Tema 1
Introducción

1.1 Compiladores y traductores
1.2 Lenguajes de programación
1.3 Estructura de un compilador
1.4 Compiladores de varias pasadas
1.5 Entorno software del compilador
1.6 Herramientas para la construcción de compiladores
1.7 Especificación y diseño de compiladores

1.1 Compiladores y traductores

- Traductores
  - Lenguaje fuente
  - Lenguaje objeto
  - Lenguaje de implementación
- Compiladores
- Intérpretes

Traductores

- *Traducir*
- Lenguaje fuente
- Lenguaje objeto
- Lenguaje de implementación
- Notación T

Compiladores

- *Compilación*
- Lenguaje fuente
  - Lenguaje de alto nivel
- Lenguaje objeto
  - Lenguaje de bajo nivel
  - ensamblador, código máquina
- Avisos de error

Intérpretes (I)

- *Interpretar*
- Interpretar-Ejecutar
- La interpretación conlleva línea a línea:
  - Reconocimiento
  - Traducción
  - Ejecución

Intérpretes (II)

- Interpretación directa de las instrucciones de alto nivel.
- Interpretación en dos fases:
  - Reconocimiento y traducción a código intermedio.
  - Interpretación/ejecución del código intermedio.

1.2 Lenguajes de programación

- Tipos
- Evolución histórica
- Ecología de los lenguajes de programación
- El mejor
- Ideas
- Tendencias actuales

Tipos de lenguajes de programación

- Lenguajes Imperativos
- Lenguajes Funcionales
- Lenguajes Lógicos
- Lenguajes Orientados a Objetos
- Lenguajes Orientados a Eventos

- Lenguajes procedurales: ¿Cómo traducir?
- Lenguajes declarativos: ¿Qué hacer?

Evolución histórica (I)

- Lenguajes máquina y ensambladores: Nivel más bajo de abstracción.
- El primer compilador fue escrito por Grace Hopper, en 1952.
- FORTRAN: Los procedimientos no pueden ser recursivos.
- COBOL
- ALGOL: Incorpora gestión dinámica de memoria y recursividad.

Evolución histórica (II)

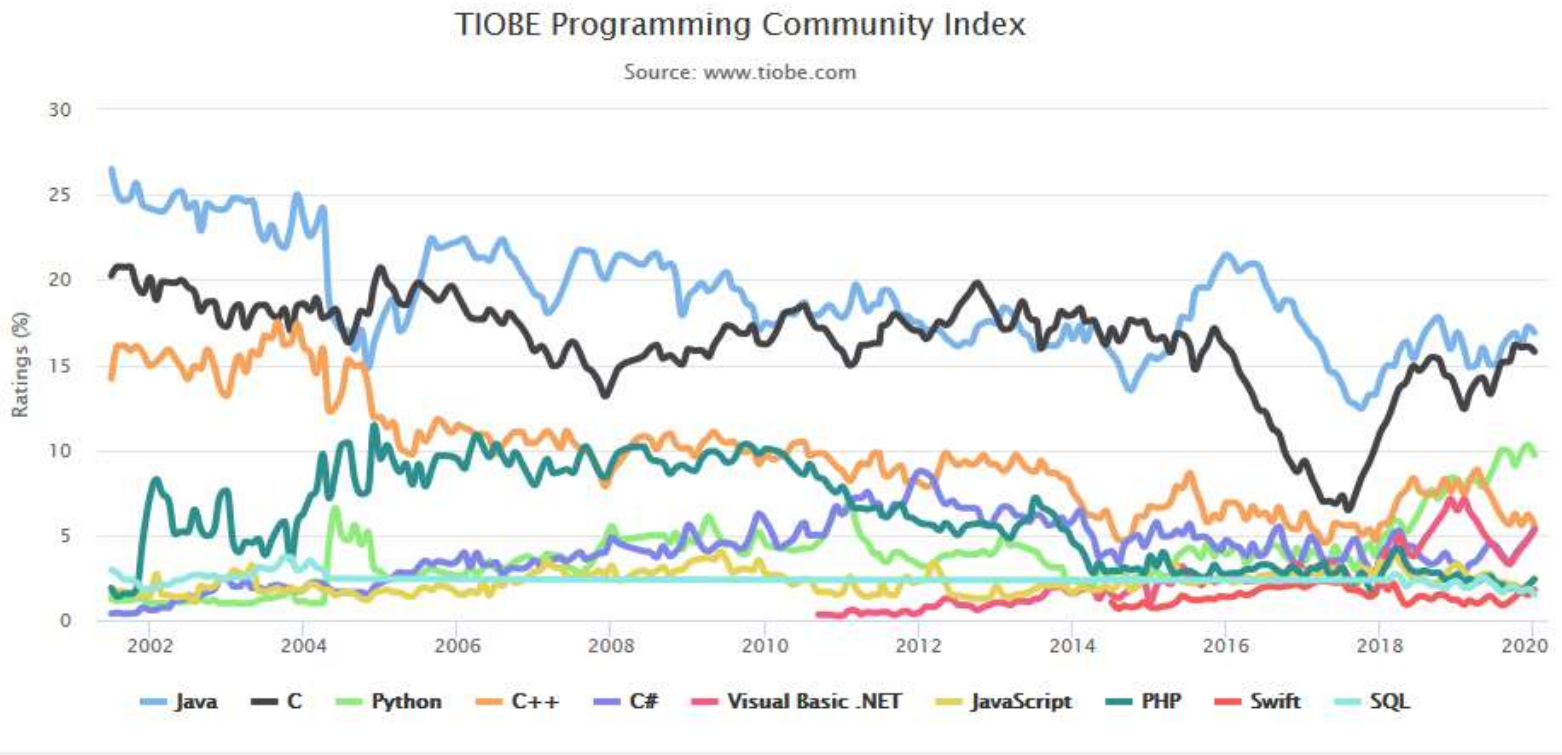
- Lenguajes de propósito general
  - Programación Estructurada
  - Orientados a objetos
  - Pascal → Modula-2 → Ada
  - Lenguaje C → C++ → Java → C#
- Lenguajes Especializados
  - Lisp: tratamientos tipos lineales y arborescentes.
  - Prolog: programación lógica
  - Php: interfaces web
  - ...

Evolución histórica (III)

www.levenez.com/lang/
https://githut.info/

Estimación de uso

http://www.tiobe.com/ **THE tiobe INDEX graph from 2002 to 2020**



¿Por qué hay cambios?

- Mejoran los lenguajes:
  - Reducir del tiempo de programación.
  - Facilitar (o posibilitar) un tipo de aplicación.

- Adiestramiento de programadores muy caro
  - Los lenguajes clásicos, muy extendidos, cambian poco.

    Se busca productividad alta con poco entrenamiento.

¿Por qué hay tantos lenguajes de programación?

- Preparados para diferentes dominios.
  - Necesidades diferentes, contradictorias
- Ejemplos
  - Gran escala: velocidad / memoria / concurrencia
  - Cálculo científico: velocidad / memoria
  - Empresas: generación de informes / persistencia
  - Programación de sistemas: control de bajo nivel / tiempo real
  - Aplicaciones Web, móviles, ...

¿Cuál es el mejor?

- No hay una métrica aceptada universalmente para saberlo.

- Criterios comúnmente establecidos:
  - Los programas que se obtienen son:
    - Fáciles de leer y escribir
    - Fiables y seguros

¿Cuál es el mejor?

- Se comprueban sobre estas características:
  - **Sencillez**:
    - Puede ser sencillo de escribir pero complejo de leer/entender.
    - La sobrecarga puede complicar la comprensión.
  - Las **estructuras de control** y **tipos de datos**:
    - Facilitan la legibilidad y escritura. Ej. Booleanos
    - Eliminar los gotos lo hace más fiable.
  - La **sintaxis** cercana al lenguaje natural o matemático.
  - Un **nivel de abstracción** elevado y la **capacidad de expresión** facilita la escritura y la fiabilidad
  - La **comprobación de tipos** y la **gestión de excepciones** hace que un lenguaje sea más fiable.

El mejor

Characteristic	Criteria		
	Readability	Writeability	Reliability
Simplicity	*	*	*
Data types	*	*	*
Syntax design	*	*	*
Abstraction		*	*
Expressivity		*	*
Type checking			*
Exception handling			*

Historia de las ideas: abstracción

- **Abstracción** = lejos de implementaciones concretas.
  - LP, compiladores:
    - Código de alto nivel, independiente de la máquina.
  - Subrutinas:
    - Comportamiento abstracto.
  - Módulos:
    - Exportar interface, ocultar implementación.
  - Tipos de datos abstractos:
    - Unir datos y operaciones .

Historia de las ideas: tipos

- Los tipos ayudan
  - Encontrar automáticamente un conjunto amplio de errores
  - Lenguajes tipados "seguros"

Historia de las ideas: reutilización

- Reutilizar = explotar patrones comunes
  - Producción masiva de algunos componentes de software (estructura de datos muy utilizadas, árboles, tablas asociativas, ...)
- Dos modos:
  - Parametrización de tipos: List(string)
  - Clases y herencia
    - Especialización de una abstracción
    - Cambio de comportamiento

Tendencias actuales

- Diseño de lenguajes:
  - Muchos lenguajes para aplicaciones especiales.
  - Los lenguajes más usados permanecerán.
- Compiladores
  - Necesidades mayores, más complejos.
  - Mayor rango que cubrir
    - Lenguajes nuevos
    - Arquitecturas nuevas
- Optimización vs. portabilidad
  - Compilación vs. interpretación
  - Compilación Just In Time

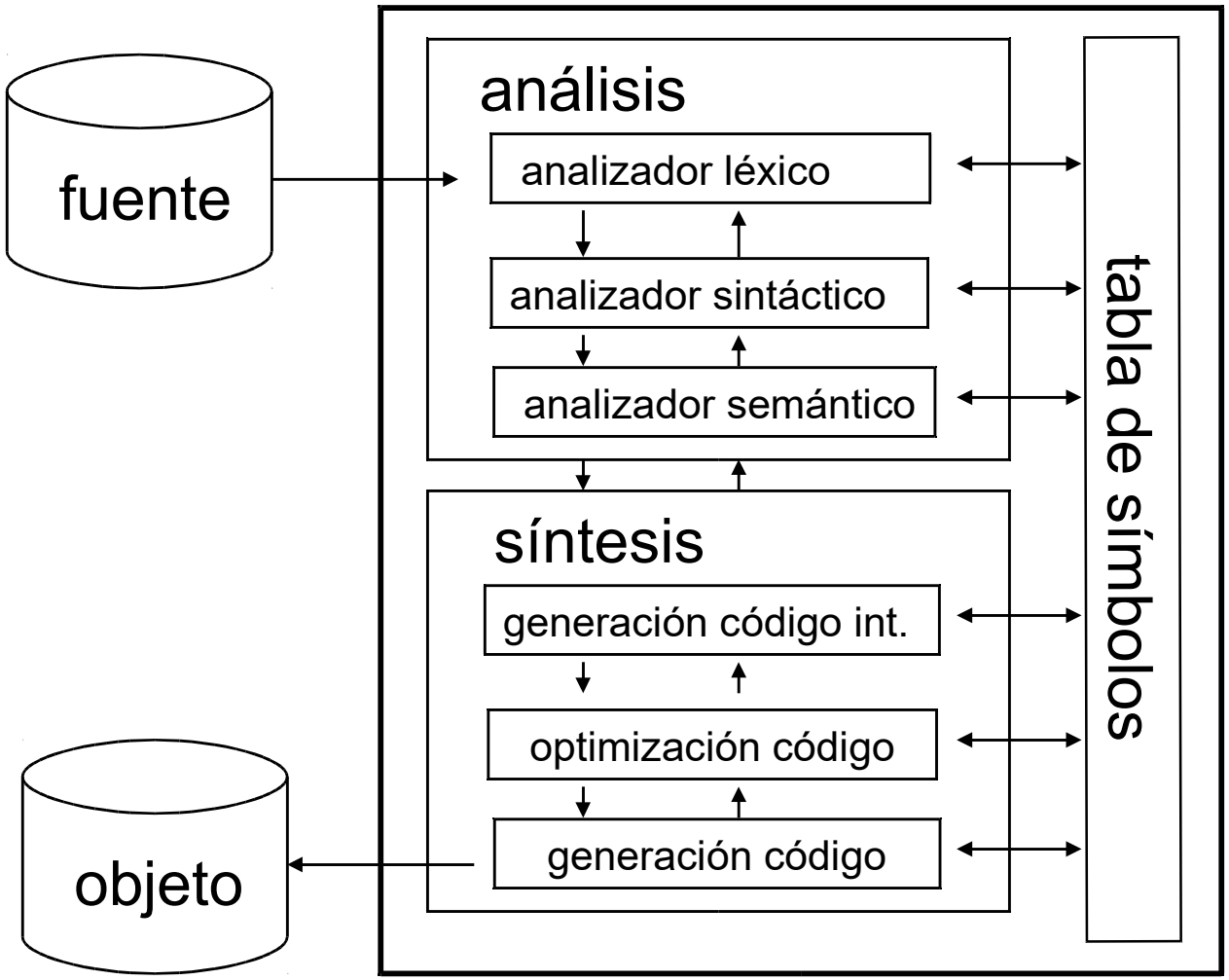
¿Por qué estudiar compilación?

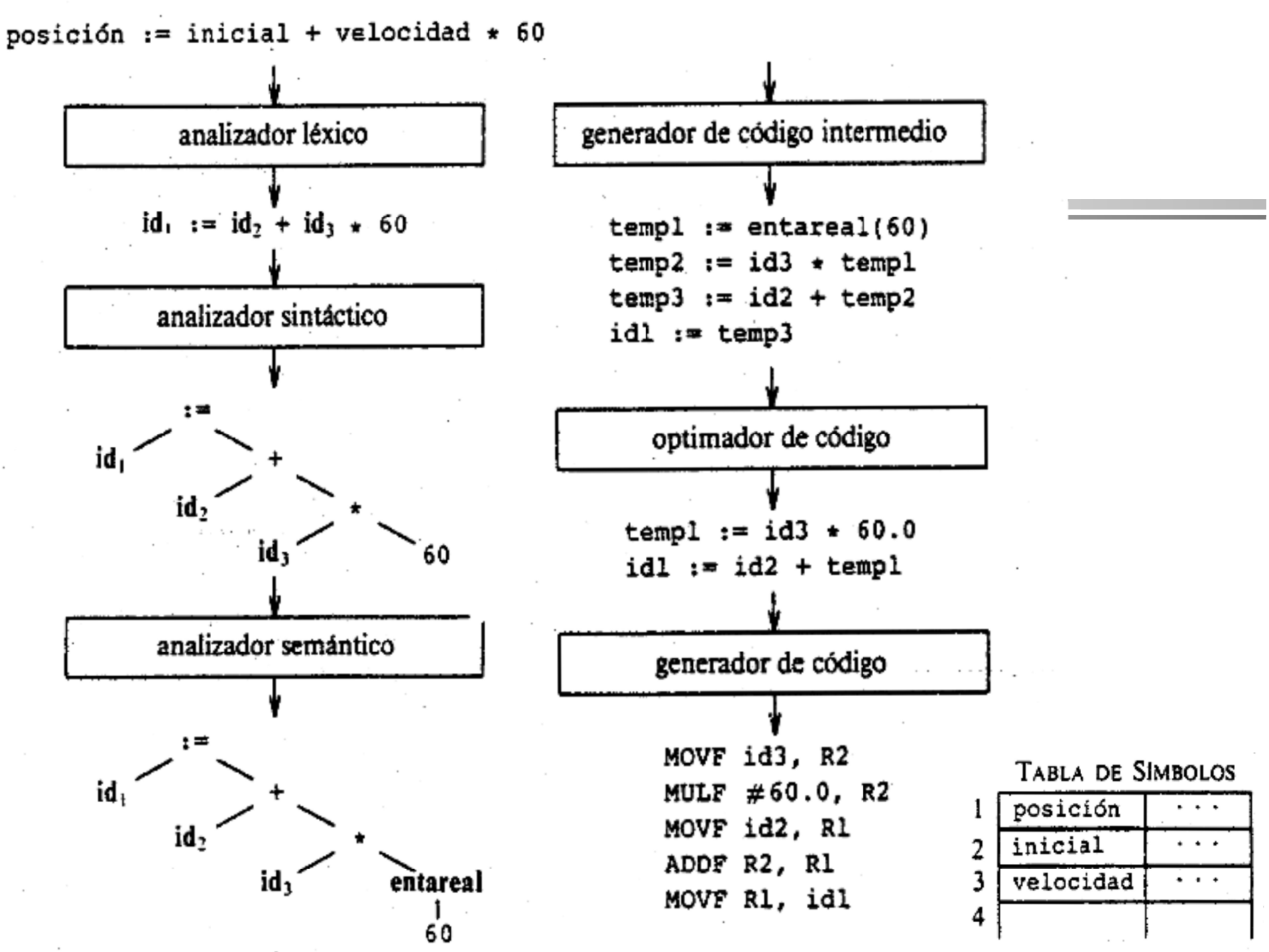
5. Aumentar las aptitudes de programación
4. Entender mejor el comportamiento de los programas
3. Facilidad para aprender nuevos lenguajes
2. Aprender a construir un sistema grande y fiable
1. Ver muchos conceptos de informática en marcha

1.3 Estructura de un compilador

- análisis / síntesis
- análisis léxico
- análisis sintáctico
- análisis semántico
- tratamiento de errores
- manejo de la tabla de símbolos
- generación de código intermedio
- optimización de código
- generación de código
- front-end / back-end

Estructura de un compilador




posición := inicial + velocidad * 60
analizador léxico
id1 := id2 + id3 * 60
analizador sintáctico
:=
id1
+
id2
*
id3
60
analizador semántico
:=
id1
+
id2
*
id3
entareal
60
generador de código intermedio
temp1 := entareal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
optimador de código
temp1 := id3 * 60.0
id1 := id2 + temp1
generador de código
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
TABLA DE SIMBOLOS
1 posición ...
2 inicial ...
3 velocidad ...
4

1.4 Compiladores de varias pasadas

- Separa la tarea de compilación en varias pasadas.
- Es común agrupar varias fases en una pasada, y entrelazar la actividad de estas fases durante la pasada.
- Separar la tarea en pasadas tiene ventajas e inconvenientes.

1.5 Entorno software del compilador

- Preprocesadores
- Ensambladores
- Linkeditores y cargadores
  - Librerías
- Decompiladores
- Entornos de programación
  - Entornos de edición
  - Entornos de documentación
  - Depuradores

1.6 Herramientas para la construcción de compiladores

- Generación de analizadores y traductores
  - de analizadores léxicos
  - de analizadores sintácticos
    - de "evaluadores de ETDSs"
- Herramientas para trabajar con árboles sintácticos abstractos
- Generación automática de código