 Estructura de un compilador**

1.4 Teoría de lenguajes formales





- Objetivo:
  - Leer caracteres e identificar componentes léxicos (*tokens*)
  - Filtrar comentarios
  - Detectar errores léxicos
- Ejemplo:

m e d i a = ( a + b ) / 2 ; / \* i n s t r u c c i o n \* /

↓ trocea la cadena y produce  
los componentes léxicos (*tokens*)

id<sub>media</sub> asign lpar id<sub>a</sub> suma id<sub>b</sub> rpar div num<sub>2</sub> pyc

- Objetivo:
  - Verificar estructuras gramaticales
  - Detectar errores de sintaxis
- Especificación:
  - Gramáticas libres de contexto
- Ejemplo:

$\langle \text{Asign} \rangle \rightarrow \text{id asign} \langle \text{Expr} \rangle \text{ pyc}$

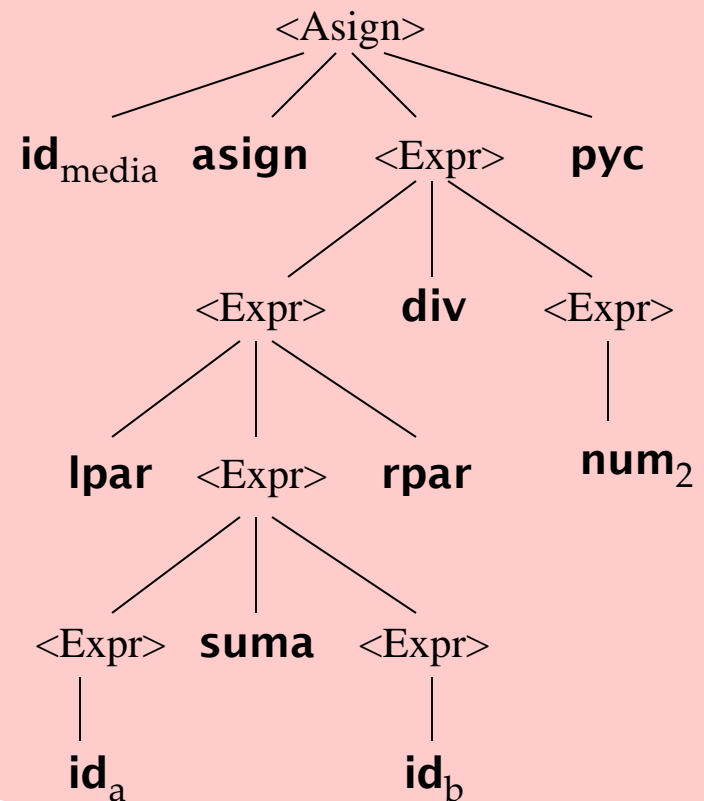
$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \text{ suma} \langle \text{Expr} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \text{ div} \langle \text{Expr} \rangle$

$\langle \text{Expr} \rangle \rightarrow \text{lpar} \langle \text{Expr} \rangle \text{ rpar}$

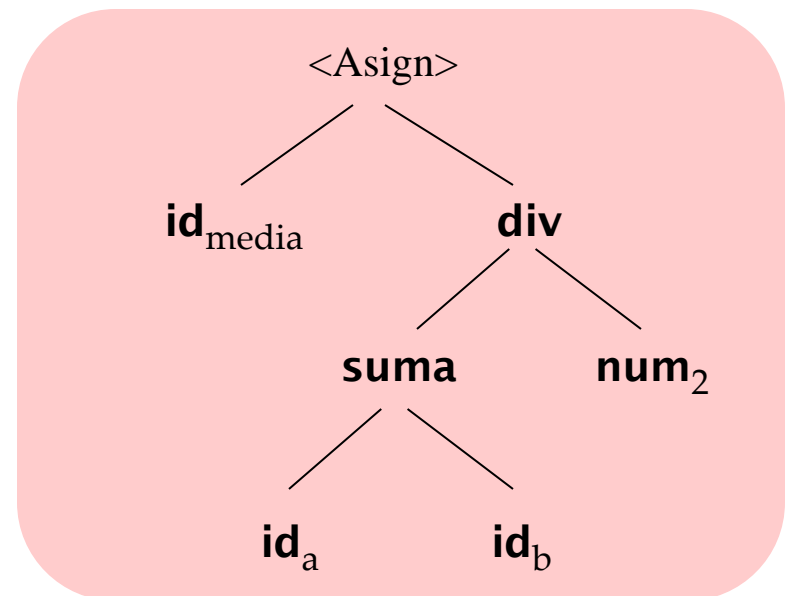
$\langle \text{Expr} \rangle \rightarrow \text{id}$

$\langle \text{Expr} \rangle \rightarrow \text{num}$



IMPORTANTE

- Objetivo:
  - Verificar restricciones semánticas (predefinición de variables, consistencia de tipos, llamadas a funciones)
  - Generar un Árbol de Sintaxis Abstracta
- Especificación:
  - Gramáticas atribuidas
- Ejemplo:



- Objetivo:
  - Generar una descripción en código de bajo nivel independiente de la plataforma
- Especificación:
  - Código de tres direcciones (destino, operando1, operando2, operación)
- Ejemplo:

$T_i \rightarrow$  Variables Temporales

```
T1 := a
T2 := b
T3 := T1 + T2
T4 := 2
T5 := T3 / T4
media := T5
```

- Objetivo:
  - Reducir el número de operaciones para mejorar el rendimiento
- Técnicas:
  - Eliminar saltos consecutivos
  - Reutilizar valores intermedios
  - Extraer instrucciones de los bucles
  - Expandir los bucles
  - Etc.

- Objetivo:
  - Obtener la representación en ensamblador para la máquina de destino
  - Optimizar el código
- Técnicas.
  - Almacenar las variables en registros
  - Utilizar instrucciones específicas
  - Etc.

- Objetivo:
  - Almacenar toda la información de los diferentes símbolos (variables, constantes, tipos, funciones)
- Características:
  - Cada etapa requiere una información diferente
  - Comportamiento dinámico (variables definidas en módulos)
  - Optimizar el tiempo de acceso (tablas hash)

- Objetivo:
  - Localizar errores
  - Informar de los detalles del error
  - Recuperación del error (tokens de sincronismo)



1.1 Conceptos

1.2 Un poco de historia

1.3 Estructura de un compilador

**1.4 Teoría de lenguajes formales**

- Alfabeto ( $\Sigma$ ): conjunto finito de símbolos
- Cadena: secuencia finita de símbolos
- Cadena vacía ( $\lambda$ ): cadena de longitud cero
- $\Sigma^k$ : Conjunto de cadenas de longitud k
- Clausura del alfabeto ( $\Sigma^*$ ): Conjunto de todas las cadenas

} ASCII  
tokens  
...

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

- Lenguaje formal: Subconjunto de  $\Sigma^*$

- Gramáticas formales: *≠ Lenguaje formal*
  - Forma de describir lenguajes formales
  - Cuatro elementos ( $N, \Sigma, P, \langle S \rangle$ ):
  - $N$ : Alfabeto de símbolos no terminales
  - $\Sigma$ : Alfabeto de símbolos terminales
  - $P$ : Conjunto de producciones o reglas  $\rightarrow$  Debe haber el menos 1 símbolo No terminal en la izq.
 
$$P \subset (N \cup \Sigma)^* N (N \cup \Sigma)^* \rightarrow (N \cup \Sigma)^*$$
  - $\langle S \rangle$ : Símbolo inicial (no terminal) (axioma)

- **Derivación directa:**  $\eta$  deriva directamente en  $\gamma$  ( $\eta \Rightarrow \gamma$ ) si

- $\eta = \omega_1 \alpha \omega_2$

- $\gamma = \omega_1 \beta \omega_2$

- $\alpha \rightarrow \beta \in P$

- **Derivación:**  $\eta$  deriva en  $\gamma$  ( $\eta \xRightarrow{*} \gamma$ ) si

- existen  $\xi_1, \dots, \xi_n$  tales que  $\eta$  deriva directamente en  $\xi_1$ ,  $\xi_i$  deriva directamente en  $\xi_{i+1}$  y  $\xi_n$  deriva directamente en  $\gamma$

- **Forma sentencial:** cadena  $\alpha \in (N \cup \Sigma)^*$  tal que  $\underline{\langle S \rangle} \xRightarrow{*} \alpha$  */ a partir del  $\langle S \rangle$*
- **Sentencia:** forma sentencial perteneciente a  $\Sigma^*$

- Lenguaje generado por la gramática G:

- $L(G) = \{ x \in \Sigma^* \mid \langle S \rangle \xRightarrow{*} x \}$

- Ejemplo:

- $G = (\{ \langle S \rangle, \langle E \rangle \}, \{ \text{id}, +, * \}, P, \langle S \rangle )$ , con las siguientes reglas:

- $\langle S \rangle \rightarrow \langle E \rangle$
    - $\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle$
    - $\langle E \rangle \rightarrow \langle E \rangle * \langle E \rangle$
    - $\langle E \rangle \rightarrow \text{id}$

- Ejemplo de derivación:

- $\langle S \rangle \Rightarrow \langle E \rangle$
  - $\langle S \rangle \Rightarrow \langle E \rangle + \langle E \rangle$
  - $\langle S \rangle \Rightarrow \langle E \rangle + \langle E \rangle * \langle E \rangle$
  - $\langle S \rangle \Rightarrow \text{id} + \langle E \rangle * \langle E \rangle$
  - $\langle S \rangle \Rightarrow \text{id} + \text{id} * \langle E \rangle$
  - $\langle S \rangle \Rightarrow \text{id} + \text{id} * \text{id} \rightarrow \text{Sentencia}$

- La jerarquía de Chomsky:

- Gramáticas no restringidas:  $\alpha \rightarrow \beta$

- $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$
- $\beta \in (N \cup \Sigma)^*$

- Gramáticas sensibles al contexto:  $\alpha \langle A \rangle \beta \rightarrow \alpha \gamma \beta$

- $\alpha, \beta, \gamma \in (N \cup \Sigma)^*$
- $\langle A \rangle \in N$



$\alpha, \beta$  pueden ser  $\lambda$

- Gramáticas independientes de contexto:  $\langle A \rangle \rightarrow \alpha \beta$

- $\langle A \rangle \in N$
- $\alpha \in (N \cup \Sigma)^*$

$(\alpha, \beta = \lambda)$

- Gramáticas regulares:  $\langle A \rangle \rightarrow \underline{a} \langle B \rangle$  ó  $\langle A \rangle \rightarrow \underline{a}$

- $\langle A \rangle, \langle B \rangle \in N$
- $a \in \Sigma$

- Dispositivos reconocedores de lenguajes:

Tipo de gramática	Dispositivo
Regular	Autómata de estados finitos
Libre de contexto	Autómata de pila
Sensible al contexto	Autómata de memoria limitada
No restringida	Máquina de Turing

Handwritten annotations: A bracket on the right groups the first two rows under the label 'AMC'. Another bracket groups the last two rows under the label 'P'. An arrow points from the first column header to the second.