

Material Audiovisual de apoyo

Traductores

**Guillermo Licea Sandoval**

**Febrero de 2022**



# Propósito del curso

- La unidad de aprendizaje tiene como finalidad que el alumno comprenda los principios, técnicas y herramientas para la construcción de traductores, para aplicarlos en la construcción de un traductor que dé solución a problemas reales.

# Competencia general del curso

- Diseñar un traductor, aplicando los principios, técnicas, herramientas y el proceso de construcción de traductores, para su implementación en problemas que requieran el proceso del traductor, mostrando una actitud reflexiva y propositiva.

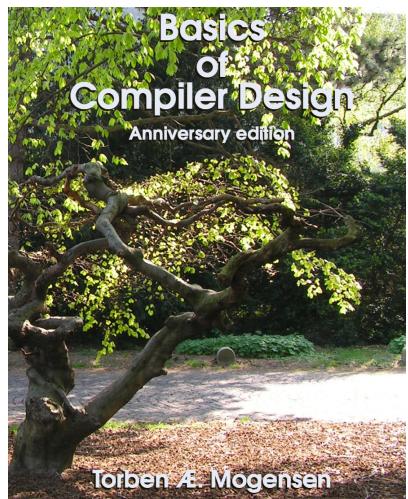
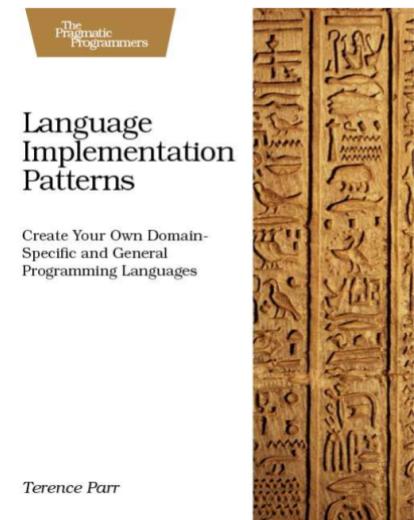
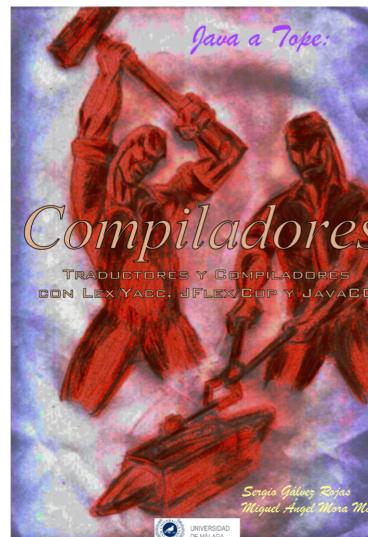
# Evidencia de desempeño

- Elaborar un prototipo de un traductor que incluya un reporte técnico donde se muestre la aplicación correcta de un lenguaje y una gramática independiente al contexto, para resolver un problema real.

# Contenido

1. Construcción de traductores.
2. Aplicaciones de lenguajes.
3. Herramientas para el análisis lexicográfico.
4. Herramientas para el análisis sintáctico.
5. Tablas de símbolos para alcance monolítico y anidado.
6. Tablas de símbolos para agregaciones de datos.
7. Herramientas para la traducción e interpretación.

# Bibliografía



- Traductores y compiladores con Lex/Yacc, JFlex/Cup Y JavaCC. Sergio Gálvez Rojas, Miguel Ángel Mora Mata, Universidad de Málaga, 2005.
- Language implementations patterns. Terence Parr. The pragmatic bookshelf, 2010.
- Basics of compiler design. Torben Ægidius Mogensen. DIKU University of Copenhagen, 2007.
- Análisis y diseño de compiladores. Emiliano Llano Diaz. Exa Ingeniería, 2002.

# Evaluación

- Tareas y prácticas 30 %
- Examen 30%
- Proyecto 40%

# Porque aprender acerca de los traductores ?

- Es considerado un tópico que se debe conocer para considerar que se tiene “cultura” en ciencias de la computación.
- Un buen artesano debe conocer sus herramientas, los compiladores son herramientas importantes para los ingenieros y científicos de la computación.
- Las técnicas utilizadas para la construcción de compiladores también son útiles para otros propósitos.
- Existe la posibilidad de que un ingeniero o científico de la computación tenga que escribir un compilador o intérprete para un lenguaje de dominio específico.



\*programadores  
que programan los  
programas\*



\*programadores que  
programan los  
programas para  
programar\*

# 1. Construcción de traductores

- Introducción
- Traductores (conceptos, tipos, ejemplo)
- Estructura de un traductor
- Ejemplo de compilación

# Introducción

- Ningún profesionista de la computación puede evadir la necesidad de conocer, por lo menos de manera general, la herramienta que utiliza durante su trabajo diario, **el traductor**.
- Existen varias situaciones en las que puede ser muy útil conocer cómo funcionan las partes de un compilador, especialmente aquélla que se encarga de partir los textos fuentes y convertirlos en frases sintácticamente válidas.

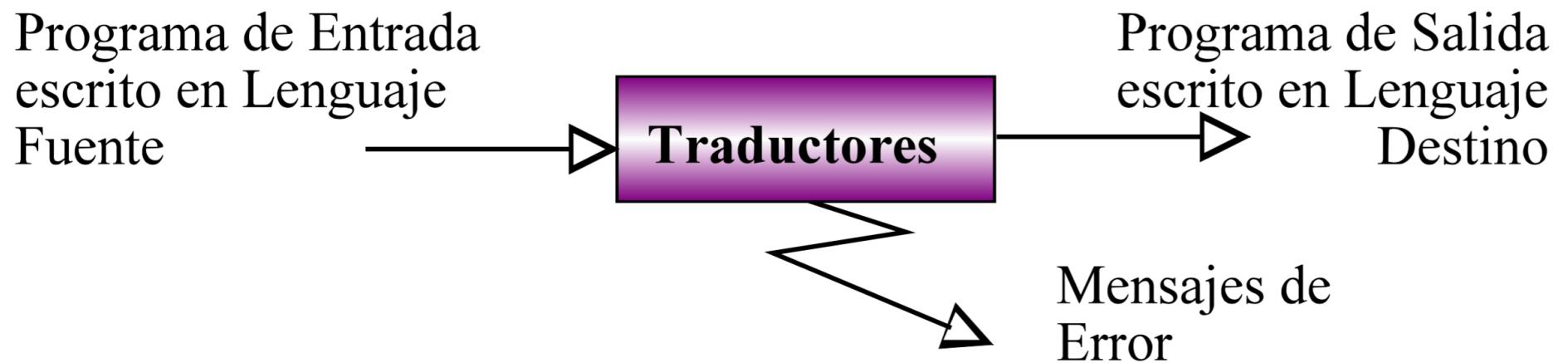
# Introducción

- Algunas aplicaciones de la construcción de traductores pueden ser la creación de preprocesadores para lenguajes que no lo tienen (por ejemplo, para trabajar fácilmente con SQL en C, se puede hacer un preprocesador para introducir SQL inmerso), o incluso la conversión del carácter ASCII 10 (LF) en “<br>” de HTML para pasar texto a la web.

# Traductores

- Un traductor se define como un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino produciendo, si se dan, mensajes de error.
- Los traductores engloban tanto a los compiladores (en los que el lenguaje destino suele ser código máquina) como a los intérpretes (en los que el lenguaje destino está constituido por las acciones atómicas que puede ejecutar el intérprete).

# Traductores



# Traductores

- Es importante destacar la velocidad con la que hoy en día se puede construir un compilador.
- En la década de 1950, se consideró a los traductores como programas notablemente difíciles de escribir.
- El primer compilador de Fortran (Formula Translator), por ejemplo, necesitó para su implementación el equivalente a 18 años de trabajo individual (realmente no se tardó tanto puesto que el trabajo se desarrolló en equipo).
- Antes que la teoría de autómatas y lenguajes formales se aplicara a la creación de traductores, su desarrollo ha estado lleno de problemas y errores.
- Sin embargo, hoy día un compilador básico puede ser el proyecto fin de carrera de cualquier estudiante universitario de computación.

# Traductores - Tipos

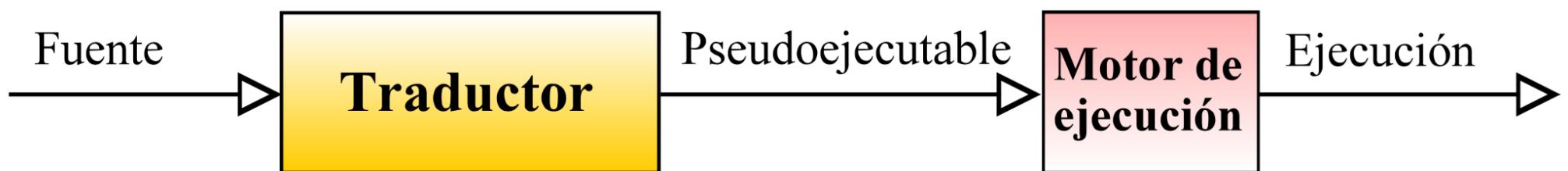
- Desde los orígenes de la computación, ha existido un abismo entre la forma en que las personas expresan sus necesidades y la forma en que una computadora es capaz de interpretar instrucciones.
- Los traductores han intentado salvar este abismo para facilitarle el trabajo a los humanos, lo que ha llevado a aplicar la teoría de autómatas a diferentes campos y áreas concretas de la computación, dando origen a los distintos tipos de traductores.

# Traductores - Tipos

- **Traductores del idioma.** Traducen de un idioma dado a otro, como por ejemplo del inglés al español.
- **Compiladores.** Es aquel traductor que tiene como entrada una sentencia en lenguaje formal y como salida tiene un archivo ejecutable, es decir, realiza una traducción de un código de alto nivel a código máquina.
- **Intérpretes.** Es como un compilador, solo que la salida es una ejecución. El programa de entrada se reconoce y ejecuta a la vez. No se produce un resultado físico (código máquina) sino lógico (una ejecución).

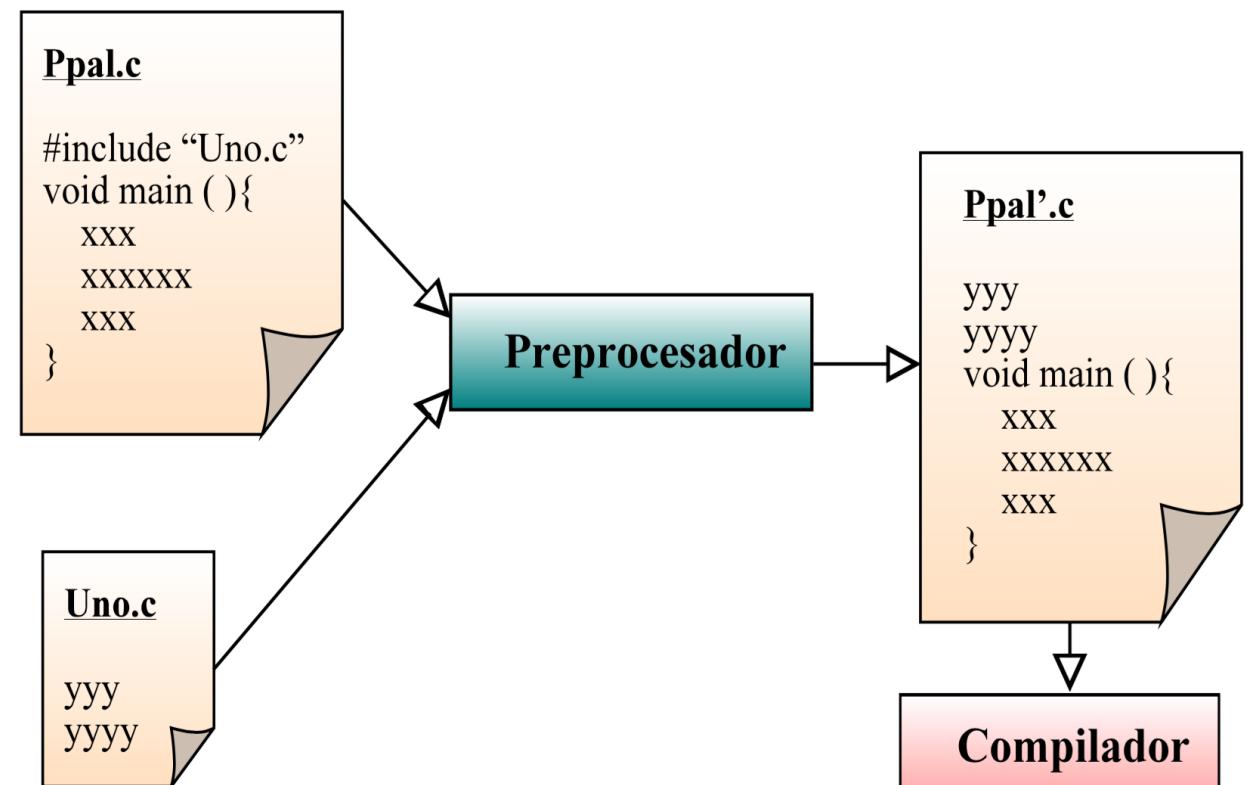
# Traductores - Tipos

- **Intérpretes.** Es como un compilador, solo que la salida es una ejecución. El programa de entrada se reconoce y ejecuta a la vez. No se produce un resultado físico (código máquina) sino lógico (una ejecución).



# Traductores - Tipos

- **Preprocesadores.** Permiten modificar el programa fuente antes de la verdadera compilación. Hacen uso de macroinstrucciones y directivas de compilación.



# Traductores - Tipos

- **Intérpretes de comandos.** Un intérprete de comandos traduce sentencias simples a invocaciones a programas de una biblioteca. Se utilizan especialmente en los sistemas operativos. Los programas invocados pueden residir en el kernel (núcleo) del sistema o estar almacenados en algún dispositivo externo como rutinas ejecutables que se traen a memoria bajo demanda.
- Por ejemplo, si bajo MS-DOS se teclea el comando copy se ejecutará la función de copia de ficheros del sistema operativo, que se encuentra residente en memoria.

# Traductores - Tipos

- **Ensambladores.** Son los pioneros de los compiladores, ya que en los inicios de la computación, los programas se escribían directamente en código máquina, y el primer paso hacia los lenguajes de alto nivel lo constituyen los ensambladores. En lenguaje ensamblador se establece una relación biunívoca entre cada instrucción y una palabra mnemotécnica, de manera que el usuario escribe los programas haciendo uso de los mnemotécnicos, y el ensamblador se encarga de traducirlo a código máquina puro. De esta manera, los ensambladores suelen producir directamente código ejecutable en lugar de producir archivos objeto.
- Un ensamblador es un compilador sencillo, en el que el lenguaje fuente tiene una estructura tan sencilla que permite la traducción de cada sentencia fuente a una única instrucción en código máquina. Al lenguaje que admite este compilador también se le llama lenguaje ensamblador. En definitiva, existe una correspondencia uno a uno entre las instrucciones ensamblador y las instrucciones máquina.

# Traductores - Tipos

- **Conversores fuente-fuente.** Permiten traducir desde un lenguaje de alto nivel a otro lenguaje de alto nivel con lo que se consigue una mayor portabilidad en los programas de alto nivel.
- **Compilador cruzado.** Es un compilador que genera código para ser ejecutado en otra máquina. Se utilizan en la fase de desarrollo de nuevos compiladores. De esta manera es posible, construir el sistema operativo de una nueva computadora recurriendo a un lenguaje de alto nivel, e incluso antes de que dicha nueva computadora disponga siquiera de un compilador.

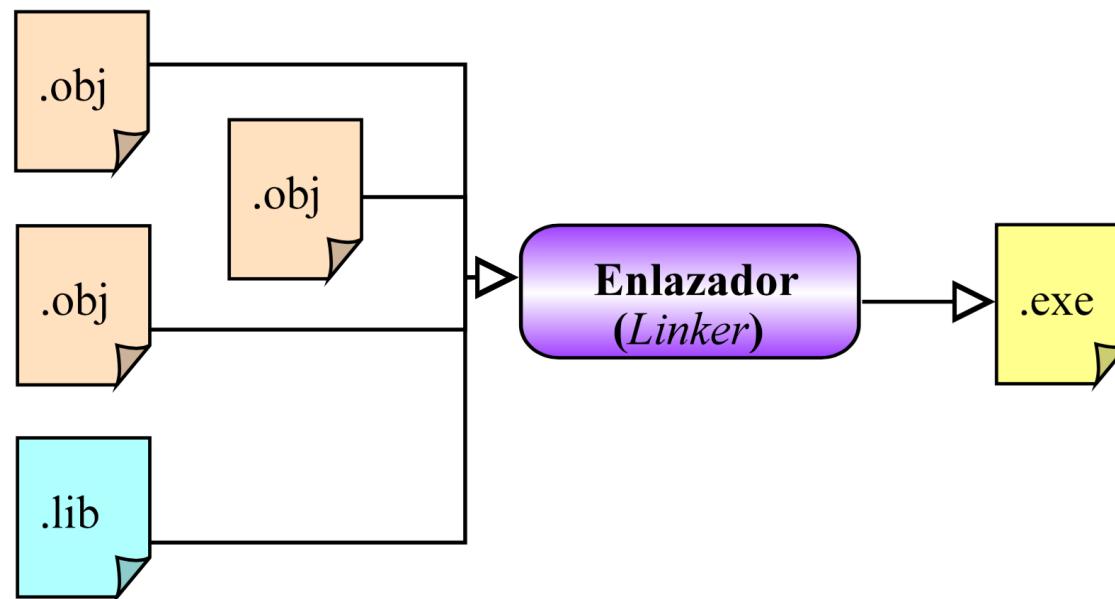
# Traductores - Conceptos

- **Compilación, enlace y carga.** Estas son las tres fases básicas que hay que seguir para que una computadora ejecute la interpretación de un texto escrito mediante la utilización de un lenguaje de alto nivel.
- Por regla general, el compilador no produce directamente un archivo ejecutable, sino que el código generado se estructura en módulos que se almacenan en un archivo objeto. Los archivos objeto poseen información relativa tanto al código máquina como a una tabla de símbolos que almacena la estructura de las variables y tipos utilizados por el programa fuente.



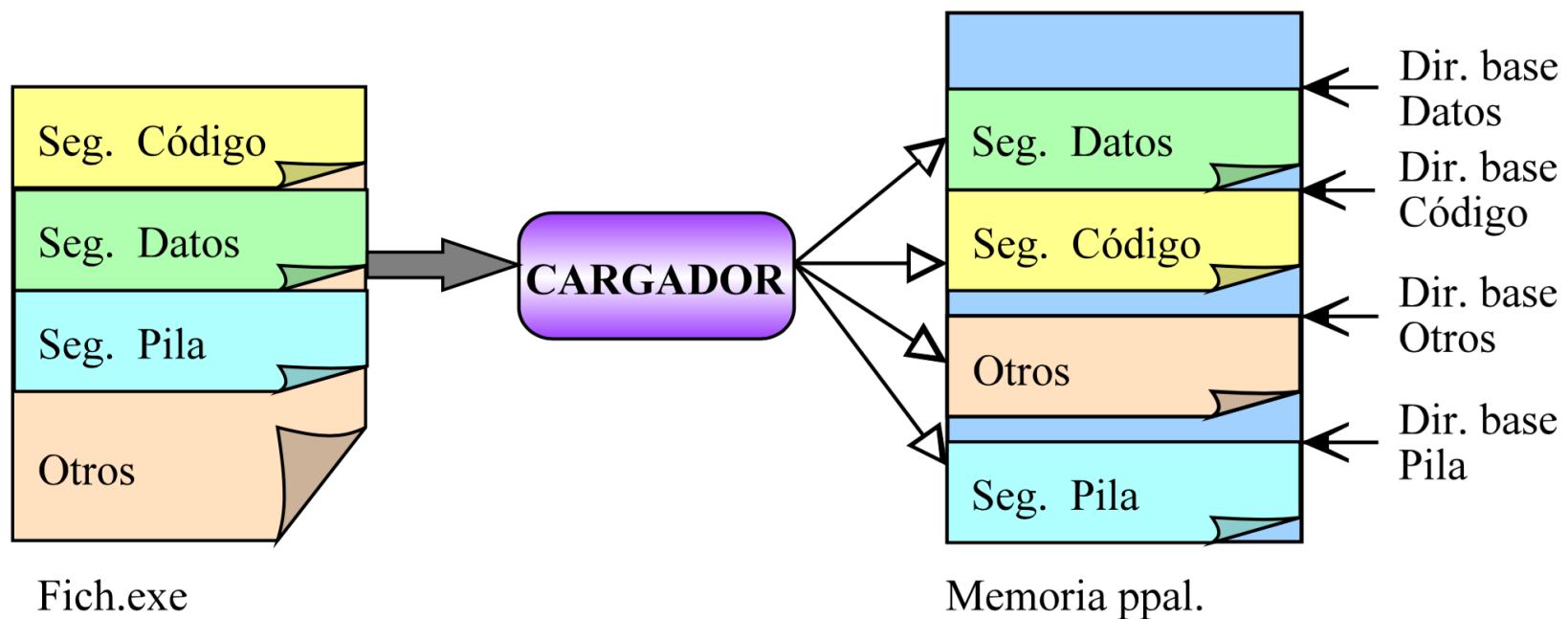
# Traductores - Conceptos

- Durante la fase de enlace, el **enlazador** o linker resuelve las referencias cruzadas, (así se llama a la utilización de objetos externos), que pueden estar declarados en otros archivos objeto, o en bibliotecas, engloba en un bloque los distintos registros que almacenan código máquina, estructura el bloque de memoria que almacena las variables en tiempo de ejecución y genera el ejecutable final incluyendo rutinas adicionales procedentes de bibliotecas.



# Traductores - Conceptos

- El **cargador** carga el archivo .exe, coloca sus diferentes segmentos en memoria (donde el sistema operativo le diga que hay memoria libre para ello) y asigna los registros base a sus posiciones correctas, de manera que las direcciones relativas funcionen correctamente.



# Traductores - Conceptos

- **Pasadas en la compilación.** Es el número de veces que un compilador debe leer el programa fuente para generar el código. Hay algunas situaciones en las que, para realizar la compilación, no es suficiente con leer el archivo fuente una sola vez. Por ejemplo, en situaciones en las que existe recursión indirecta (una función A llama a otra B y la B llama a la A). Cuando se lee el cuerpo de A, no se sabe si B está declarada más adelante o se le ha olvidado al programador; o si lo está, si los parámetros reales coinciden en número y tipo con los formales o no. Por todo esto, en una pasada posterior hay que controlar los errores y llenar los datos que faltan.

# Traductores - Conceptos

- **Compilación incremental.** Cuando se desarrolla un programa fuente, éste se recompila varias veces hasta obtener una versión definitiva libre de errores. Pues bien, en una compilación incremental sólo se recompilan las modificaciones realizadas desde la última compilación. Lo ideal es que sólo se recompilen aquellas partes que contenían los errores o que, en general, hayan sido modificadas, y que el código generado se reinserte con cuidado en el archivo objeto generado en la última compilación. Esto es muy difícil de conseguir y no suele ahorrar tiempo de compilación más que en casos muy concretos.

# Traductores - Conceptos

- **Autocompilador.** Es un compilador escrito en el mismo lenguaje que compila (o parecido). Normalmente, cuando se extiende entre muchas máquinas diferentes el uso de un compilador, y éste se desea mejorar, el nuevo compilador se escribe utilizando el lenguaje del antiguo, de manera que pueda ser compilado por todas esas máquinas diferentes, y dé como resultado un compilador más potente de ese mismo lenguaje.

# Traductores - Conceptos

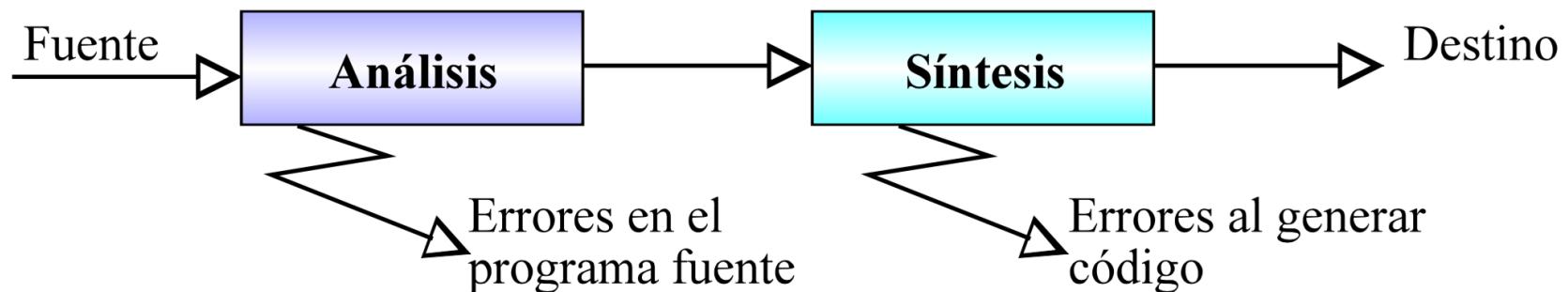
- **Metacompilador.** Un metacompilador es un compilador de compiladores. Se trata de un programa que acepta como entrada la descripción de un lenguaje y produce el compilador de dicho lenguaje. Hoy por hoy no existen metacompiladores completos, pero sí parciales en los que se acepta como entrada una gramática de un lenguaje y se genera un autómata que reconoce cualquier sentencia del lenguaje . A este autómata podemos añadirle código para completar el resto del compilador. Ejemplos de metacompiladores son: Lex, YACC, FLEX, Bison, JavaCC, JLex, Cup, PCCTS, MEDISE, etc.

# Traductores - Conceptos

- **Descompilador.** Un descompilador realiza una labor de traducción inversa, pasa de un código máquina (programa de salida) al equivalente escrito en el lenguaje que lo generó (programa fuente). Cada descompilador trabaja con un lenguaje de alto nivel concreto.
- Los descompiladores se utilizan especialmente cuando el código máquina ha sido generado con opciones de depuración, y contiene información adicional de ayuda al descubrimiento de errores (puntos de ruptura, seguimiento de trazas, opciones de visualización de variables, etc.). También se emplean cuando el compilador no genera código máquina puro, sino pseudocódigo para ser ejecutado a través de un pseudointérprete.

# Estructura de un traductor

- Un traductor divide su labor en dos etapas: una que analiza la entrada y genera estructuras intermedias y otra que sintetiza la salida a partir de dichas estructuras.



# Estructura de un traductor

- Los objetivos de la etapa de análisis son: controlar la corrección del programa fuente, y generar las estructuras necesarias para comenzar la etapa de síntesis.
- Para llevar esto a cabo, la etapa de análisis consta de las siguientes fases: análisis lexicográfico, análisis sintáctico y análisis semántico.

# Estructura de un traductor

- **Análisis lexicográfico.** Divide el programa fuente en los componentes básicos del lenguaje a compilar. Cada componente básico es una subsecuencia de caracteres del programa fuente, y pertenece a una categoría gramatical: números, identificadores de usuario (variables, constantes, tipos, nombres de procedimientos, ...), palabras reservadas, signos de puntuación, etc.
- **Análisis sintáctico.** Comprueba que la estructura de los componentes básicos sea correcta según las reglas gramaticales del lenguaje que se compila.
- **Análisis semántico.** Comprueba que el programa fuente respeta las directrices del lenguaje que se compila (todo lo relacionado con el significado): verificación de tipos, rangos de valores, existencia de variables, etc.

# Estructura de un traductor

- Cualquiera de estas tres fases puede emitir mensajes de error derivados de fallas cometidas por el programador en la redacción de los textos fuente. Mientras más errores controle un compilador, menos problemas dará un programa en tiempo de ejecución. Por ejemplo, el lenguaje C no controla los límites de un arreglo, lo que provoca que en tiempo de ejecución puedan producirse comportamientos del programa de difícil explicación.

# Estructura de un traductor

- La etapa de síntesis construye el programa objeto deseado (equivalente semánticamente al fuente) a partir de las estructuras generadas por la etapa de análisis. Para ello se compone de tres fases fundamentales: generación de código intermedio, generación de código máquina, optimización.

# Estructura de un traductor

- **Generación de código intermedio.** Genera un código independiente de la máquina muy parecido al ensamblador. No se genera código máquina directamente porque así es más fácil hacer pseudocompiladores y además se facilita la optimización de código independientemente del microprocesador.
- **Generación del código máquina.** Crea un bloque de código máquina ejecutable, así como los bloques necesarios destinados a contener los datos.
- **Fase de optimización.** La optimización puede realizarse sobre el código intermedio (de forma independiente de las características concretas del microprocesador), sobre el código máquina, o sobre ambos. Y puede ser una aislada de las dos anteriores, o estar integrada con ellas.

# Estructura de un traductor - Construcción sistemática de compiladores

- Con frecuencia, las fases anteriores se agrupan en una etapa inicial (front- end) y una etapa final (back- end). La etapa inicial comprende aquellas fases, o partes de fases, que dependen exclusivamente del lenguaje fuente y que son independientes de la máquina para la cual se va a generar el código. En la etapa inicial se integran los análisis léxicos y sintácticos, el análisis semántico y la generación de código intermedio. La etapa inicial también puede hacer cierta optimización de código e incluye además, el manejo de errores correspondiente a cada una de esas fases.

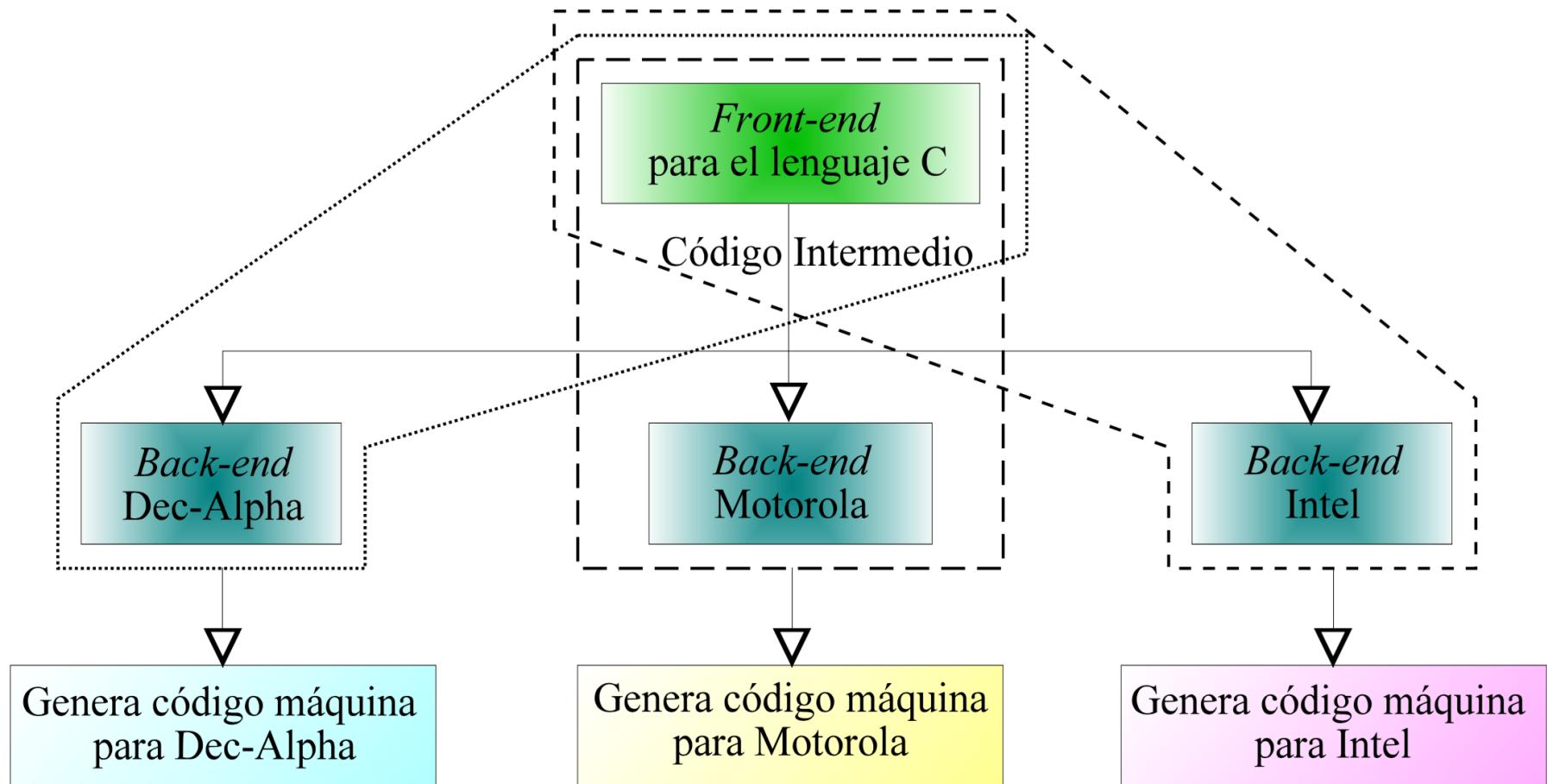
# Estructura de un traductor - Construcción sistemática de compiladores

- La etapa final incluye aquellas fases del compilador que dependen de la máquina destino y que, en general, no dependen del lenguaje fuente sino sólo del lenguaje intermedio. En esta etapa, se encuentran aspectos de la fase de generación de código, además de su optimización, junto con el manejo de errores necesario y el acceso a las estructuras intermedias que haga falta.

# Estructura de un traductor - Construcción sistemática de compiladores

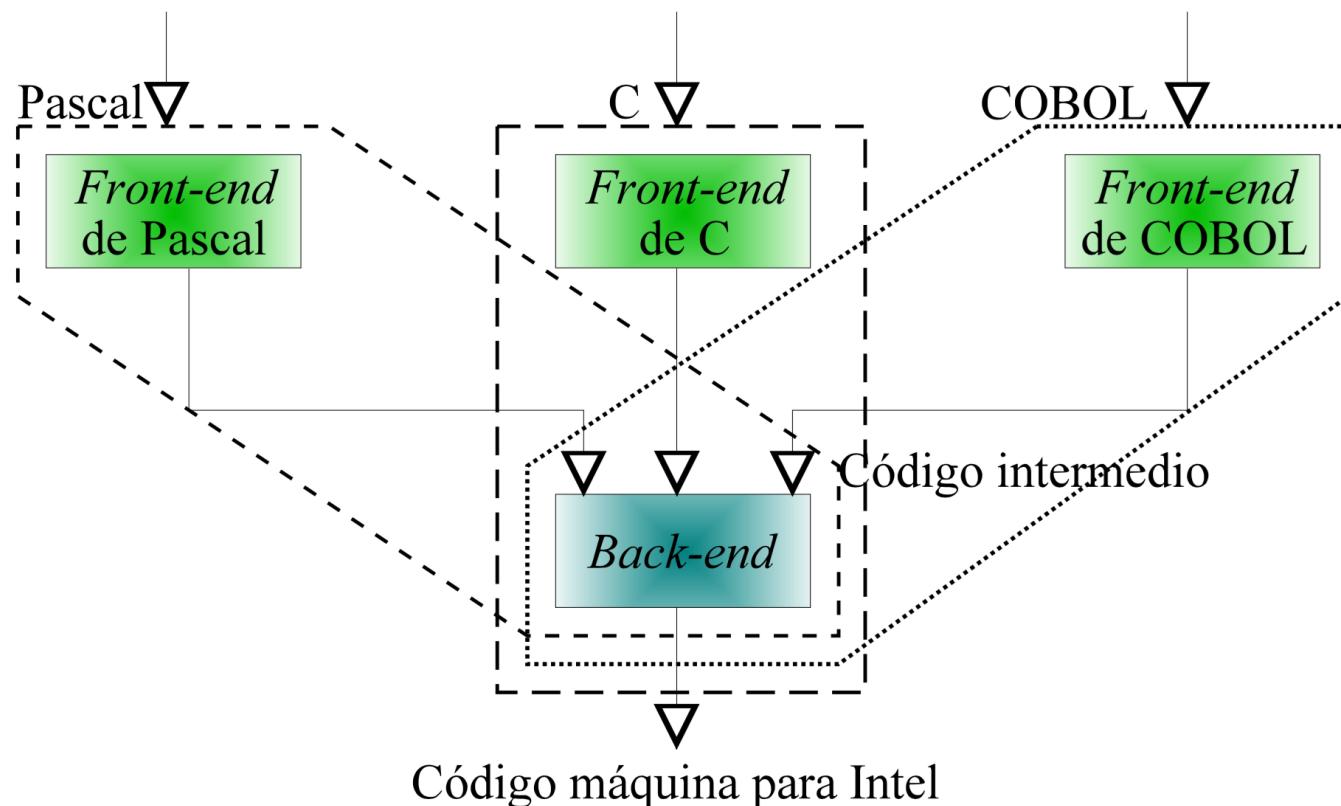
- Se ha convertido en una práctica común el tomar la etapa inicial de un compilador y rehacer su etapa final asociada para producir un compilador para el mismo lenguaje fuente en una máquina distinta. También resulta tentador crear compiladores para varios lenguajes distintos y generar el mismo lenguaje intermedio para, por último, usar una etapa final común para todos ellos, y obtener así varios compiladores para una máquina.

# Estructura de un traductor - Construcción sistemática de compiladores



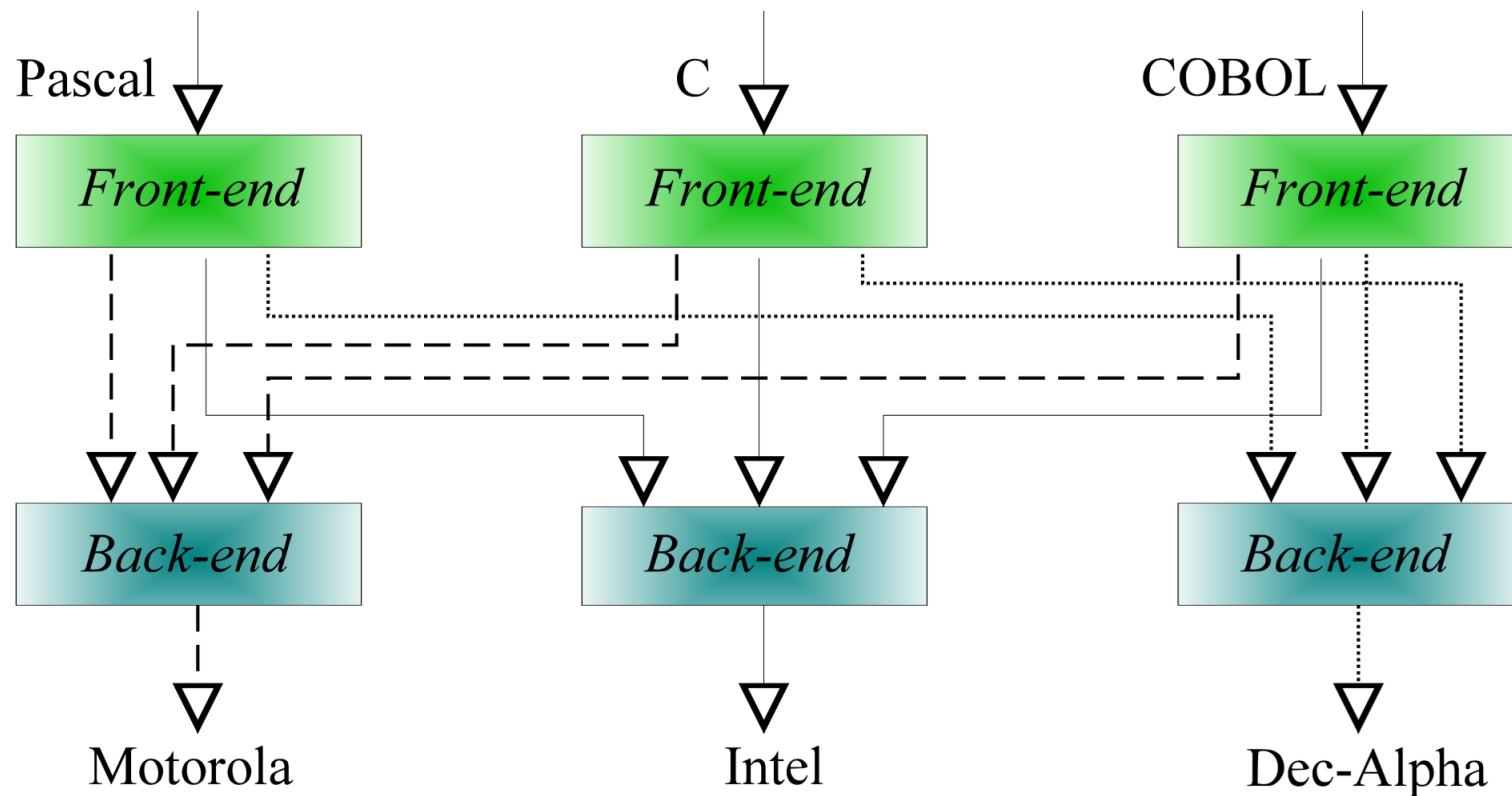
# Estructura de un traductor - Construcción sistemática de compiladores

- De manera inversa se podrían construir tres compiladores de Pascal, C y COBOL para una misma máquina, por ejemplo Intel.



# Estructura de un traductor - Construcción sistemática de compiladores

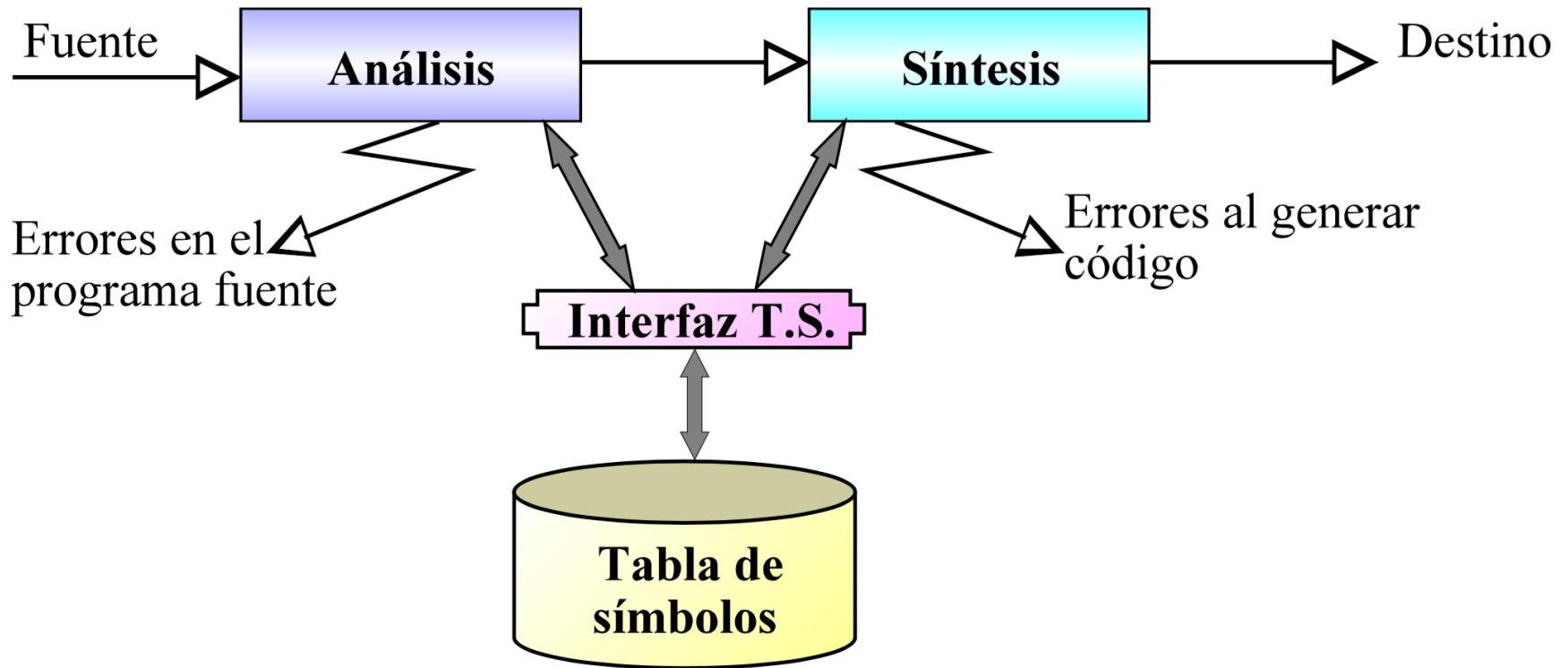
- Por último, la creación de compiladores de Pascal, C y COBOL para las máquinas Dec-Alpha, Motorola e Intel, pasaría por la combinación de los métodos anteriores.



# Estructura de un traductor - La tabla de símbolos

- Una función esencial de un compilador es registrar los identificadores de usuario (nombres de variables, de funciones, de tipos, etc.) utilizados en el programa fuente y reunir información sobre los distintos atributos de cada identificador.
- Estos atributos pueden proporcionar información sobre la memoria asignada a un identificador, la dirección de memoria en que se almacenará en tiempo de ejecución, su tipo, su ámbito (la parte del programa donde es visible), etc.
- La tabla de símbolos es una estructura de datos que posee información sobre los identificadores definidos por el usuario, ya sean constantes, variables, tipos u otros. Dado que puede contener información de diversa índole, debe hacerse de forma que su estructura no sea uniforme, esto es, no se guarda la misma información sobre una variable del programa que sobre un tipo definido por el usuario. Hace funciones de diccionario de datos y su estructura puede ser una tabla hash, un árbol binario de búsqueda, etc., con tal de que las operaciones de acceso sean lo bastante eficientes.

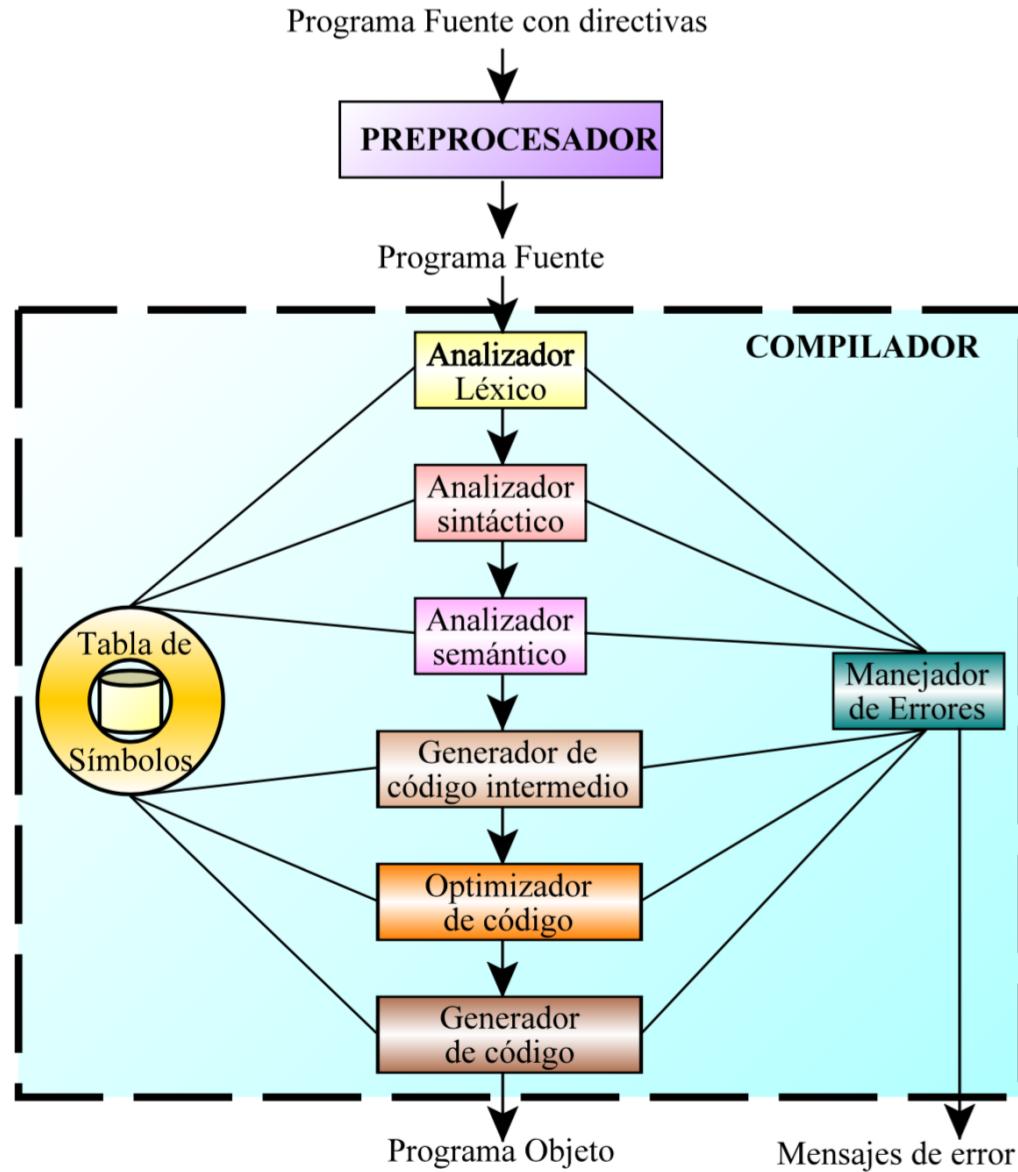
# Estructura de un traductor - La tabla de símbolos



# Estructura de un traductor - La tabla de símbolos

- Tanto la etapa de análisis como la de síntesis accede a esta estructura, por lo que se halla muy acoplada al resto de fases del compilador. Por ello conviene dotar a la tabla de símbolos de una interfaz lo suficientemente genérica como para permitir el cambio de las estructuras internas de almacenamiento sin que estas fases deban ser retocadas. Esto es así porque es común hacer un primer prototipo de un compilador con una tabla de símbolos fácil de construir (y por tanto, ineficiente), y cuando el compilador ya ha sido finalizado, entonces se procede a sustituir la tabla de símbolos por otra más eficiente en función de las necesidades que hayan ido surgiendo a lo largo de la etapa de desarrollo anterior.

# Ejemplo de compilación



# Ejemplo de compilación

- La sentencia con la que se va a trabajar es la siguiente:

```
#define PORCENTAJE 8
comision = fijo + valor * PORCENTAJE;
```

- Para no complicar demasiado el ejemplo, asumiremos que las variables referenciadas han sido previamente declaradas de tipo int, e inicializadas a los valores deseados.

# Ejemplo de compilación - Preprocesamiento

- El código fuente de una aplicación se puede dividir en módulos almacenados en archivos distintos. La tarea de reunir el programa fuente a menudo se confía a un programa distinto, llamado preprocesador.
- El preprocesador también puede expandir abreviaturas, llamadas macros, a proposiciones del lenguaje fuente. En nuestro ejemplo, la constante PORCENTAJE se sustituye por su valor, dando lugar al texto:

**comision = fijo + valor \* 8;**

- que pasa a ser la fuente que entrará al compilador.

# Ejemplo de compilación - Análisis

- En esta etapa se controla que el texto fuente sea correcto en todos los sentidos, y se generan las estructuras necesarias para la generación de código.

# Ejemplo de compilación - Análisis lexicográfico

- En esta fase, la cadena de caracteres que constituye el programa fuente se lee de izquierda a derecha y se agrupa en componentes léxicos, que son secuencias de caracteres que tienen un significado atómico; además el analizador léxico trabaja con la tabla de símbolos introduciendo en ésta los nombres de las variables.
- En nuestro ejemplo los caracteres de la proposición de asignación

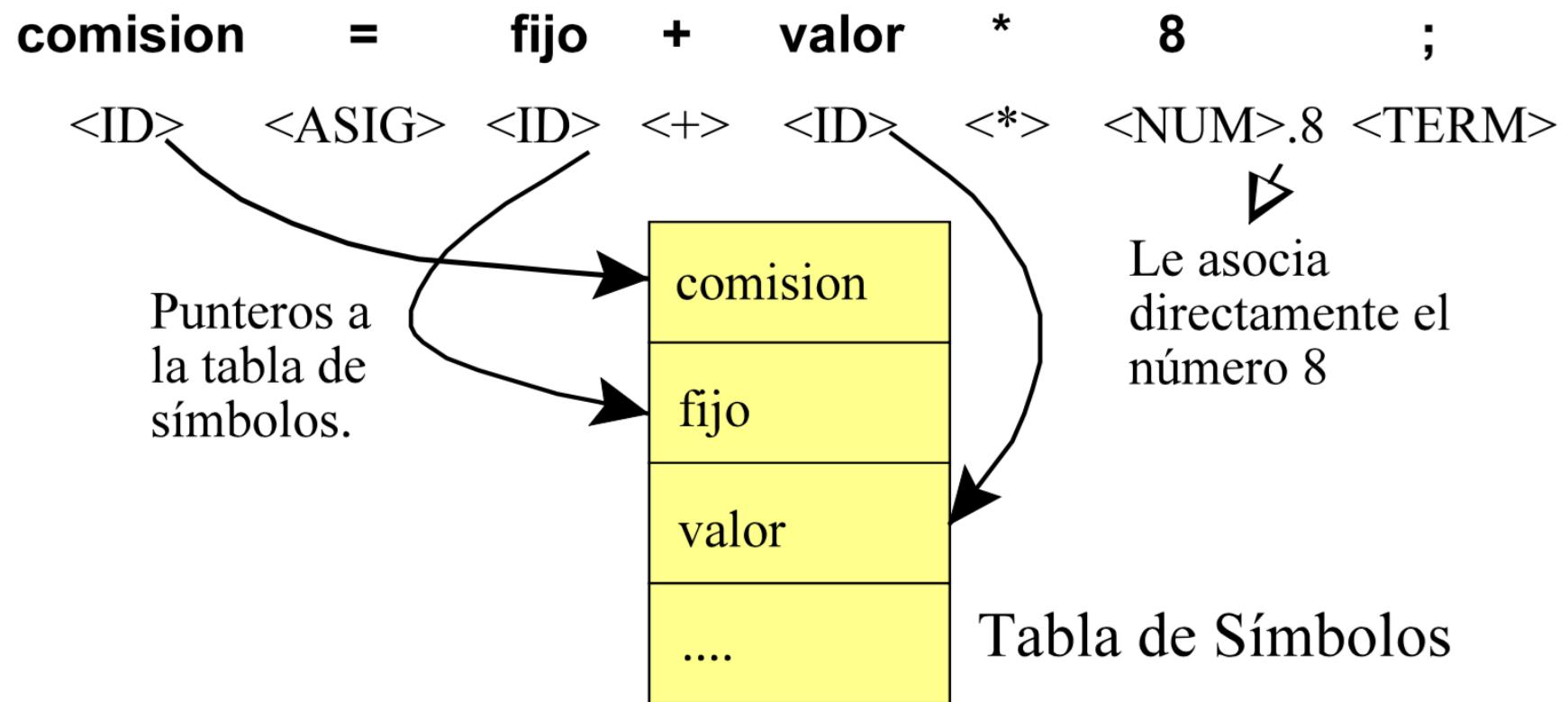
**comision= fijo + valor \* 8 ;**

- se agruparían en componentes léxicos.

# Ejemplo de compilación - Análisis lexicográfico

1. El identificador comision.
2. El símbolo de asignación '='.
3. El identificador fijo.
4. El signo de suma '+'.
5. El identificador valor.
6. El signo de multiplicación '\*'.
7. El número 8.
8. El símbolo de fin de sentencia ';'.

# Ejemplo de compilación - Análisis lexicográfico



# Ejemplo de compilación - Análisis sintáctico

- Trabaja con una gramática de contexto libre y genera el árbol sintáctico que reconoce su sentencia de entrada. En nuestro caso las categorías gramaticales del análisis léxico son los terminales de la gramática.

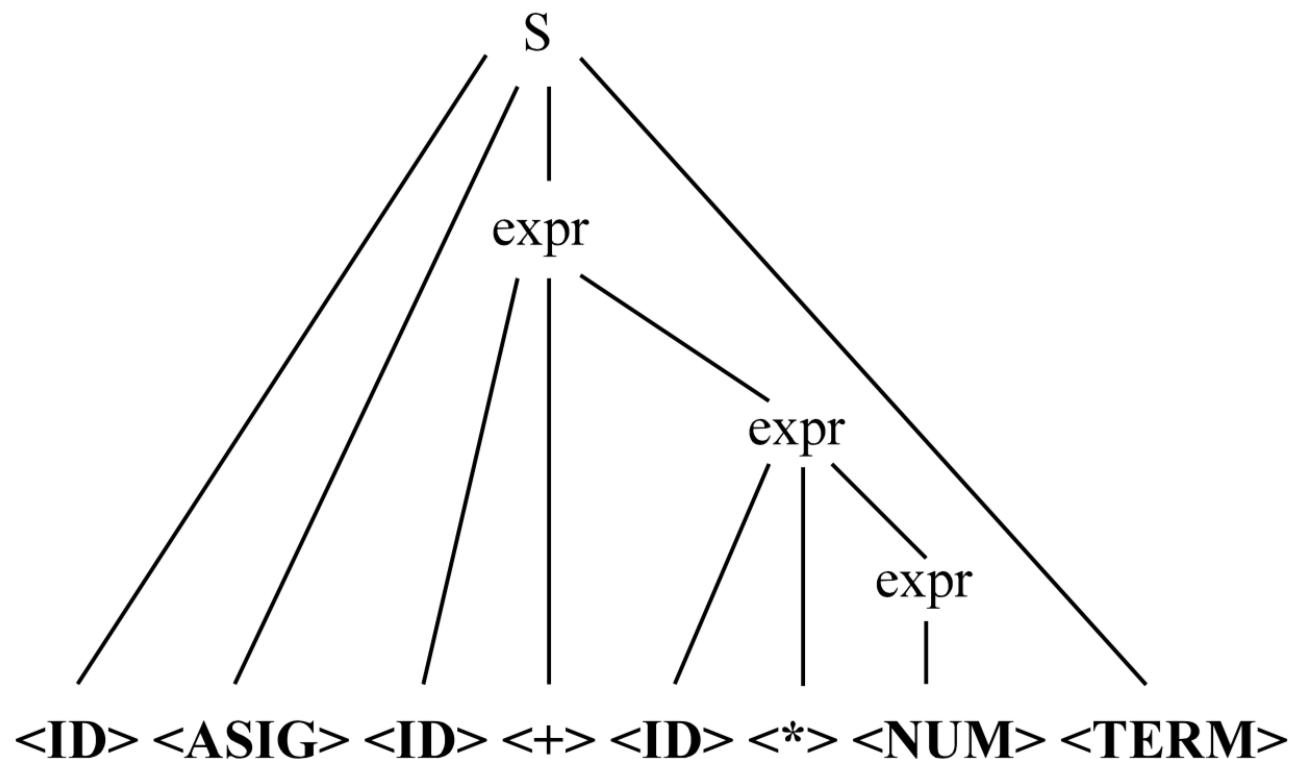
**S → <ID> <ASIG> expr <TERM>**

**expr → <ID>**

**| <ID> <+> expr | <ID> <\*> expr | <NUM>**

# Ejemplo de compilación - Análisis sintáctico

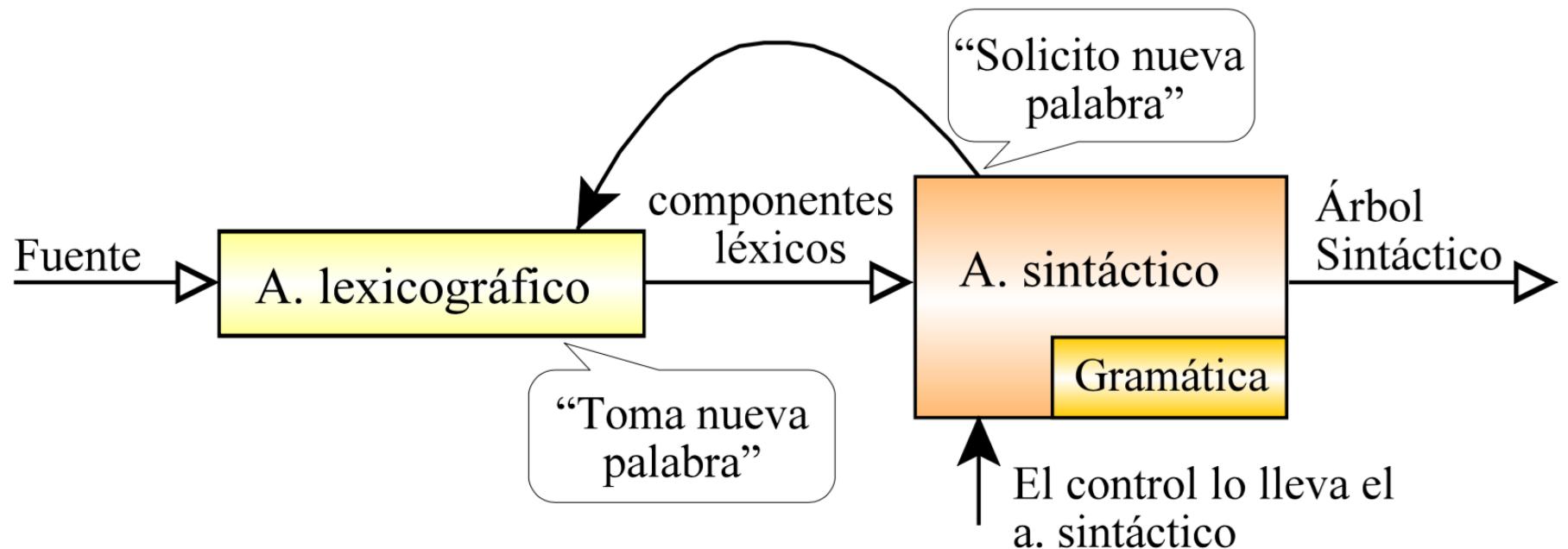
- El análisis sintáctico intenta generar un árbol sintáctico que encaje con la sentencia de entrada.



# Ejemplo de compilación - Compilación dirigida por sintaxis

- El analizador sintáctico construyen su árbol poco a poco (de izquierda a derecha), y cada vez que necesita un nuevo componente léxico para continuar la construcción, lo solicita al analizador lexicográfico; éste lee nuevos caracteres del archivo de entrada hasta conformar un nuevo componente y, una vez obtenido, se lo suministra al analizador sintáctico, quien continúa la construcción del árbol hasta que vuelve a necesitar otro componente, momento en que se reinicia el proceso. Este mecanismo finaliza cuando se ha obtenido el árbol y ya no hay más componentes en el archivo de entrada, o bien cuando es imposible construir el árbol.
- Esto es tan sólo el principio de lo que se denomina compilación dirigida por sintaxis, la cual es aquél mecanismo de compilación en el que el control lo lleva el analizador sintáctico, y todas las demás fases están sometidas a él.

# Ejemplo de compilación - Compilación dirigida por sintaxis



# Ejemplo de compilación - Análisis semántico

- Esta fase revisa el árbol sintáctico junto con los atributos y la tabla de símbolos para tratar de encontrar errores semánticos. Para todo esto se analizan los operadores y operandos de expresiones y proposiciones. Finalmente reúne la información necesaria sobre los tipos de datos para la fase posterior de generación de código.
- El componente más importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si los operandos de cada operador son compatibles según la especificación del lenguaje fuente. Si suponemos que nuestro lenguaje solo trabaja con números reales, la salida de esta fase sería su mismo árbol, excepto porque el atributo de **<NUM>**, que era el entero 8 a la entrada, ahora pasaría a ser el real 8,0. Además se ha debido controlar que las variables implicadas en la sentencia: comision, fijo y valor son compatibles con el tipo numérico de la constante 8,0.

# Ejemplo de compilación - Síntesis

- En la etapa anterior se ha controlado que el programa de entrada es correcto. Por tanto, el compilador ya se encuentra en disposición de generar el código máquina equivalente semánticamente al programa fuente. Para ello se parte de las estructuras generadas en dicha etapa anterior: árbol sintáctico y tabla de símbolos.

# Ejemplo de compilación - Generación de código intermedio

- Después de la etapa de análisis, se suele generar una representación intermedia explícita del programa fuente. Dicha representación intermedia se puede considerar como un programa para una máquina abstracta.
- Cualquier representación intermedia debe tener dos propiedades importantes; debe ser fácil de generar y fácil de traducir al código máquina destino.
- En el presente ejemplo se trabajará con una forma intermedia llamada “código de tres direcciones”, que es muy parecida a un lenguaje ensamblador para un microprocesador que carece de registros y sólo es capaz de trabajar con direcciones de memoria y literales.

# Ejemplo de compilación - Generación de código intermedio

- En el código de tres direcciones cada instrucción tiene como máximo tres operandos. Siguiendo el ejemplo propuesto, se generaría el siguiente código de tres direcciones:

**t1 = 8.0**

**t2 = valor \* t1**

**t3 = fijo + t2**

**comision = t3**

# Ejemplo de compilación - Generación de código intermedio

- De este ejemplo se pueden destacar varias propiedades del código intermedio escogido:
  - Cada instrucción de tres direcciones tiene a lo sumo un operador, además de la asignación.
  - El compilador debe generar un nombre temporal para guardar los valores intermedios calculados por cada instrucción: **t1**, **t2** y **t3**.
  - Algunas instrucciones tienen menos de tres operandos, como la primera y la última instrucciones del ejemplo.

# Ejemplo de compilación - Optimización de código

- Esta fase trata de mejorar el código intermedio, de modo que en la siguiente fase resulte un código de máquina más rápido de ejecutar.
- Algunas optimizaciones son triviales. En nuestro ejemplo hay una forma mejor de realizar el cálculo de la comisión, y pasa por realizar sustituciones triviales en la segunda y cuarta instrucciones, obteniéndose:

**t2 = valor \* 8.0**

**comision= fijo + t2**

# Ejemplo de compilación - Optimización de código

- El compilador puede deducir que todas las apariciones de la variable **t1** pueden sustituirse por la constante 8,0, ya que a **t1** se le asigna un valor que ya no cambia, de modo que la primera instrucción se puede eliminar.
- Algo parecido sucede con la variable **t3**, que se utiliza sólo una vez, para transmitir su valor a **comision** en una asignación directa, luego resulta seguro sustituir **comision** por **t3**, a raíz de lo cual se elimina otra de las líneas del código intermedio.

# Ejemplo de compilación - Generación de código máquina

- La fase final de un compilador es la generación de código objeto, que por lo general consiste en código máquina reubicable o código ensamblador.
- Cada una de las variables usadas por el programa se traduce a una dirección de memoria (esto también se ha podido hacer en la fase de generación de código intermedio).
- Después, cada una de las instrucciones intermedias se traduce a una secuencia de instrucciones de máquina que ejecuta la misma tarea.
- Un aspecto decisivo es la asignación de variables a registros.

# Ejemplo de compilación - Generación de código máquina

- Siguiendo el mismo ejemplo, y utilizando los registros R1 y R2 de un microprocesador hipotético, la traducción del código optimizado podría ser:

**MOVE [1Ah], R1**

**MULT #8.0, R1**

**MOVE [15h], R2**

**ADD R1, R2**

**MOVE R2, [10h]**

# Ejemplo de compilación - Generación de código máquina

- El primer y segundo operandos de cada instrucción especifican una fuente y un destino, respectivamente.
- Este código traslada el contenido de la dirección [1Ah] al registro R1, después lo multiplica por la constante real 8.0.
- La tercera instrucción pasa el contenido de la dirección [15h] al registro R2.
- La cuarta instrucción le suma el valor previamente calculado en el registro R1.
- Por último el valor del registro R2 se pasa a la dirección [10h]. Como el lector puede suponer, la variable comision se almacena en la dirección [10h], fijo en [15h] y valor en [1Ah].