UAI Universidad Abierta Interamericana

SISTEMAS OPERATIVOS

Mg. Leandro Ezequiel Mascarello

<leandro.mascarello@uai.edu.ar>

UAIOnline ultra

A la hora de elegir un lenguaje de programación, se debe prestar especial atención a dos cosas:

- El lenguaje debe contar con todos los componentes básicos necesarios para el proyecto de software que se quiera desarrollar.
- Segundo tiene que permitir programar e implementar este proyecto de la manera más sencilla posible.

La **buena legibilidad y simplicidad del código fuente** son fundamentales para garantizar lo segundo, porque estas características no solo facilitan el aprendizaje del lenguaje de programación, sino también, obviamente, su posterior utilización en el día a día.

Compiladores

En primer lugar, para que el ordenador o el procesador puedan comprender las instrucciones que contiene un programa desarrollado previamente, el código fuente escrito en los lenguajes de programación actuales debe convertirse a un formato legible por máquina.

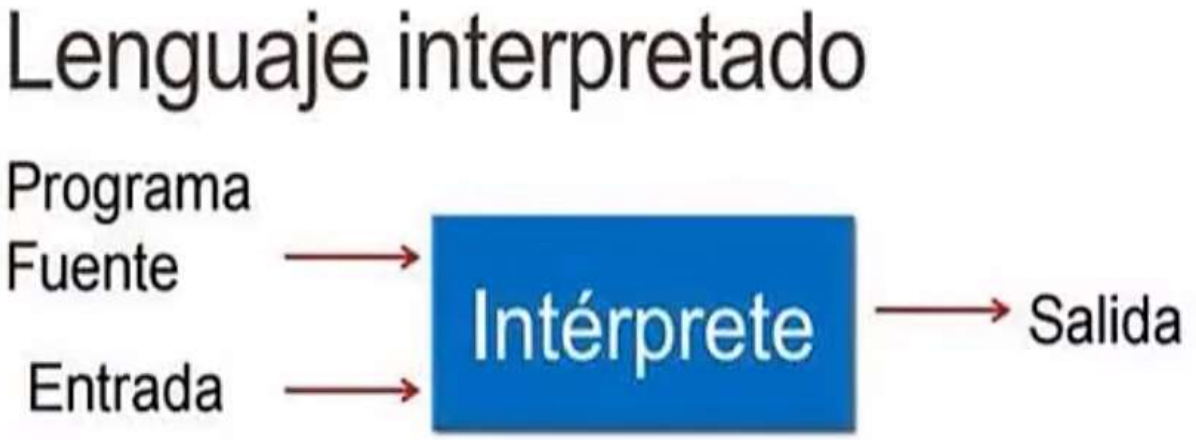
De este procedimiento, dependiendo del lenguaje de programación, se encarga un compilador o un intérprete. ¿Qué son exactamente estos dos programas? Y ¿en qué se diferencian?

¿Qué es un intérprete?

Un intérprete es un **programa informático** que procesa el código fuente de un proyecto de software durante su tiempo de ejecución, es decir, mientras el software se está ejecutando, y actúa como una **interfaz** entre ese proyecto y el procesador.

Un intérprete siempre procesa el código línea por línea, de modo que lee, analiza y prepara cada secuencia de forma consecutiva para el procesador.

Este principio también se aplica a las secuencias recurrentes, que se ejecutan de nuevo cada vez que vuelven a aparecer en el código.



¿Qué es un intérprete?

Para procesar el código fuente del software, el intérprete recurre a sus propias bibliotecas internas: en cuanto una **línea de código fuente se ha traducido a los correspondientes comandos legibles por máquina**, esta se envía directamente al procesador.

El proceso de conversión no finaliza hasta que se ha interpretado todo el código.

Solo se interrumpe prematuramente si se produce un fallo durante el procesamiento, lo que **simplifica mucho la resolución de los errores**, ya que la línea de código problemática se detecta inmediatamente después de ocurrir el fallo.

> **! Nota**
>
> **BASIC**, **Perl**, **Python**, **Ruby** y **PHP** son algunos de los lenguajes de programación más famosos que dependen de un intérprete para ser traducidos de código fuente a código máquina. Por ello, también suelen llamarse **lenguajes interpretados**.

¿Qué es un compilador?

- Un compilador es un **programa informático** que traduce todo el código fuente de un proyecto de software a código máquina antes de ejecutarlo.
- Solo entonces el procesador ejecuta el software, obteniendo todas las instrucciones en código máquina antes de comenzar.
- De esta manera, el procesador cuenta con todos los componentes necesarios para ejecutar el software, procesar las entradas y generar los resultados.
- No obstante, en muchos casos, durante el proceso de compilación tiene lugar un paso intermedio fundamental: antes de generar la traducción final en código máquina, la mayoría de los compiladores suelen convertir el código fuente en un **código intermedio** (también llamado código objeto) que, a menudo, es compatible con diversas plataformas y que, además, también puede ser utilizado por un intérprete.
- Al producir el código, el compilador determina **qué instrucciones** van a enviarse al procesador y **en qué orden**. Si las instrucciones no son interdependientes, incluso es posible que puedan procesarse en paralelo.

Estructura General de los Compiladores

Front End (Análisis)

Fases que dependen del lenguaje fuente

- Análisis Léxico.
- Análisis Sintáctico.
- Análisis Semántico (Estático).
- Creación de la Tabla de Símbolos.
- Generación de Código Intermedio.
- Optimización.
- Manejo de errores correspondiente a las fases del Front End.

Back End (Síntesis)

Fases que dependen de la máquina destino

- Generación de la salida.
- Optimización.
- Manejo de errores correspondiente a las fases del Back End..
- Operaciones sobre la Tabla de Símbolos.

La parte de la **compilación** está dividida en dos tareas principales:

Análisis (Front End): En la tarea del análisis se divide al programa fuente en sus elementos, componentes y crea una representación intermedia del programa fuente.

Síntesis (Back End): La tarea de la síntesis construye el programa objeto deseado a partir de la representación intermedia. La *síntesis* es la que requiere las técnicas más especializadas.

Compiladores

Análisis léxico (*explorador o scanner*)

Esta fase del compilador efectúa la lectura real del programa fuente, el cual generalmente está en la forma de un flujo de caracteres. El rastreador realiza lo que se conoce como análisis léxico: recolecta secuencias de caracteres en unidades significativas denominadas **tokens**: las cuales son como las palabras de un lenguaje natural, como el español.

Análisis Sintáctico (*explorador o scanner*)

El analizador sintáctico recibe el código fuente en la forma de **tokens** proveniente del analizador léxico y realiza el **análisis sintáctico**, que determina la estructura del programa. Esto es semejante a realizar el análisis gramatical sobre una frase en un lenguaje natural. El análisis sintáctico determina los elementos estructurales del programa y su relaciones. Los resultados del análisis sintáctico por lo regular se representan como un **árbol de análisis gramatical** o **un árbol sintáctico**.

Análisis Semántico

La semántica de un programa es su "significado", en oposición a su sintaxis, o estructura. La semántica de un programa determina su comportamiento durante el tiempo de ejecución, pero la mayoría de los lenguajes de programación tienen características que se pueden determinar antes de la ejecución e incluso no se pueden expresar de manera adecuada como sintaxis y analizarse mediante el analizador semántico.

Generador de código intermedio

El *código intermedio* es un código abstracto independiente de la máquina para la que se generará el código objeto. El código intermedio ha de cumplir dos requisitos importantes: ser fácil de producir a partir del análisis sintáctico, y ser fácil de traducir al lenguaje objeto. Esta fase puede no existir si se genera directamente código máquina, pero suele ser conveniente emplearla.

Optimizador de código

Esta etapa es utilizada para el mejoramiento del código o para su optimización. El punto más anticipado en el que la mayoría de las etapas de operación se pueden realizar es precisamente después del análisis semántico, y puede haber posibilidad para el mejoramiento del código que dependerán sólo del código fuente.

Generación de código destino

El generador de código destino toma el código optimizado y genera el código para la máquina objetivo (ensamblador). Las posiciones de memoria se seleccionan para cauda una de las variables usadas por el programa. 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 registro.

Símbolo: Es una entidad abstracta, es un elemento fundamental del lenguaje coma las letras y los dígitos son ejemplos de símbolos.

Alfabeto (∑): Es un conjunto finito no vacío de símbolos utilizados en un lenguaje. Convencionalmente se utiliza el símbolo ∑ par designar un alfabeto. ∑ = {A, B, C, …, X,Y, Z, a, b, c, …, x, y, z, 0, 1, 2, …, 9}, denota un alfabeto qué consiste de los símbolos de la "A" a la "Z", de la "a" a la "z" y de los números decimales del "0" al "9".

Unidad léxica o Palabra (W): Es un conjunto finito de símbolos yuxtapuestos de un alfabeto. UL = #aabbaa# denota la cadena de caracteres llamadas UL (Unidad Léxica) que comience y termina con el símbolo # y contiene la cadena "aabbaa" entre los caracteres #.

Palabra vacía: Es una palabra que consiste de cero elementos, la cual representamos por medio de ε.

Longitud de una palabra: Sea W una palabra, su longitud denotada por |W| es el número de símbolos que la constituyen. W="aabbaa" su longitud es de 6, si la palabra cadena es |ε| su longitud es 0.

Token: Es un número entero que se le asigna a una unidad léxica.

Lenguaje (L): Es un conjunto finito o infinito de palabras, sobre un alfabeto determinado. Así, sobre el alfabeto $\sum = \{a, b, c\}$ se puede determinar el lenguaje formado por la palabra que contiene la misma cantidad de a's, b's y c's. Formalmente esto se pondría como $L = \{W \mid |W|a = |W|b = |W|c\}$ esto es, $L = \{\varepsilon, abc, bca, cba, bac, acb, \ldots\}$.

Producción (P): Reglas para la sustitución de cadenas, es decir, proceso de convertir símbolos no terminales en símbolos terminales; El símbolo → se usa comúnmente para representar las producciones. Por ejemplo, la producción $S \rightarrow a\ b$ significa qué puedes sustituir S por a b, o que S se puede definir como a b.

Símbolo no terminal (N): Es una representación de una instancia a distintos nivel de abstracción, esto es, son los símbolos que se encuentran al lado izquierdo de una producción. Los símbolos no terminales también son conocidos como variables sintácticas, son representados por la letra **N** y dicen que N^* representa el conjunto de todas las cadenas posibles N.

Símbolo terminal (T): Es una instancia específica de un símbolo no terminal, esto es, son los símbolos que aparecen en una frase. Estos símbolos nunca aparecen en el lado izquierdo de una producción. Las unidades léxicas int, if, While son ejemplos de símbolos terminales, los cual es pertenecen a la gramática del lenguaje se.

Gramática (G): La gramática es un conjunto de reglas que definen si una secuencia arbitraria de símbolos está escrita correctamente, una gramática es una tupla de 4 elementos denotada cómo G = { N, T, S, P}.

Donde:

G: Gramática del lenguajes.
N: Conjunto finito de símbolos no terminales.
T: Conjunto finito de símbolos terminales.
S: Símbolo inicial.
P:Conjunto de producciones.

Tipos de Compiladores

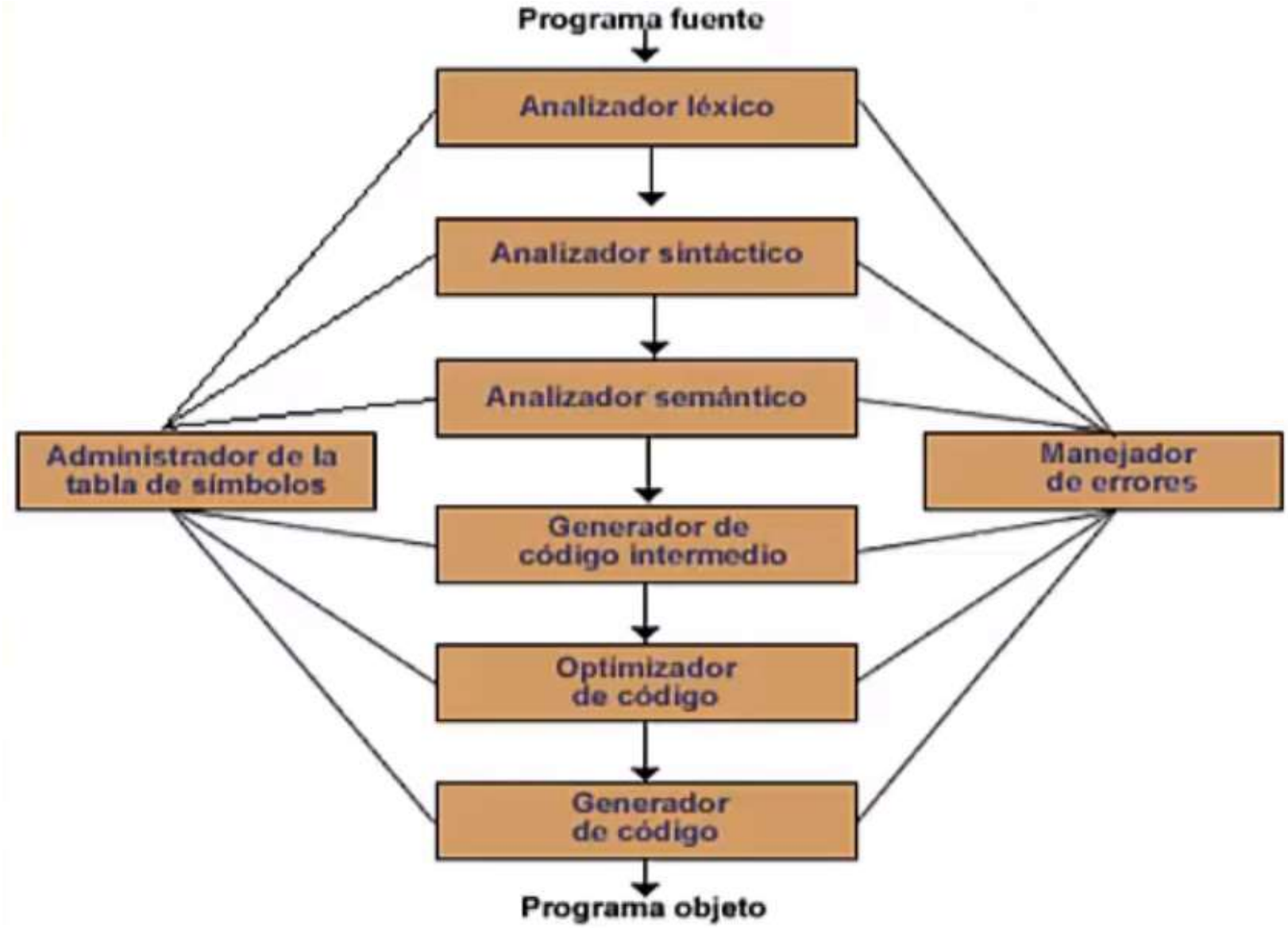
- Compiladores cruzados
- Compiladores optimizadores
- Compiladores de una sola pasada
- Compiladores de varias pasadas
- Compiladores

Fases de un Compilador

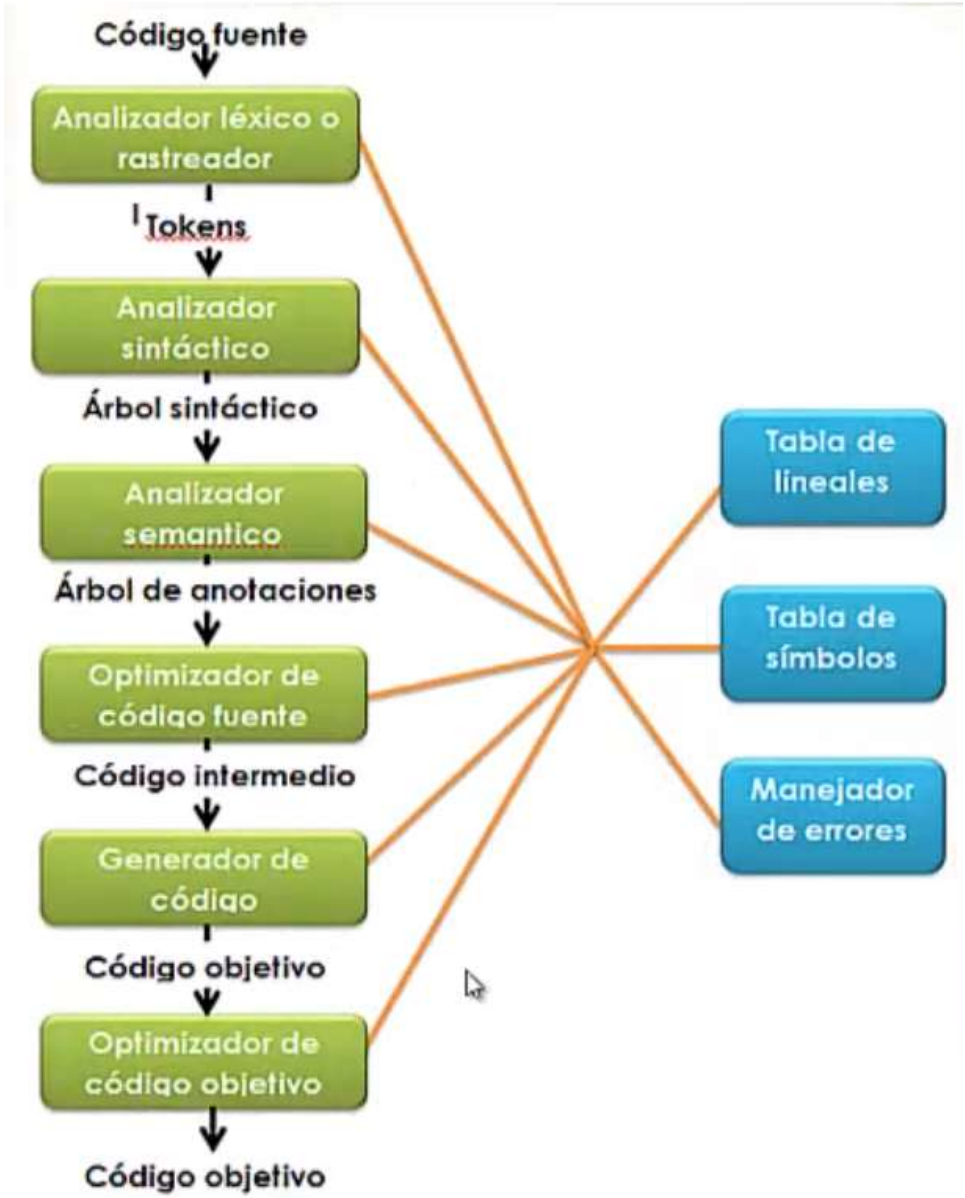
- Un compilador está formado por dos procesos **análisis** y **síntesis.**
- **Análisis:** El cual se trata de la escritura correcta del código fuente. Esta a su vez comprende varias fases:
- · **Análisis léxico**
- · **Análisis sintáctico**
- · **Análisis semántico**

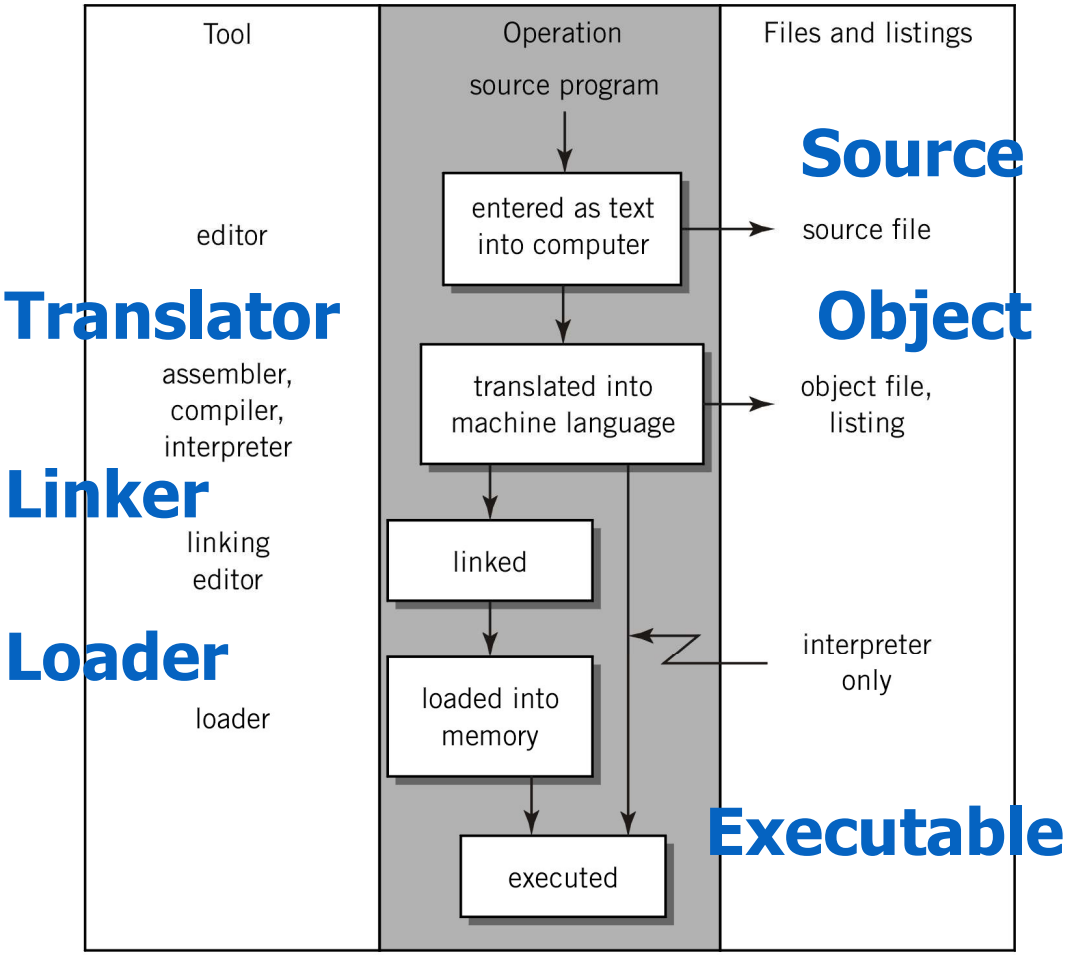
- **Síntesis:** Después del proceso de análisis se procede a generar grupos de los componentes que conforman el programa, para generar una salida.
- · **Generación de código intermedio**
- · **Optimización de código**
- · **Generación de código**

Fases de un Compilador



Fases de un Compilador

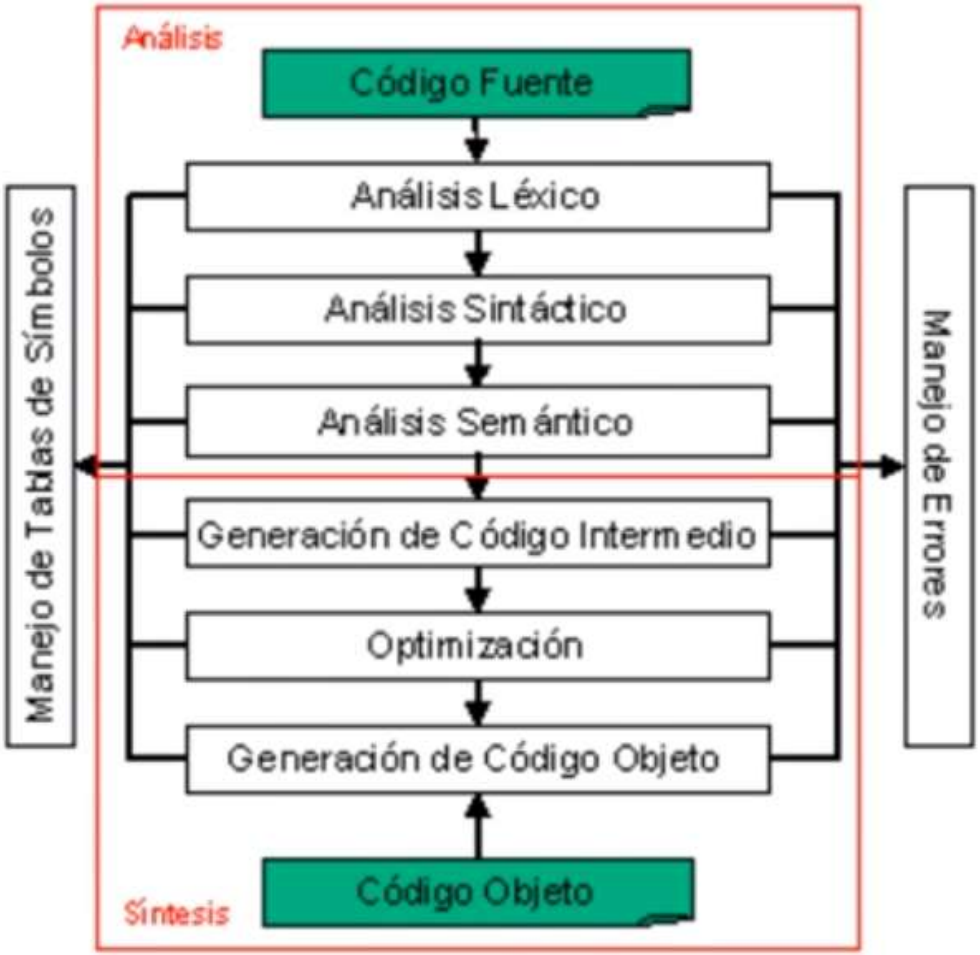




Analizador Léxico

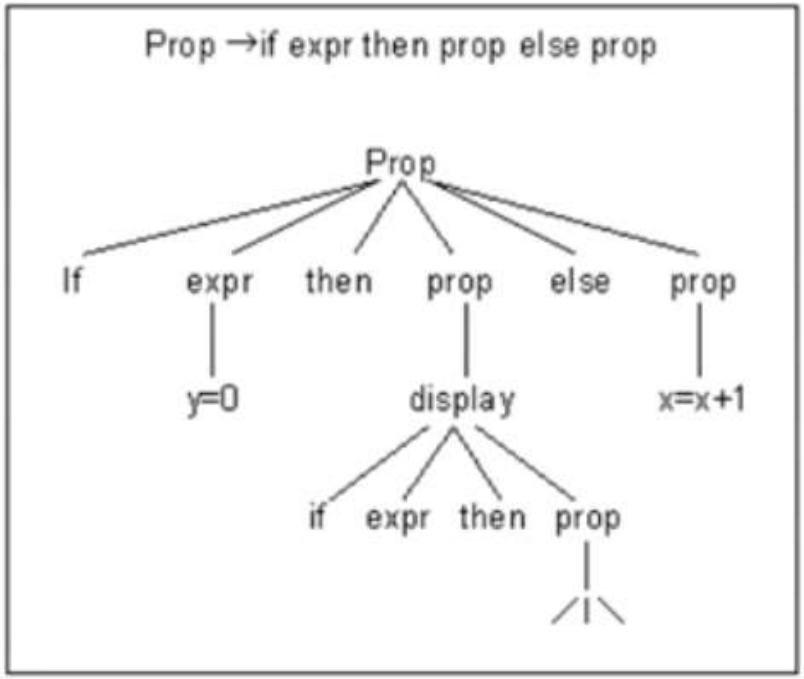
EJEMPLO

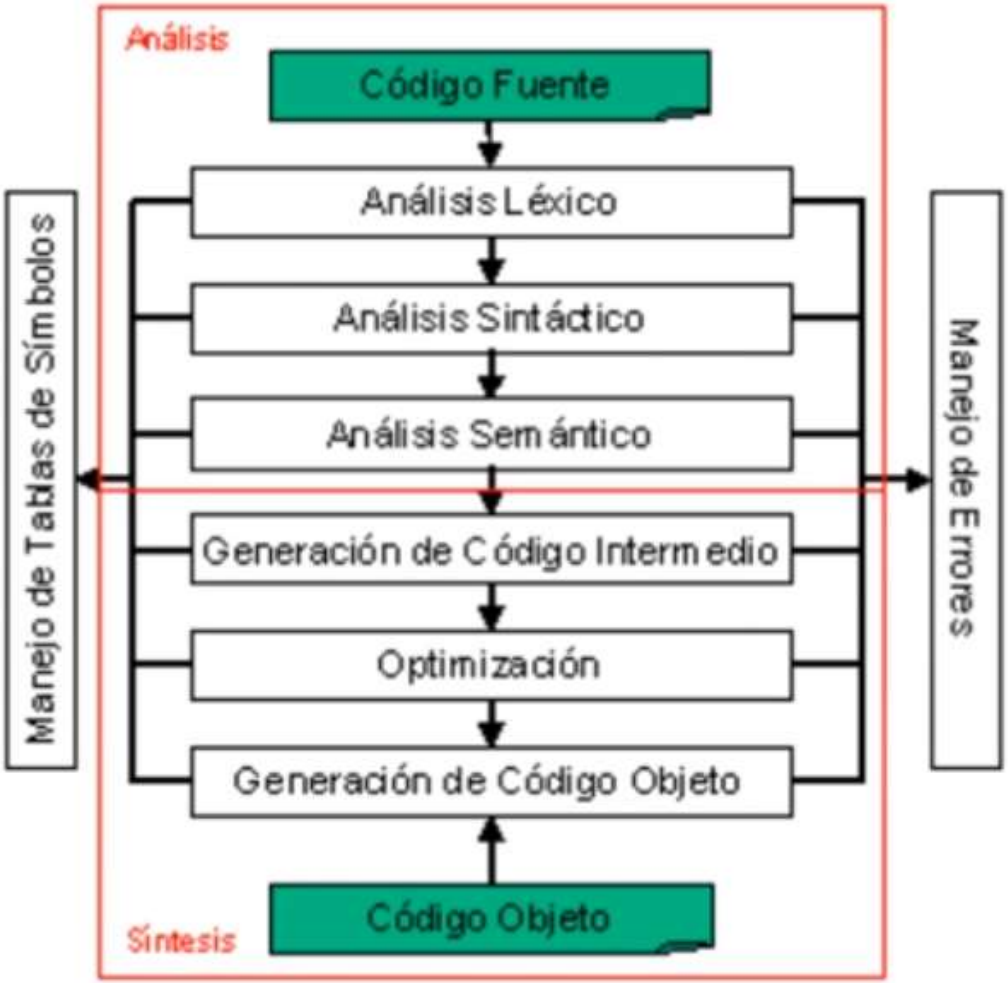
Token	Atributo	Observaciones
IF	20	Palabra reservada
cuenta	1	Identificador
=	15	Operador de comparación
sueldo	1	Identificador
THEN	21	Palabra reservada
jefe	1	Identificador
:=	10	Asignación
Justo	1	Identificador
;	27	Separador de sentencias



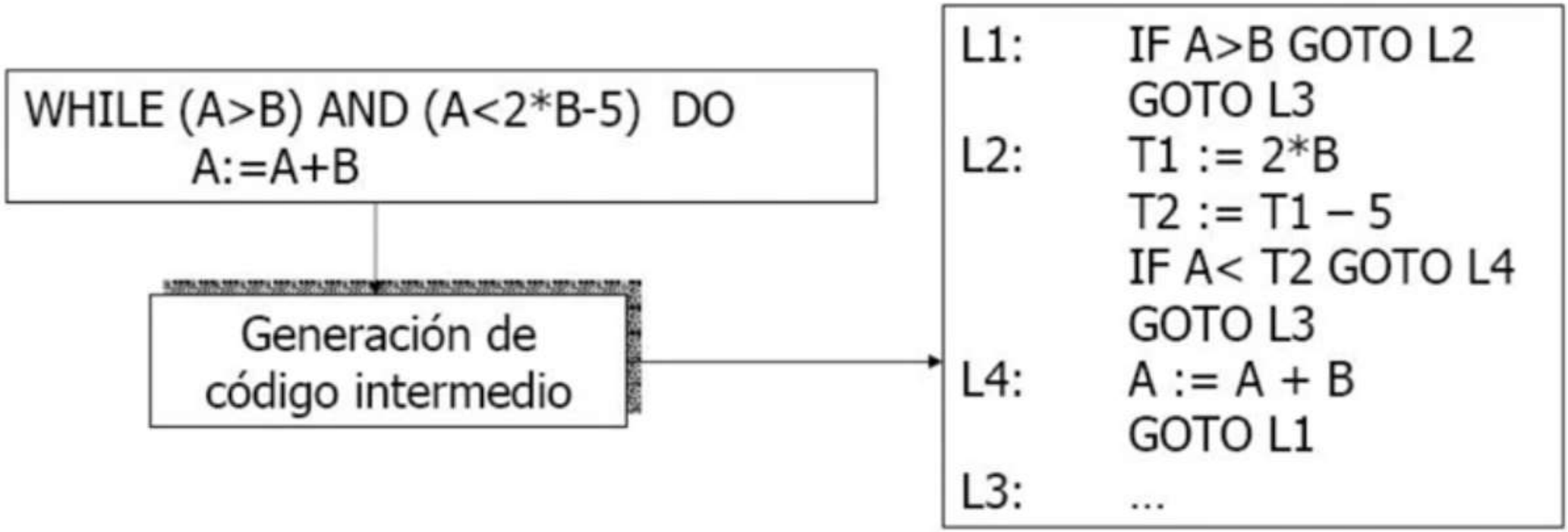
Analizador Sintáctico

EJEMPLO





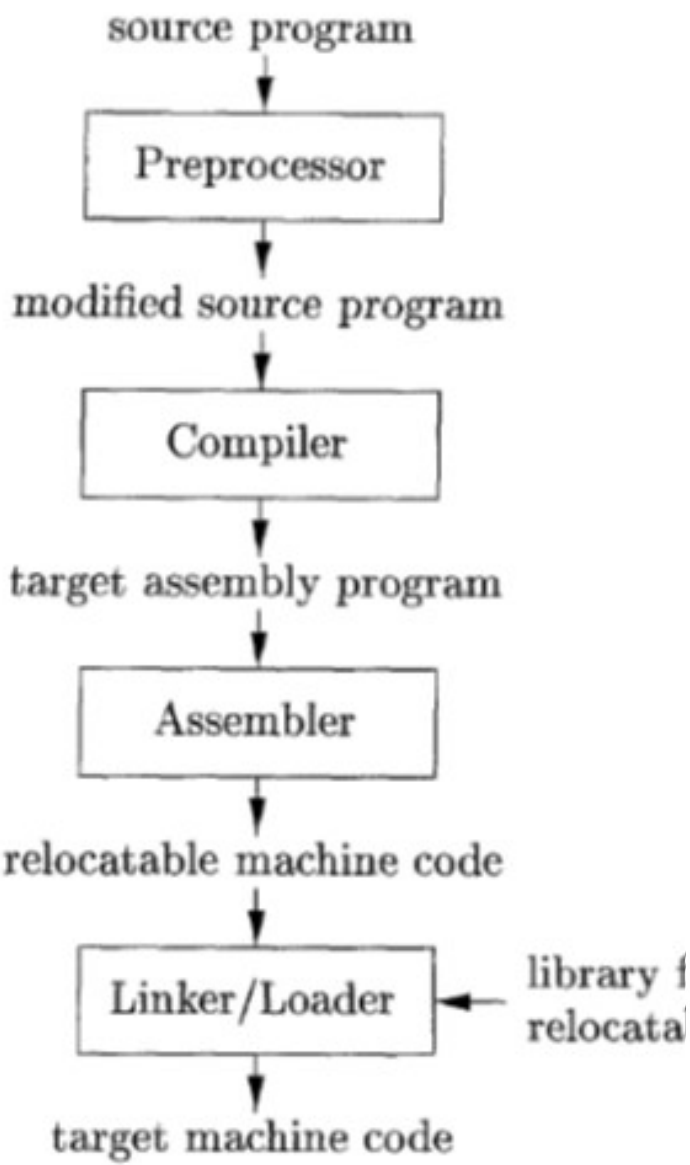
Generación de Código Intermedio



Ejemplo de Optimización de Código

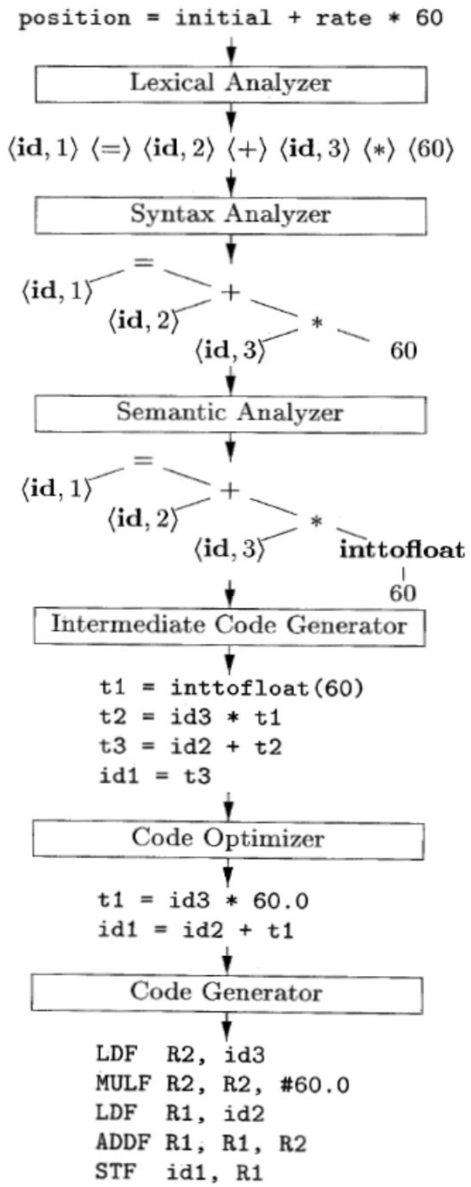
Código de tres direcciones	Código optimizado
E1: R1 := A + 3 IF R1 > B THEN GO TO E2 R2 := A + B R3 := R2 + C IF R3 > 10 THEN GO TO E2 GOTO E3 E2: A := A + 1 E3: ...	E1: R := A + 3 IF R > B THEN GO TO E2 R := A + B R := R + C IF R <= 10 THEN GO TO E3 E2: INC (A) E3: ...

Compiladores

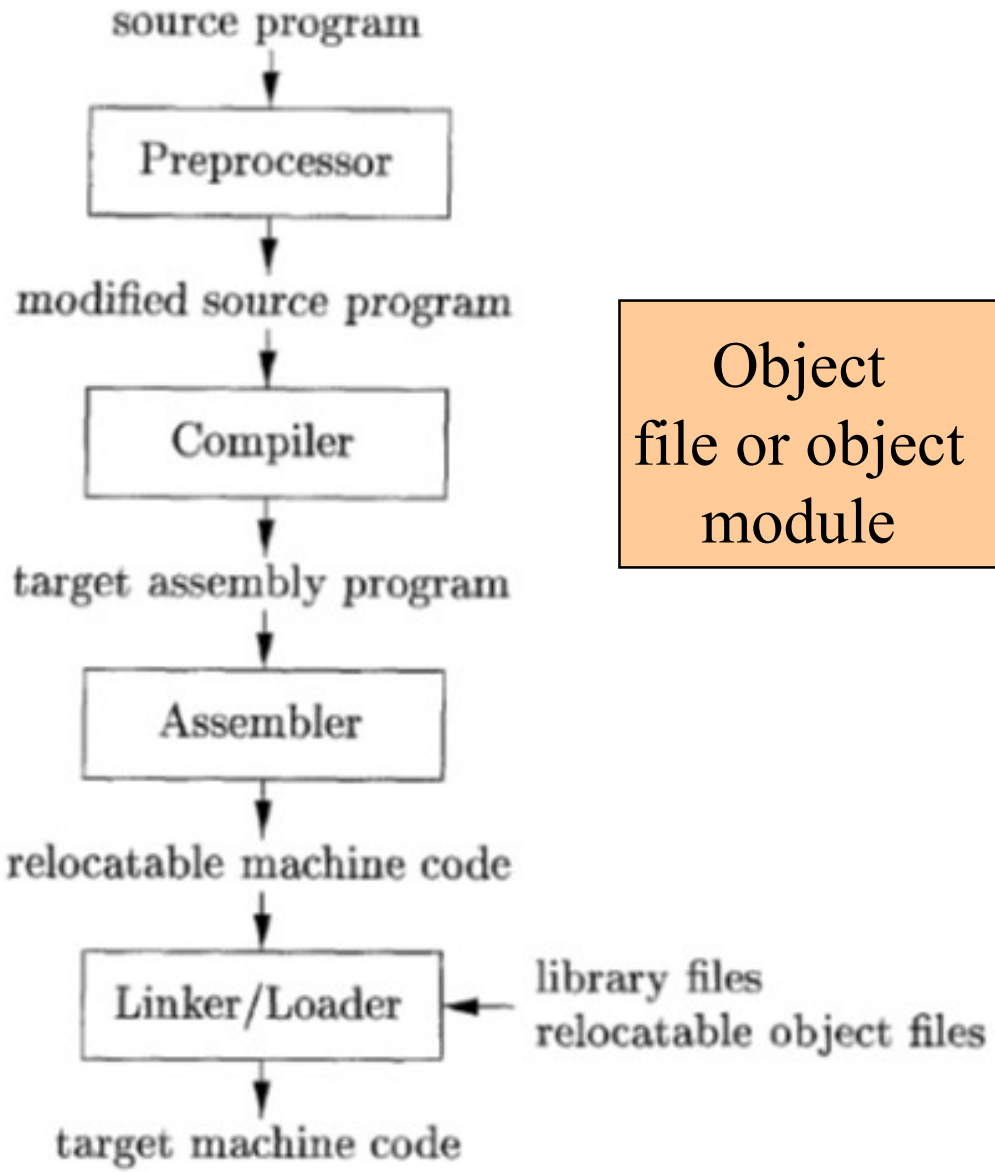


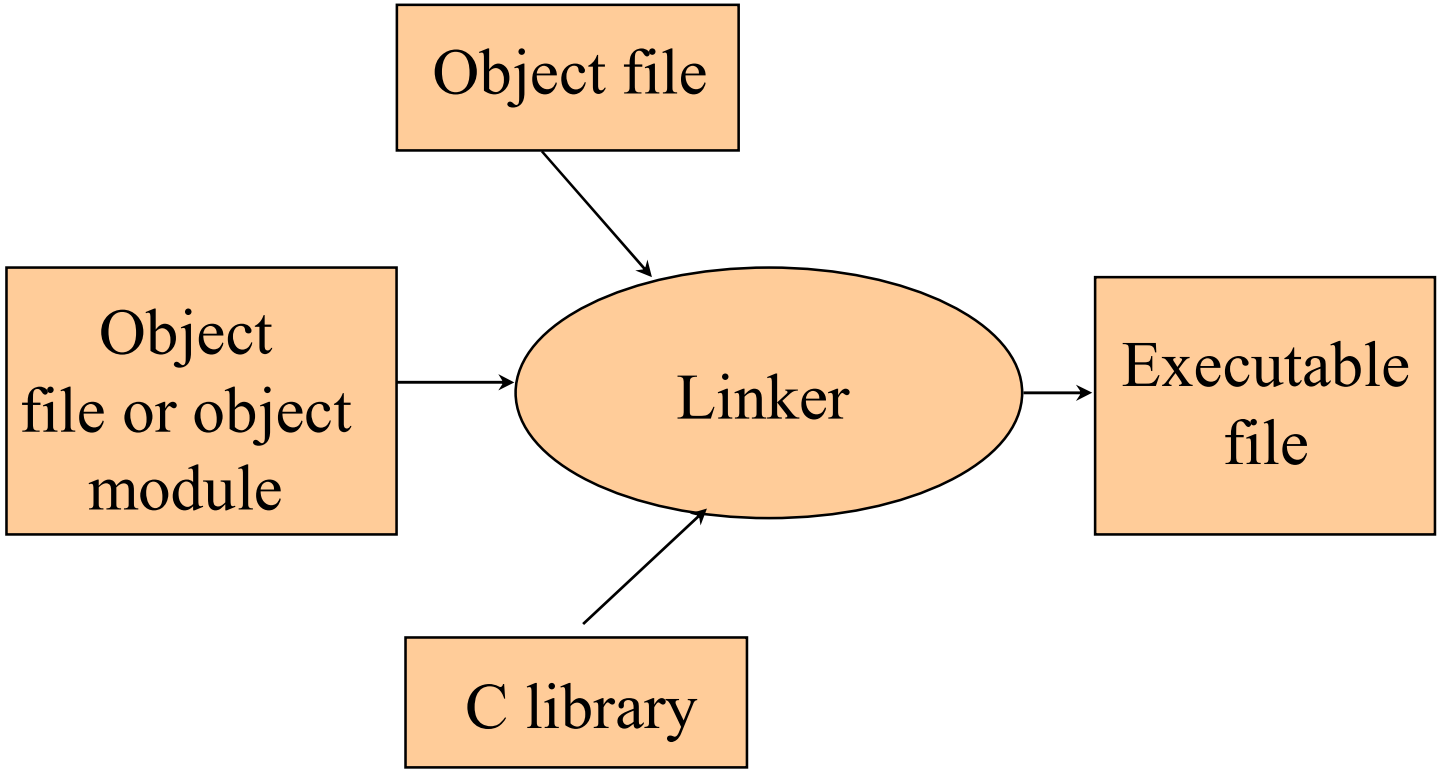
1	position	···
2	initial	···
3	rate	···

SYMBOL TABLE



Linking

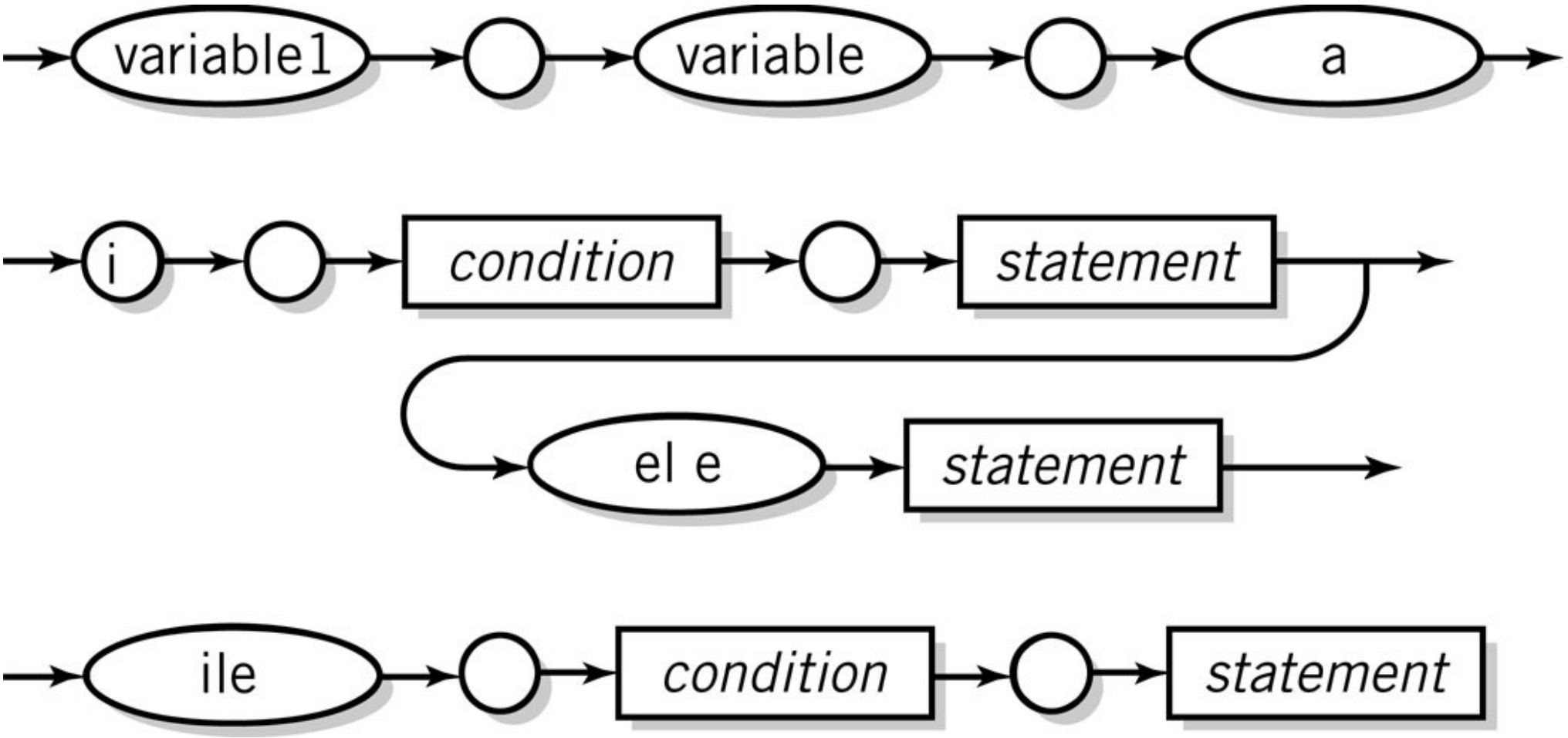

source program
Preprocessor
modified source program
Compiler
target assembly program
Assembler
relocatable machine code
Linker/Loader
library files
relocatable object files
target machine code


Object file
Object file or object module
Linker
Executable file
C library

Descripciones del lenguaje informático

- Narrative
- Syntax (Railroad) Diagrams
- BNF
  - Backus-Naur Form
  - Context-Free Grammar

Railroad Diagram



Reglas BNF típicas para Java

<compilation-unit> —>[package <qualified-identifier> ;] {import-declaration}{type-declaration} .

<qualified-identifier —> <Java-identifier> { . <Java-identifier> } .

<import-declaration —>import <qualified-identifier> [. *] .

<Java-identifier> —> <J Letter>{<J Letter>|<digit>} .

<J Letter> —> A | B | C.. | Y | Z | a | b | c.. | z | _ | $.

<digit> —> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 .

<type-declaration> –><modifier-option> <class-declaration>|<interface-declaration> .

Compilador e intérprete: diferencias

- Tanto los compiladores como los intérpretes cumplen la función de **convertir el código de software que se ha escrito a un formato ejecutable y legible por máquina**.
- Sin esta traducción, los procesadores informáticos no podrían ejecutar el software en lenguajes como C, C++, PHP, Python o Ruby, lo que convierte estos programas en unos componentes imprescindibles para utilizar ordenadores, portátiles o smartphones.
- En los apartados anteriores, hemos visto que compiladores e intérpretes presentan algunas diferencias básicas, algo que debe tenerse especialmente en cuenta a la hora de elegir un **lenguaje de programación adecuado** para desarrollar un nuevo software.
- En el **Sistema Operativo** solo representa un **Proceso** que reserva su espacio en **Memoria Principal** y trabajando en conjunto con la **Memoria Virtual**.

Compilador

- **Ventaja: autónomo y eficiente**
- Una gran ventaja de los programas que se compilan es que son unidades autónomas listas para ser ejecutadas.

- **Desventaja: específico a un hardware**
- Dado que un compilador traduce el código fuente a un lenguaje máquina específico, los programas deben ser compilados específicamente para OS X, Windows o Linux, así como para arquitecturas de 32 o 64 bits.

Compilador e intérprete

	Intérprete	Compilador
Momento en que se traduce el código fuente	Durante el tiempo de ejecución del software	Antes de ejecutar el software
Procedimiento de traducción	Línea por línea	Siempre todo el código
Presentación de errores de código	Después de cada línea	En conjunto, después de toda la compilación
Velocidad de traducción	Alta	Baja
Eficiencia de traducción	Baja	Alta
Coste de desarrollo	Bajo	Alto
Lenguajes típicos	PHP, Perl, Python, Ruby, BASIC	C, C++, Pascal

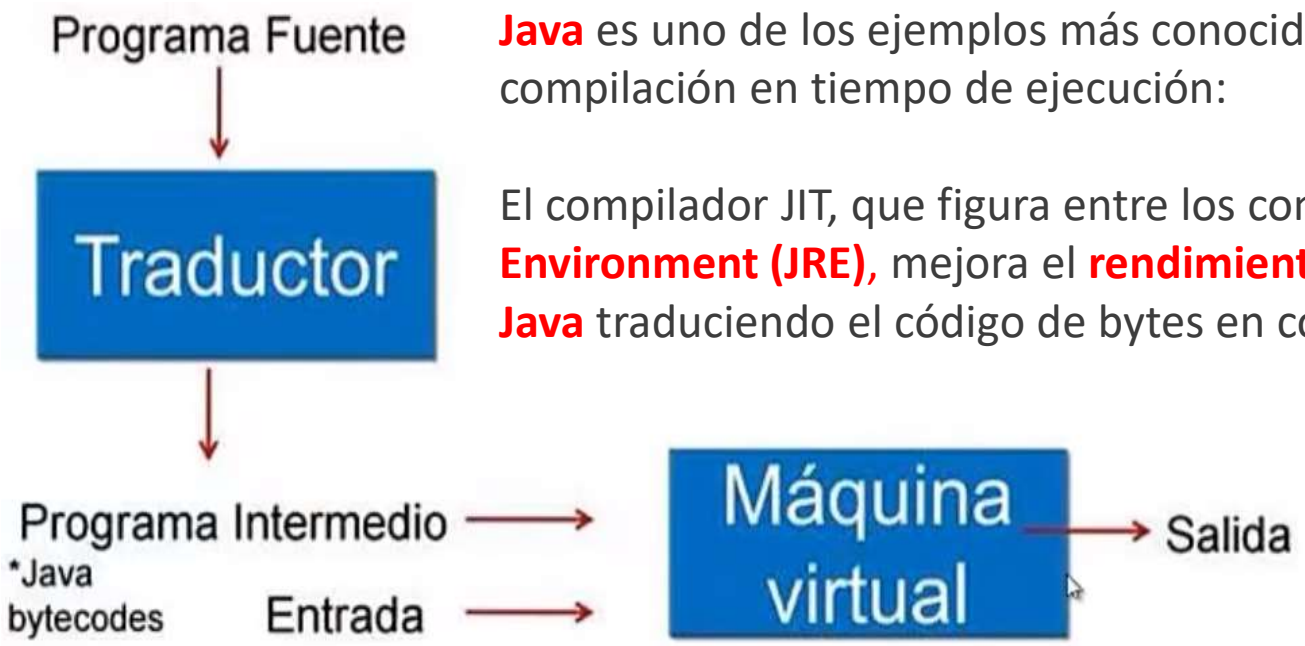
	Ventaja	Inconveniente
Intérprete	Proceso de desarrollo sencillo (sobre todo en términos de depuración)	Proceso de traducción poco eficiente y velocidad de ejecución lenta
Compilador	Proporciona al procesador el código máquina completo y listo para ejecutar	Cualquier modificación del código (resolución de errores, desarrollo del software, etc.) requiere volverlo a traducir

Compilador e intérprete

- Si observamos las diferencias entre compilador e intérprete, vemos claramente los **puntos fuertes y débiles** de cada solución para traducir el código fuente:
- Con el ***intérprete***, los programas se pueden ejecutar de inmediato y, por lo tanto, se inician mucho más rápido.
- Además, el desarrollo es mucho más fácil que con un compilador, porque el **proceso de depuración** (es decir, la corrección de errores) se lleva a cabo igual que la traducción, línea por línea.
- En el caso del ***compilador***, primero debe traducirse todo el código antes de poder resolver los errores o iniciar la aplicación.
- Sin embargo, una vez que se ejecuta el programa, los servicios del compilador ya no son necesarios, mientras que el intérprete continúa utilizando los **recursos informáticos**.

Solución híbrida de intérprete y compilador: compilación en tiempo de ejecución.

- Para compensar los puntos débiles de ambas soluciones, también existe el llamado modelo de compilación en tiempo de ejecución (en inglés, ***just-in-time-compiler***).
- Este tipo de compilador, que a veces también se conoce por el término inglés ***compreter*** (acrónimo de ***compiler*** e ***interpreter***), rompe con el modelo habitual de compilación y traduce el código del programa durante el tiempo de ejecución, al igual que el intérprete.
- De esta forma, la **alta velocidad de ejecución** típica de los compiladores se complementa con la **simplificación del proceso de desarrollo**.



Java es uno de los ejemplos más conocidos de lenguaje basado en compilación en tiempo de ejecución:

El compilador JIT, que figura entre los componentes del **Java Runtime Environment (JRE)**, mejora el **rendimiento de las aplicaciones Java** traduciendo el código de bytes en código máquina de manera dinámica.