Introducción
Traductores - Compilador

Paradigmas y Lenguajes

Evolución de los Lenguajes de programación

- 1940
  - “que funcione”
  - prelingual: previo a los lenguajes conocidos
- 1950
  - ensambladores
  - cálculo numérico (FORTRAN)
  - “explotación de la potencia de la máquina”
- 1960
  - COBOL, LISP, ALGOL, BASIC
  - estructuras de datos
  - recursividad
  - “aumento de la expresividad”
- 1970
  - reducción de la dependencia de la máquina: portabilidad
  - aumento de la correctitud de los programas
  - PASCAL, ALGOL 68, C
- 1980:
  - “reducción/manejo de la complejidad”
  - MODULA 2, ADA (introduce paralelismo(ejecutar con varios procesadores)-> programación concurrente (aplicaciones militares)), SMALLTALK(primer lenguaje de O.O. puro), MIRANDA.
- 1990
  - paralelo, distribuido(un programa es dividido en varias computadoras, división de tareas a través de la red. Ej.: cliente - servidor)
  - lenguajes visuales
- 2000
  - programación WEB/interacción entre varios lenguajes

Clasificación por Paradigmas

- Programación Imperativa
  - Programación Estructurada
- Programación Funcional
- Programación Orientada a Objetos
- Programación Lógica

Clasificación por Nivel

- Nivel de Abstracción
- Bajo: instrucciones simples. El manejo de la memoria es explícito. Ej.: ensambladores
- Alto: manejan estructuras de control que hacen menor referencia a la forma de tratamiento en la computadora. Acceso a memoria explícita.
- Muy alto: nivel de abstracción mayor. Máquinas totalmente abstractas. Máquina virtual. El acceso a memoria es implícito.

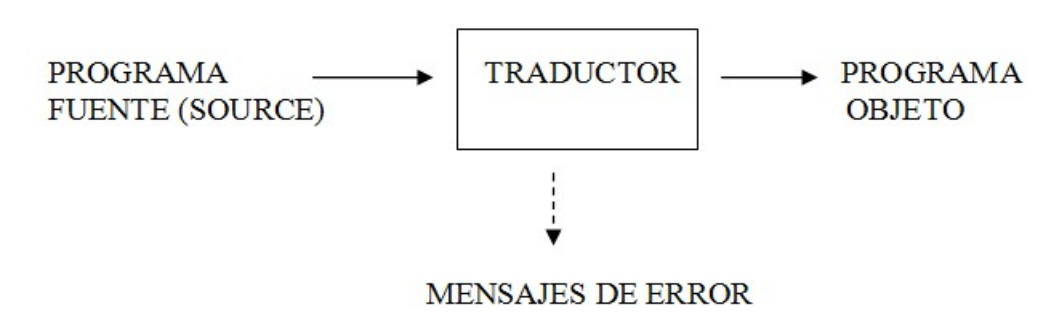
Clasificación por Generación

- **Generaciones**
- 1ª generación: lenguaje máquina
- 2ª generación: lenguaje ensamblador
- 3ª generación: lenguajes procedurales (imperativos)
- 4ª generación: lenguajes de aplicación: Ej. SQL.

  Lenguajes declarativos: especifican qué se quiere hacer y no cómo se debe hacer
- 5ª generación: técnicas de inteligencia artificial y lenguajes de inferencia
- 6ª generación: redes neuronales: simulan el comportamiento de las neuronas del cerebro humano.

Traductores

- **Traductor:** (Translator)
  - Es un programa que toma como entrada un programa escrito en un lenguaje (programa fuente) y lo convierte a un programa equivalente (en cuanto a su significado) escrito en otro lenguaje (programa objeto)



Traductores

- **Compilador**: programa que transforma un programa escrito en un lenguaje de alto nivel (por ejemplo Pascal, C, Java, Smalltalk, etc), en otro programa equivalente escrito en un lenguaje de bajo nivel (normalmente Ensamblador).
- **Ensamblador:** convierte de bajo nivel a bajo nivel (ASSEMBLER a lenguaje máquina).
- **Preprocesador**: convierte de alto nivel a alto nivel.
- **Intérprete**: convierte y ejecuta instrucción por instrucción. Ej.: BASIC, PERL, SMALLTALK, PHP

Traductores

- **Comparación entre compilador e intérprete**

Compilador
- o Mayor velocidad de ejecución del programa
- o Menor requerimientos de memoria
- o Control de errores de tipo, en tiempo de compilación
- o Una compilación, varias ejecuciones
- o Asociado a lenguajes de tipado estático

Intérprete
- o Facilidad en la depuración
- o Mayor flexibilidad: asociado a lenguajes de tipado dinámico, con alto nivel de expresividad

Traductores

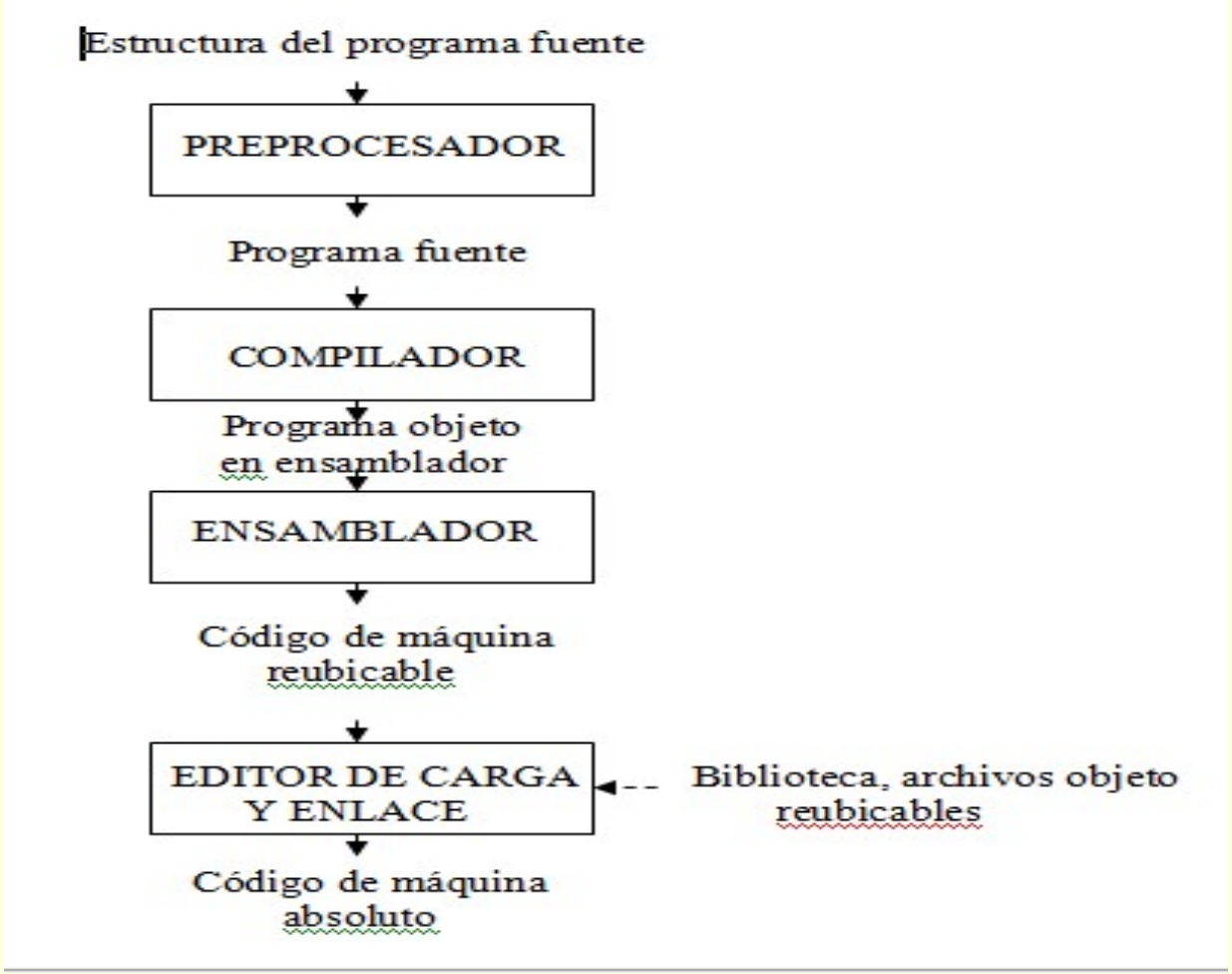
- **Objetivos de la traducción**
  - **Conversión del programa**
  - **Detección de errores**

- **Tipos de errores**
  - **Léxicos: en los componentes atómicos del lenguaje**
  - **Sintácticos: en la estructura del programa**
  - **Semánticos: en cuanto al significado del programa**
    - **de tipo / declaración**
    - **lógicos**
    - **conceptuales (aplicación de conceptos)**
    - **de comprensión del problema / requerimientos**
    - **Otros (dependientes de la arquitectura / S.O. / configuración**

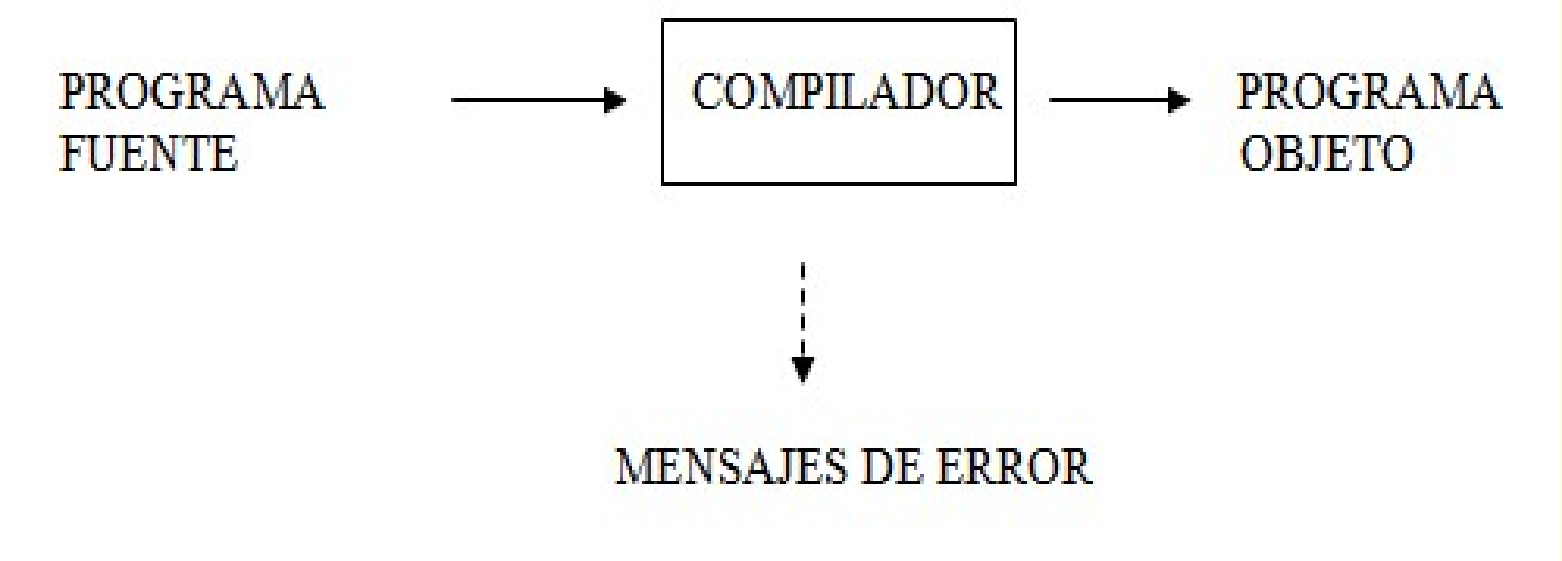
Traductores

- **Herramientas del entorno de la traducción (IDE)**
  - **Editores de estructura (inteligentes)**
  - **Impresores estéticos**
  - **Verificadores estáticos**
  - **Depuradores (debugger)**
  - **Ayuda sensible al contexto**
  - **Perfiladores (profiler)**

Sistema de procesamiento de un lenguaje

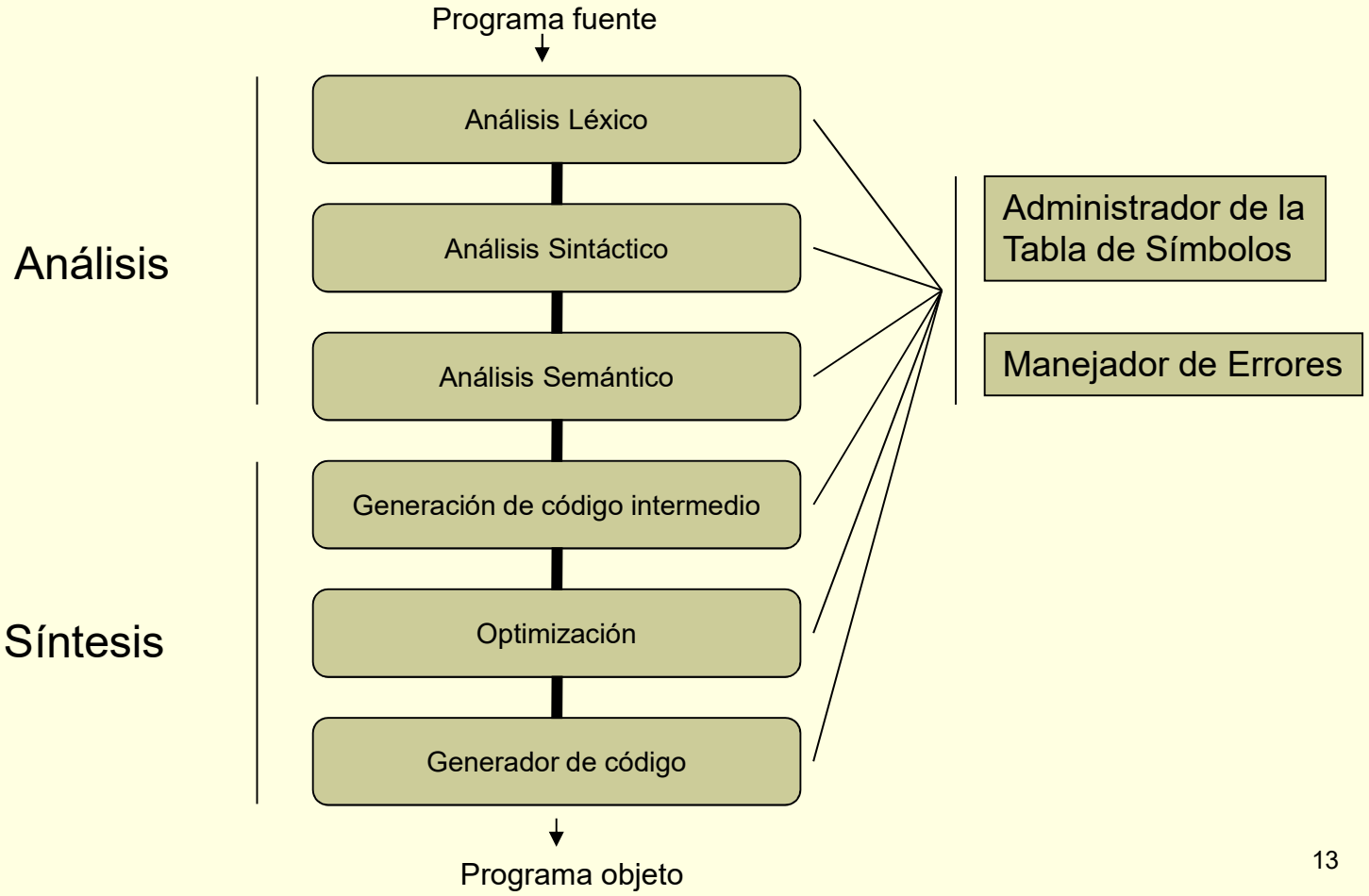


Compilación

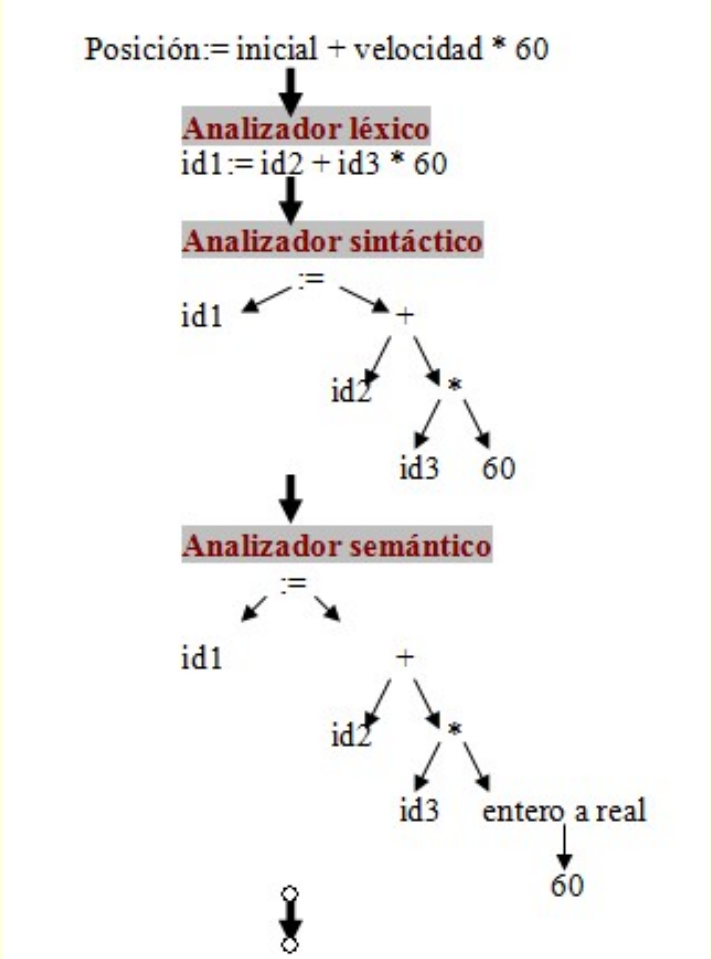


Fases de la Compilación

- Fase: división conceptual del proceso de compilación



Fases: ejemplo de Compilación



Generador de código intermedio

```
Temp1:= entero a real
Temp2:= id3 * Temp1
Temp3:=id2 + Temp2
Id1:= Temp3
```

Optimizador

```
Temp1:= id3 * 60.0
Id1:= id2 + Temp1
```

Generador de código

```
MOVF id3, R2
MULTF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

Fases

Ventajas de la división en fases:

- Reduce la complejidad del proceso
- Independencia de la arquitectura, S.O. y léxico (portabilidad, compiladores “cruzados”)
- Conocimientos, técnicas y herramientas especializadas
- Compiladores de compiladores
  - Generadores de analizadores léxicos: LEX
  - Generadores de analizadores sintácticos: YACC
  - Dispositivos de traducción

Pasada:

División física de la compilación. Compila por partes, dejando archivos con representaciones intermedias. La mayoría de los compiladores actuales son de una sola pasada.

Traductores, Compiladores

Fin