Universidad Europea
LAUREATE INTERNATIONAL UNIVERSITIES

Introducción

Compiladores y Lenguajes Formales

Universidad Europea
LAUREATE INTERNATIONAL UNIVERSITIES

INDICE

1. ¿Por qué son necesarios los compiladores?
2. ¿Cómo nos independizamos de la máquina?
3. Definición de Compilador
4. Tipos de compiladores
5. Fases de un compilador
6. Lenguajes de programación
7. Como se diseña un compilador
8. Motivaciones y aplicaciones

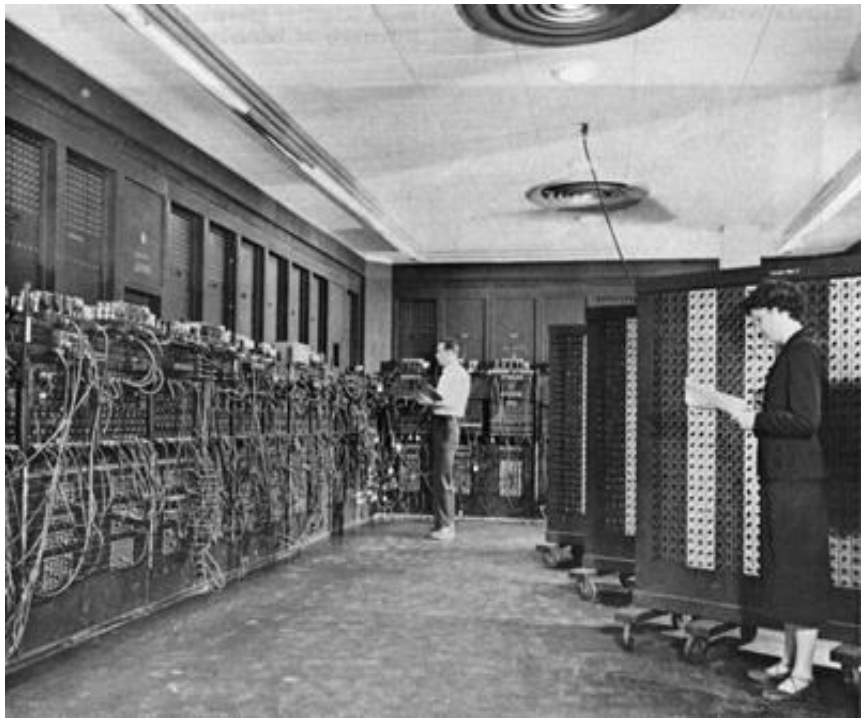
¿Por qué son necesarios los compiladores?

- Llegada de John Von Neumann al proyecto ENIAC (1945) → cablear el ordenador para cada nueva tarea.

• Decide resolver este problema, consiguiéndolo en 1949, escribiendo secuencias de código o programas que hacen que estos ordenadores realizaran los cálculos deseados.

• Estos programas se escribían utilizando códigos numéricos que representaban las operaciones que se iban a realizar.

• El lenguaje utilizado para construir estos programas se denominó lenguaje máquina porque estaba totalmente relacionado con el hardware de la máquina.



¿Por qué son necesarios los compiladores?

- Como se puede ver en el siguiente ejemplo, esta forma de escribir un programa es difícil y tediosa por lo que pronto se dio nombre a los códigos de las operaciones y a las direcciones de memoria
- Un ejemplo de este código máquina es el siguiente:

	CODIGO OPERACIÓN	DIRECCIÓN
CODIGO MÁQUINA	00010101	10000011
ENSAMBLADOR	LOAD	X

- Con el lenguaje ensamblador se mejoró enormemente la velocidad y exactitud con la que se escribían los programas, de hecho todavía se encuentra en uso en situaciones donde se necesita velocidad y se tiene poco espacio para el código

¿Por qué son necesarios los compiladores?

- **Desventajas del lenguaje ensamblador**:
  - No es fácil de escribir
  - Es difícil de leer y entender
  - Depende de la máquina para la que se ha escrito, lo que implica reescribirlo si se va a ejecutar en otra máquina.

SEGUIMOS UNIDOS A LA MÁQUINA O PROCESADOR

¿Cómo nos independizamos de la máquina?

- Generando un código intermedio, denominado código objeto → ¿qué ganamos? El no tener que recordar las direcciones de memoria, así como otros aspectos totalmente ligados con la máquina. Esto se pensaba que no iba a ser fácil, además de poco eficiente.

	CODIGO OPERACIÓN	DIRECCIÓN
ENSAMBLADOR	MOV X, 5	10000011
CÓDIGO	X = 5	X

- Había que conseguir crear un lenguaje que nos permitiera realizar acciones de tal forma que fuera fácil de manejar y aprender por una persona. Este tipo de lenguajes se denominó de **alto nivel**, en contraposición con los de **bajo nivel** (código máquina y ensamblador) totalmente dependientes de la máquina.

¿Cómo nos independizamos de la máquina?

- De esta forma se llegó al que se denomina el primer compilador
  - Grace Hopper, el A-0. Se empezó a trabajar en él en el otoño de 1951 y la primera rutina “compilada” se probó con éxito en un UNIVAC en la primavera de 1952 (Hopper & Mauchly, 1953).
  - El concepto de compilador de estos momentos consistía en:
    - Aceptación del pseudocódigo
    - Su decodificación
    - La búsqueda de la subrutina apropiada
    - La asignación de las direcciones de memoria a las subrutinas en el programa,
    - La selección e inserción de las posiciones de memoria de los argumentos y resultados
    - La organización de la transferencia del control y la escritura del programa terminado en una cinta.
- Esto era mas un **cargador o enlazador**

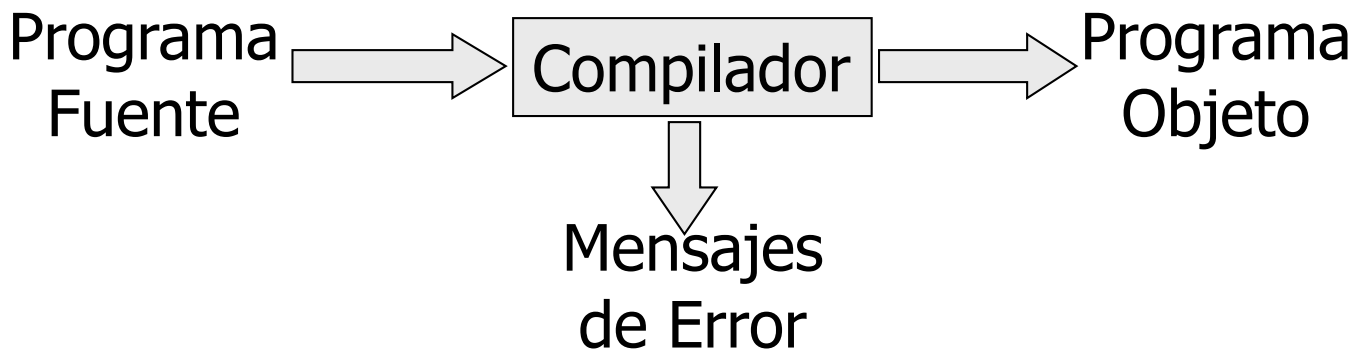
¿Cómo nos independizamos de la máquina?

- El A-0 de Grace Hooper era mas un **cargador o enlazador** que el concepto actual de compilador, pero fue la primera vez que se usó la palabra **compiler.**
- Por otro lado, en 1954, **John Backus** comenzó el desarrollo de un compilador de FORTRAN para IBM, concretamente para el IBM 704, que le llevó dos años y medio y 18 hombres para realizarlo. Consistió en dos componentes:
  - el lenguaje FORTRAN y
  - el traductor para el IBM 704.

- Este desarrollo fué un verdadero compilador tal y como lo entendemos actualmente

Compilador. DEFINICIÓN

- Proceso de traducción que convierte un programa fuente escrito en un lenguaje de alto nivel a un programa objeto en código máquina y listo por tanto para ejecutarse en el ordenador, con poca o ninguna preparación adicional



Herramientas necesarias para Compiladores

- **Los procesadores de lenguaje, e**s el nombre genérico que reciben todas las aplicaciones informáticas en las cuales uno de los datos fundamentales de entrada es un **lenguaje. Esto engloba a las herramientas necesarias para un compilador:**

  - **Editores (editors)**
  - **Traductores (translators)**
  - **Compiladores (compilers)**
  - **Intérpretes (interpreter)**
  - **Ensambladores (assemblers)**
  - **Montadores de enlace o enlazadores (linkers)**
  - **Cargadores (loaders)**
  - **Desensambladores (dissemblers)**
  - **Decompiladores (decompilers)**
  - **Depuradores (debuggers)**
  - **Analizadores de rendimiento (profilers)**
  - **Optimizadotes de código (code optimizers)**
  - **Compresores (compressors)**

Herramientas necesarias para Compiladores

- **Editores:** Son herramientas software que se utilizan para leer y escribir los programas que posteriormente el compilador traducirá a código máquina. Actualmente, se integran con los compiladores (.NET) constituyendo un IDE (entorno de desarrollo interactivo), aunque se pueden utilizar por separado (Eclipse). Para que los ficheros que generan se puedan leer por cualquier editor es necesario que tengan un formato estándar, concretamente ASCII.
- **Traductores:** Como su propio nombre indica, traducen o convierten un programa escrito en un lenguaje (lenguaje fuente) a otro lenguaje (lenguaje objeto). El lenguaje fuente puede ser cualquier lenguaje, normalmente de alto nivel como Visual Basic, Java, COBOL, C++, C#, etc. y el lenguaje objeto puede ser desde otro lenguaje de alto nivel o bien ensamblador o código máquina.
- **Intérpretes:** Es un traductor de lenguaje, pero que a diferencia de un compilador, el código que genera se ejecuta al mismo tiempo que se va traduciendo y no genera como en el caso del compilador un fichero ejecutable. Los lenguajes típicos que utilizan intérprete en lugar de compilador son BASIC y LISP.

Herramientas necesarias para Compiladores

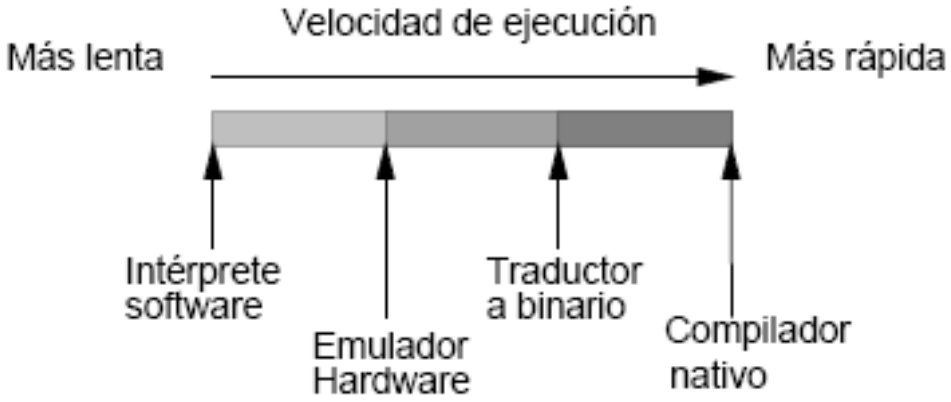
- **Ensambladores:** Es un traductor para una máquina en particular. Traduce los nombres simbólicos que tienen las instrucciones (Ej: **LOAD**) a cada una de las instrucciones en código binario (**00010101)** que entiende la máquina para la que está diseñado.
- **Preprocesadores:** Es un programa que funciona de forma independiente una vez que el compilador lo llama, antes de comenzar el proceso de compilación. Las tareas que realiza un preprocesador son la sustitución de las macros por las sentencias que las componen, la inclusión de otros archivos o librerías o la eliminación de comentarios.
- **Montadores de enlace:** Puesto que muchos programas se compilan en momentos diferentes, nos hace falta un montador de enlace o enlazador que sea capaz de unir los diferentes módulos con sus respectivos códigos objeto para producir un fichero ejecutable. También sirve para enlazar el código objeto con las funciones que necesita de una librería de funciones u otros módulos, además de con la memoria o los dispositivos de entrada y salida utilizando para ello llamadas a funciones del sistema operativo.

Herramientas necesarias para Compiladores

- **Cargadores:** Esta herramienta software se encarga asignar las direcciones y el espacio de memoria necesario para la ejecución del programa. Esto se puede realizar porque el montador de enlace le proporciona direcciones de memoria relativas, denominándose este tipo de código reubicable o relocalizable, y el cargador le asigna posiciones reales o absolutas. El cargador es normalmente una herramienta que trae el sistema operativo, en el caso de Linux se denomina ld (/lib/ld-linux.so) y en el del antiguo MS-DOS, el cargador se encontraba dentro del intérprete de comandos COMMAND.COM (Cueva, 1998).
- **Depuradores:** Son herramientas que nos permiten encontrar los errores y solucionarlos en un programa, una vez éste ha sido compilado. Normalmente permiten seleccionar variables y visualizar los valores que van teniendo a partir de una ejecución paso a paso
- **Desensambladores:** Es una herramienta que traduce el lenguaje máquina a lenguaje ensamblador. Se utiliza para realizar ingeniería inversa.

Herramientas necesarias para Compiladores

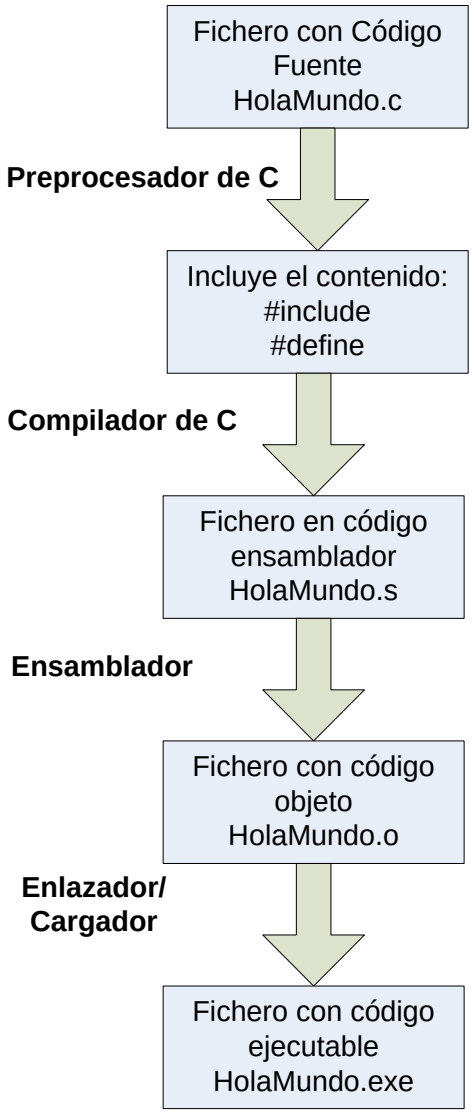
- Solución:



- Intérpretes software o emuladores software:
  - Es un programa de código binario que lee una a una las instrucciones binarias de la arquitectura antigua que están en un fichero ejecutable y las interpreta.
  - Ej: Los intérpretes de la máquina abstracta JVM y que permite ejecutar los códigos binarios (bytecodes → .class)

Proceso de Compilación

Para la creación del fichero de código fuente utilizamos un **editor**, como por ejemplo Eclipse, que incorpora un depurador para ayudarnos a encontrar errores



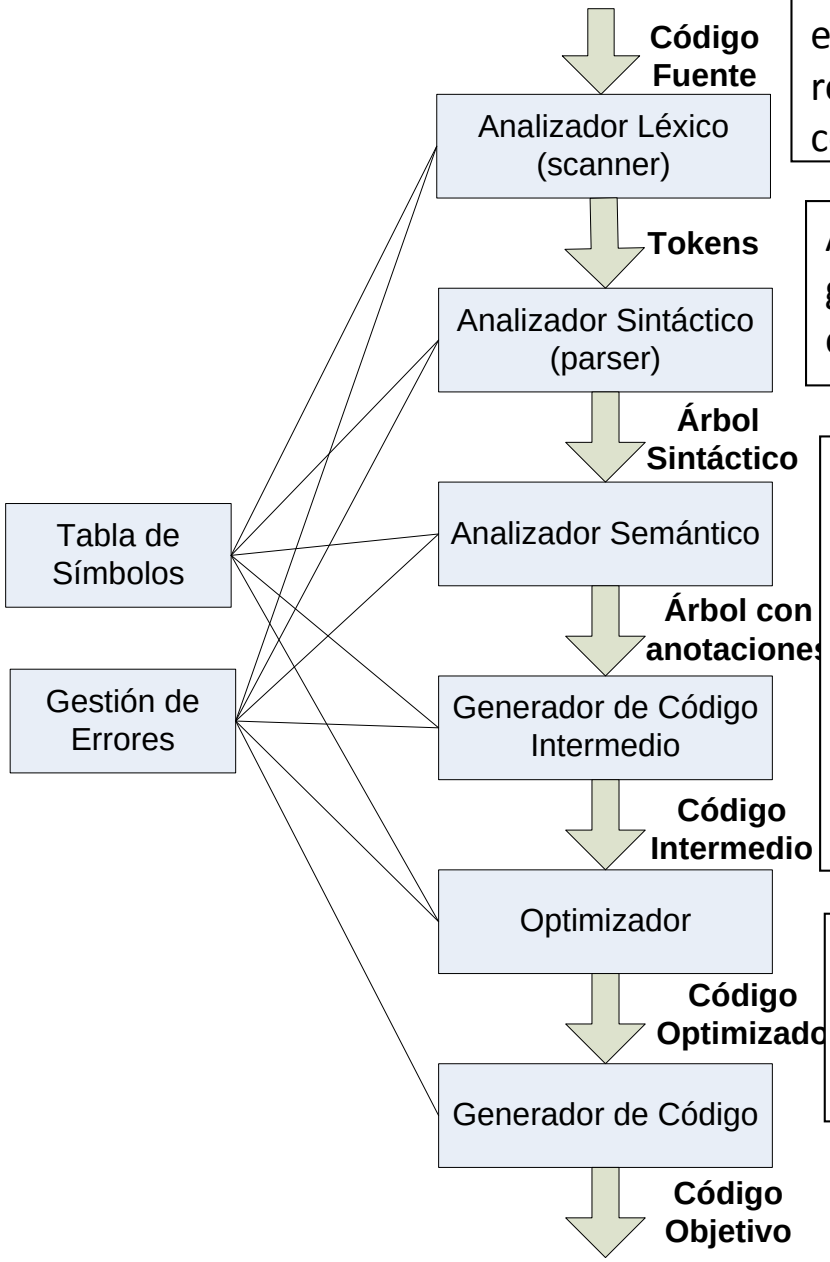
El **preprocesador** de C (cpp) incluye el contenido de un fichero (Ej: #include <stdio.h>) a través de las directivas #include, #define o # if

Traduce el código fuente una vez lo ha analizado a código ensamblador. Este proceso lo explicaremos detalladamente en la siguiente pantalla.

Una vez ha procesado el ensamblador del procesador que tenemos en nuestro ordenador, se genera el **código objeto** que ya es código binario.

Se **enlaza** el código objeto con otros ficheros que forman parte de nuestro programa (el enlazador facilita una compilación separada) y **el cargador** le proporciona direcciones de memoria relativas.

Fases de la Compilación



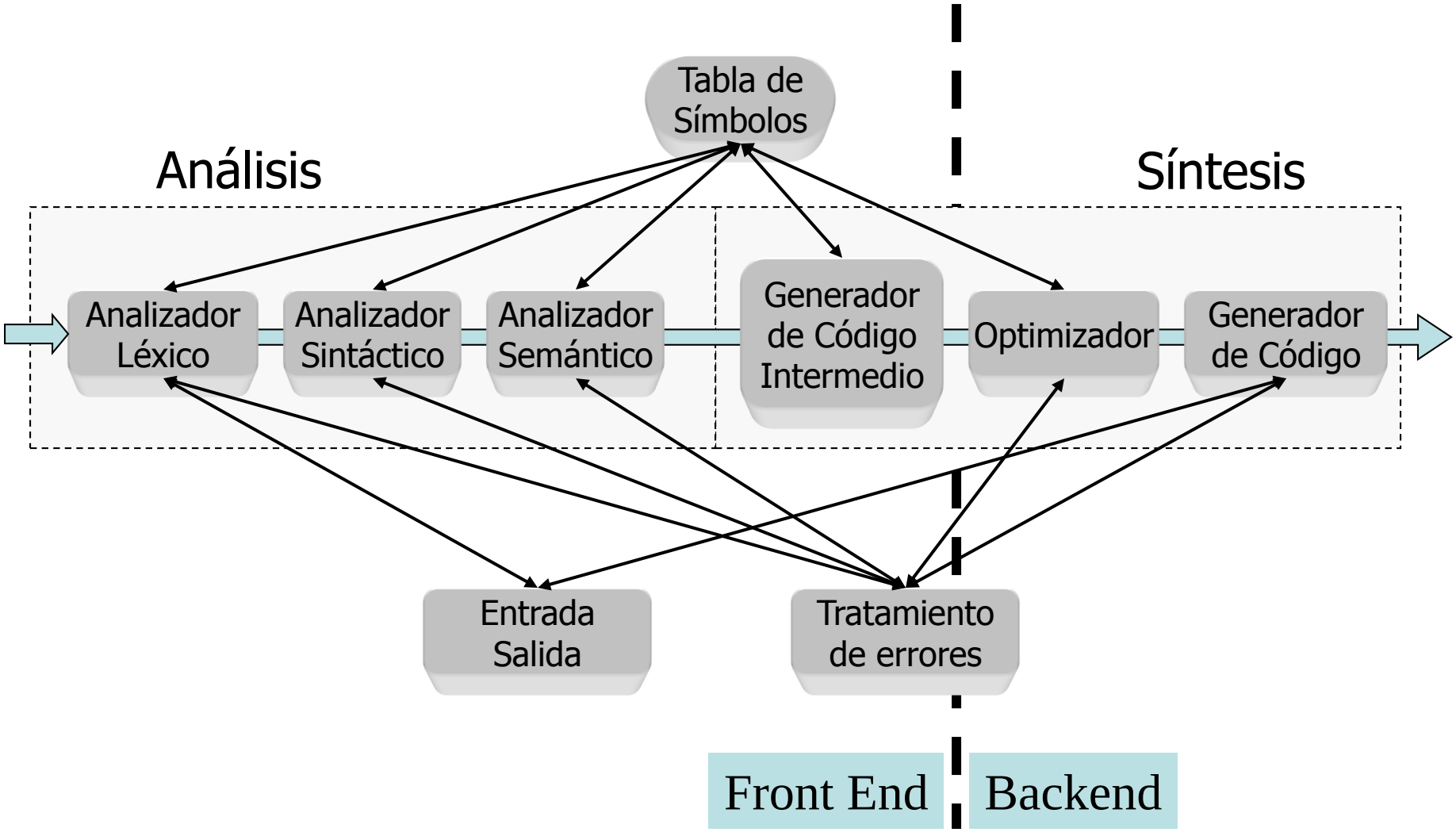
Se comienza a leer la cadena de entrada y los caracteres que se reconocen se agrupan en componentes léxicos (tokens)

Agrupa los tokens en frases que la gramática del lenguaje reconoce construyendo un árbol sintáctico

El análisis semántico identifica el comportamiento, busca errores semánticos, verifica los tipos, determina operadores y operandos. Para ello utiliza atributos que se añaden al árbol, construyendo así el árbol con anotaciones (o decorado).

Se intenta mejorar el código intermedio, y por tanto el código objetivo mediante distintas técnicas.

Fases de la Compilación



Fases de la Compilación . Análisis del programa fuente

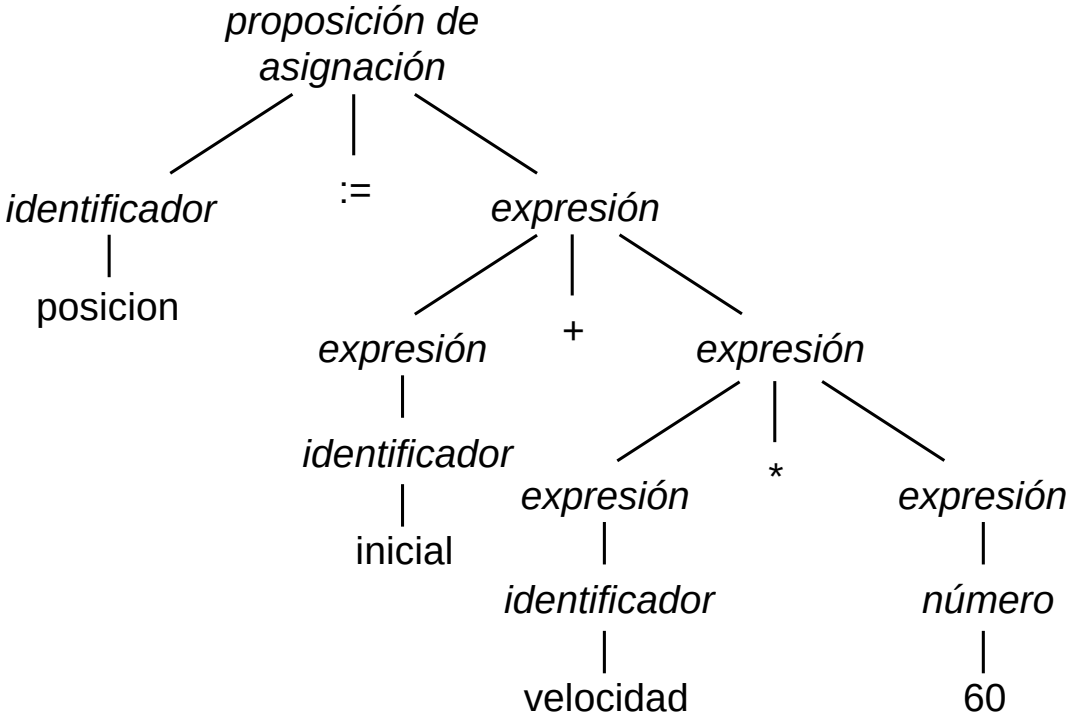
- En la compilación, el análisis consta de 3 fases:
  - Análisis léxico. El analizador léxico *(scanner*) lee la cadena de caracteres del programa fuente, agrupándolos en componentes léxicos (*tokens*).

posicion := inicial + velocidad * 60

- Identificadores: posicion, inicial, velocidad.
- Operador de asignación: :=.
- Operador de suma: +.
- Número: 60.

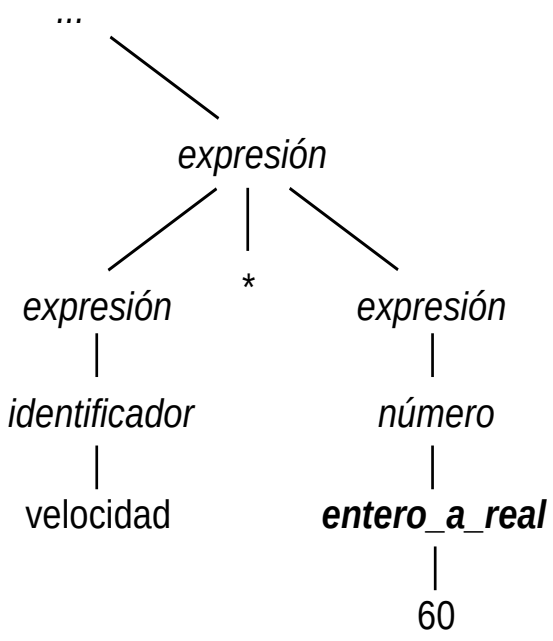
Fases de la Compilación.Análisis del programa fuente

– Análisis sintáctico. El analizador sintáctico agrupa los tokens en frases gramaticales que se usan para sintetizar la salida. Se representa con ayuda de un árbol de análisis sintáctico.



Fases de la Compilación . Análisis del programa fuente

- Análisis semántico. Revisa el programa fuente para comprobar la validez semántica de las sentencias aceptadas por el analizador sintáctico.
  - Un componente importante es la verificación de tipos. El compilador comprueba que cada operador tiene operandos
    - Supongamos que *posicion*, *inicial* y *velocidad* están declarados de tipo real, y que 60 es un número entero.
      - El analizador semántico se ocupa de comprobar que *velocidad* y 60 sean del mismo tipo.
      - En este caso, se puede promocionar el tipo de 60 a real, para que no se produzca un error.

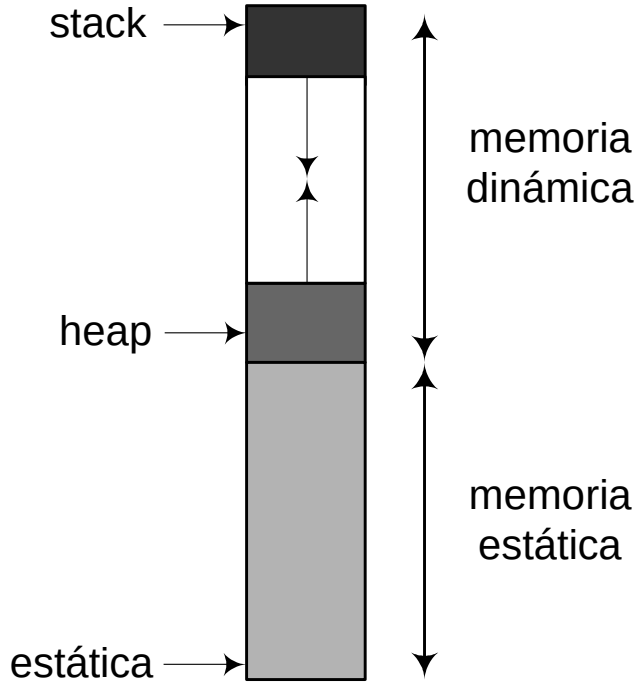


Fases de la Compilación. Manejo de errores y tablas de símbolos

- Durante todas las fases anteriores pueden producirse errores. **El manejador de errores** tiene como objetivos:
  - Informar al programador del error producido.
  - Proporcionar un mecanismo que permita proseguir la compilación con datos erróneos.
- **La tabla de símbolos** es una estructura que almacena información (atributos) sobre cada uno de los identificadores encontrados en el PF.
  - El analizador léxico se encarga de insertar los nuevos identificadores en la tabla.
  - El resto de las fases van añadiendo información.
  - Ejemplos: tipo, valor, memoria asignada, etc.

Fases de la Compilación.
Gestión de Memoria en tiempo de ejecución

- Generalmente, los compiladores reparten la memoria en tres áreas diferentes.
  - Asignación estática de memoria. Se usa con variables globales. Se asigna en tiempo de compilación y se mantiene durante la ejecución.
  - Asignación de memoria dinámica de pila (stack). Variables locales. Se asigna en tiempo de ejecución y se utiliza una estructura de tipo LIFO. Es clave para los lenguajes recursivos.
  - Asignación dinámica de montón (heap). Estructuras dinámicas de datos. Permiten asignar y liberar memoria en tiempo de ejecución.



Fases de un Compilador. Síntesis del programa objeto

- En general, la fase de síntesis se lleva a cabo en 3 pasos:
  - Generación de código intermedio. El código generado corresponde a una máquina abstracta. Se reduce el número de programas necesarios para construir traductores y facilita el transporte a otras máquinas.
  - Optimización de código. Trata de producir un código intermedio lo más eficiente posible.
  - Generación de código. Se convierte el código intermedio en ensamblador, código máquina, u otro lenguaje.

Fases de un Compilador. Agrupamiento de las fases

- Las fases anteriormente descritas se suelen agrupar en:
  - **Front-end.** Constituido por las fases que sólo dependen del lenguaje fuente y son independientes de la máquina.
    - Normalmente incluye el análisis léxico, sintáctico y semántico, la creación de la tabla de símbolos y la generación y optimización de código intermedio.
  - **Back-end.** Comprende las fases que dependen de la máquina objeto y son independientes del programa fuente.
    - Generalmente incluye la generación y optimización del código dependiente de la máquina.

Fases de un Compilador. Agrupamiento de las fases

- Se suelen aplicar varias fases **en una sola pasada** (e.g.: análisis léxico, sintáctico, semántico y generación de código intermedio).

- Existen ciertos factores que pueden incidir en la necesidad de aplicar varios pasos:
  - Complejidad del lenguaje fuente (e.g.: sentencias goto).
  - Técnicas de recuperación de errores.
  - Técnicas de optimización de código.
  - Necesidades de depuración.
  - Tipo de usuario (e.g.: estudiante o profesionales).

Fases de la Compilación

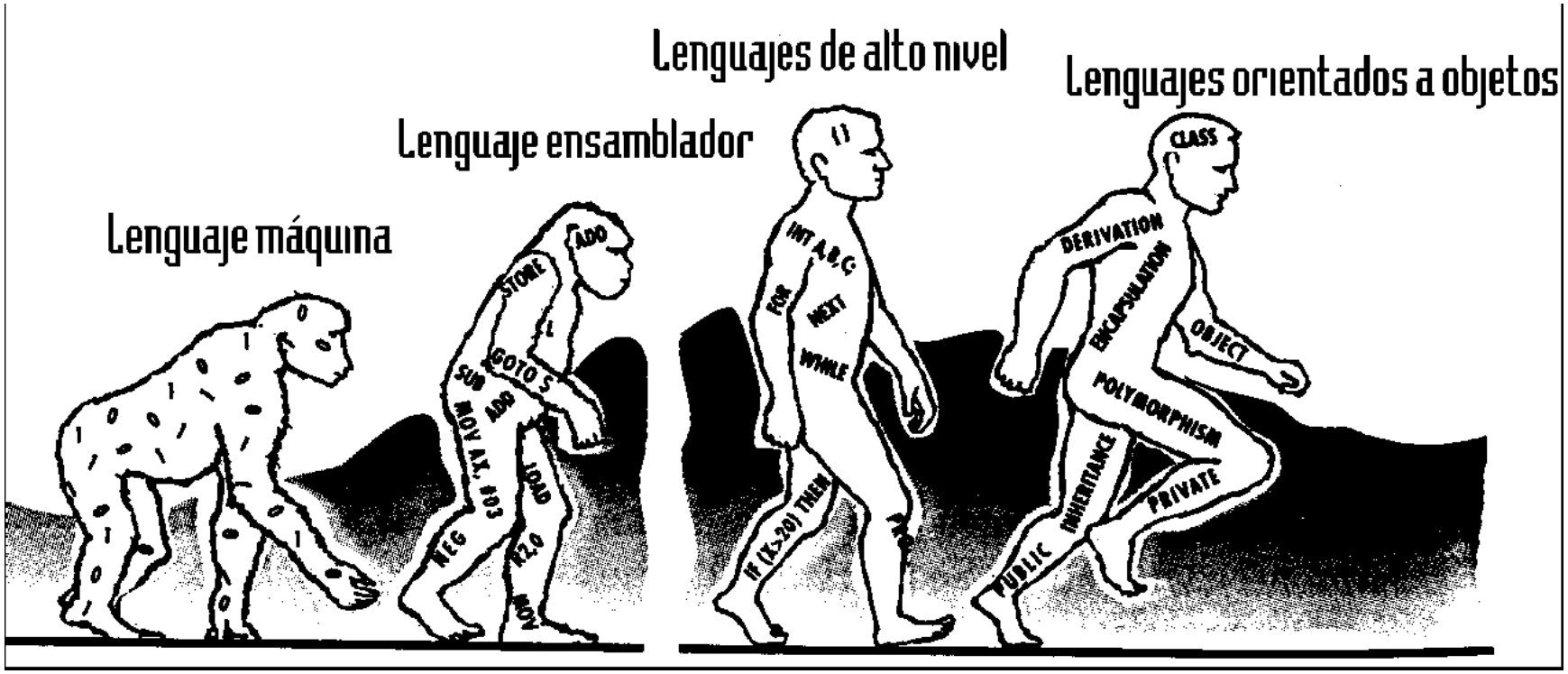
- Para desarrollar un compilador hay que tener en cuenta tres pilares básicos:
  1. La definición léxica, sintáctica y semántica de leng. fuente
  2. La estructura interna del compilador
  3. La arquitectura del ordenador y su juego de instrucciones que constituirá el lenguaje objeto.
- Tercer Pilar:
  - ¿Cómo se pueden ejecutar las aplicaciones desarrolladas para otras arquitecturas de ordenadores en la nueva arquitectura?
  - Dado que un compilador es un programa demasiado complejo para escribirlo directamente en lenguaje máquina ¿Con qué se escribe el primer compilador?

Lenguajes de Programación

Fuente: Universidad de Oviedo

http://www.di.uniovi.es/procesadores/Apuntes/ConceptosBasicos/10_Conceptos_Basicos_Procesadores_Lenguaje.pdf

Lenguajes de Programación



Lenguajes de Programación

- Los lenguajes de programación permiten la comunicación de órdenes al ordenador.
- Se pueden definir como *una notación formal para describir algoritmos que serán ejecutados por un ordenador*.
- Se pueden clasificar desde distintos puntos de vista:
  - Según su grado de independencia de la máquina.
  - Según la forma de sus instrucciones.
  - Por generaciones.
  - Según la forma de ejecución

  Ver : **http://www.levenez.com/lang/**

Lenguajes de Programación

- Según su grado de independencia de la máquina.
  - ***Lenguaje máquina***. Cada instrucción se representa por un código numérico y unas direcciones.
  - ***Lenguaje ensamblador***. Versión simbólica del lenguaje máquina.
  - ***Lenguajes de medio nivel***. Posibilidades de manejo de estructuras de control y de datos, además de acceso directo al sistema (i.e.: C).
  - ***Lenguajes de alto nivel***. Pocas posibilidades de acceso directo al sistema y una gran capacidad de estructuración (i.e.: Pascal). Destacan los Lenguajes Orientados a Objetos (i.e.: C++, Smalltalk).
  - ***Lenguajes para problemas concretos***. Utilizados para resolución de problemas en un campo específico (i.e.: SQL, SPSS, Postscript)

Lenguajes de Programación

- Según la forma de sus instrucciones.(Imperativos, Declarativos, Concurrentes y OO)
  - ***Lenguajes imperativos o procedimentales***.
    - Su unidad básica es la instrucción o sentencia, y hacen uso de estructuras de control y de manejo de bloques (C, Pascal, Ada). COBOL y FORTRAN son los precursores.
    - Uso intensivo de variables.
    - Manejo frecuente de instrucciones de asignación
    - Resolución de algoritmos por medio de estructuras de control secuenciales, alternativas y repetitivas → Uso intensivo del GOTO
    - Gestión de memoria dinámica heap en tiempo de ejecución: Los lenguajes imperativos se dotaron de mecanismos que permiten reservar y liberar memoria dinámicamente (ALGOL).

Lenguajes de Programación

- ***Lenguajes declarativos*.** Son lenguajes de muy alto nivel que utilizan una notación próxima al del problema real que resuelven. Hay dos tipos de lenguajes: **Funcionales** y **Lógicos**
  - Funcionales (LISP): Tienen todas sus construcciones como llamadas a funciones matemáticas. No hay instrucciones
    - Bajo rendimiento y gran consumo de memoria frente a los imperativos (actualmente se están acercando)
  - Lógicos (PROLOG): Definen sus instrucciones siguiendo un tipo de Lógica. PROLOG → Lógica clausal restringida a cláusulas de Horn.
    - Maneja relaciones (predicados) entre objetos (datos)
    - Las relaciones se especifican con reglas y hechos. Demostración de hechos sobre las relaciones por medio de preguntas.

Lenguajes de Programación

– ***Lenguajes Concurrentes*.**

- Son los que permiten la ejecución simultánea ("paralela" o "concurrente") de dos o varias tareas.
- La concurrencia puede ser una característica propia del lenguaje, o el resultado de ampliar las instrucciones de un lenguaje no concurrente.
- Ejemplos: Ada, Concurrent C, Concurrent Pascal, Concurrent Prolog, CSP, Argus, Actors, Linda, Monitors, Ccc32 compilador deCconcurrente paraWindows.etc...

Lenguajes de Programación

- ***Lenguajes orientados a objetos*.**
  - Un lenguaje de programación se dice que es un *lenguaje basado en objetos* si soporta directamente tipos abstractos de datos y clases (por ejemplo Ada80 y Visual Basic 4).
  - Un *lenguaje* orientado a objetos es también basado en objetos, pero añade mecanismos para soportar la ***herencia*** (como un medio para soportar la jerarquía de clases) y el ***polimorfismo*** (como un mecanismo que permite relizar una acción diferente en función del objeto o mensaje recibido).
  - Ejemplos: Smalltalk, C++, Object Pascal, Turbo Pascal, Delphi, CLOS, Prolog++, Java y Eiffel.

Lenguajes de Programación

- **Por generaciones:**
  - ***Primera generación***. Lenguajes máquina y ensamblador, años 40 y 50.
  - ***Segunda generación***. Lenguajes con asignación estática de memoria. Fortran, 1958.
  - ***Tercera generación*.** Programación estructurada, años sesenta y setenta. Algol60.
  - ***Cuarta generación***. Lenguajes de muy alto nivel para tareas muy específicas (i.e.: bases de datos).
  - ***Quinta generación***. Lenguajes ligados a la IA.
  - Generación OO. Años 80.
  - **Generación visual.** Diseño de interfaz de usuario, años 90.
  - **Generación Internet.** Aplicaciones multiplataforma, finales de los 90.

Lenguajes de Programación

Según la forma de ejecución:

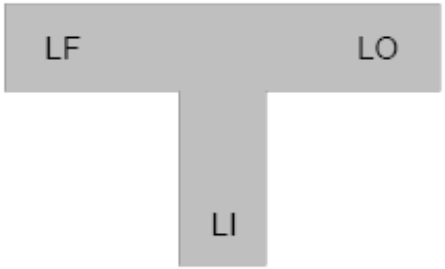
- **Compilados:** Son lenguajes que están, como su propio nombre indica compilados, es decir se convierte a binario y se ejecutan tantas veces como sea necesario. La mayoría de los lenguajes actuales son compilados: C, C++, C#, Java, Pascal, etc. aunque también tienen su versión interpretada

- **Interpretados:** Cada vez que se usa el programa debe utilizarse un traductor denominado “intérprete” que se encarga de traducir las instrucciones del código fuente a código máquina según van siendo utilizadas. Ejemplos de estos lenguajes son BASIC, Python o Ruby.

Tipos de Procesadores de Lenguaje

Procesadores de Lenguaje

- **Traductor**:
  - Es un programa que procesa un texto fuente y genera un texto objeto. Se suele utilizar notación en T



- **Compilador**:
  - Un traductor que transforma textos fuente de lenguaje de alto nivel a lenguajes de bajo nivel se le denomina compilador

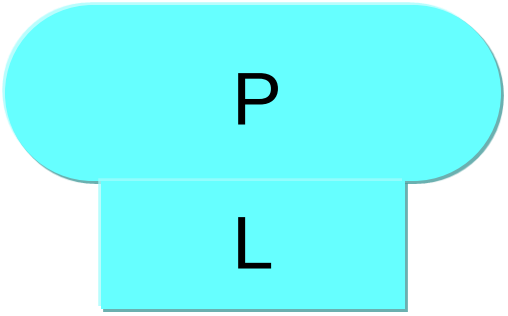
Diagramas de Tombstone

Diagramas de Tombstone

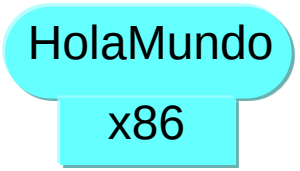
- Conjunto de “piezas de puzzle” útiles para razonar acerca de los procesadores de lenguaje y los programas.
  - Tienen diferentes tipos de piezas
  - Hay reglas de formación, no todos los diagramas están permitidos.

Diagramas de Tombstone

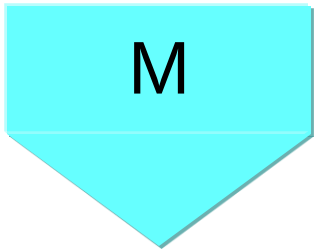
- Diagrama para Programas



Programa P
expresado en
el lenguaje L



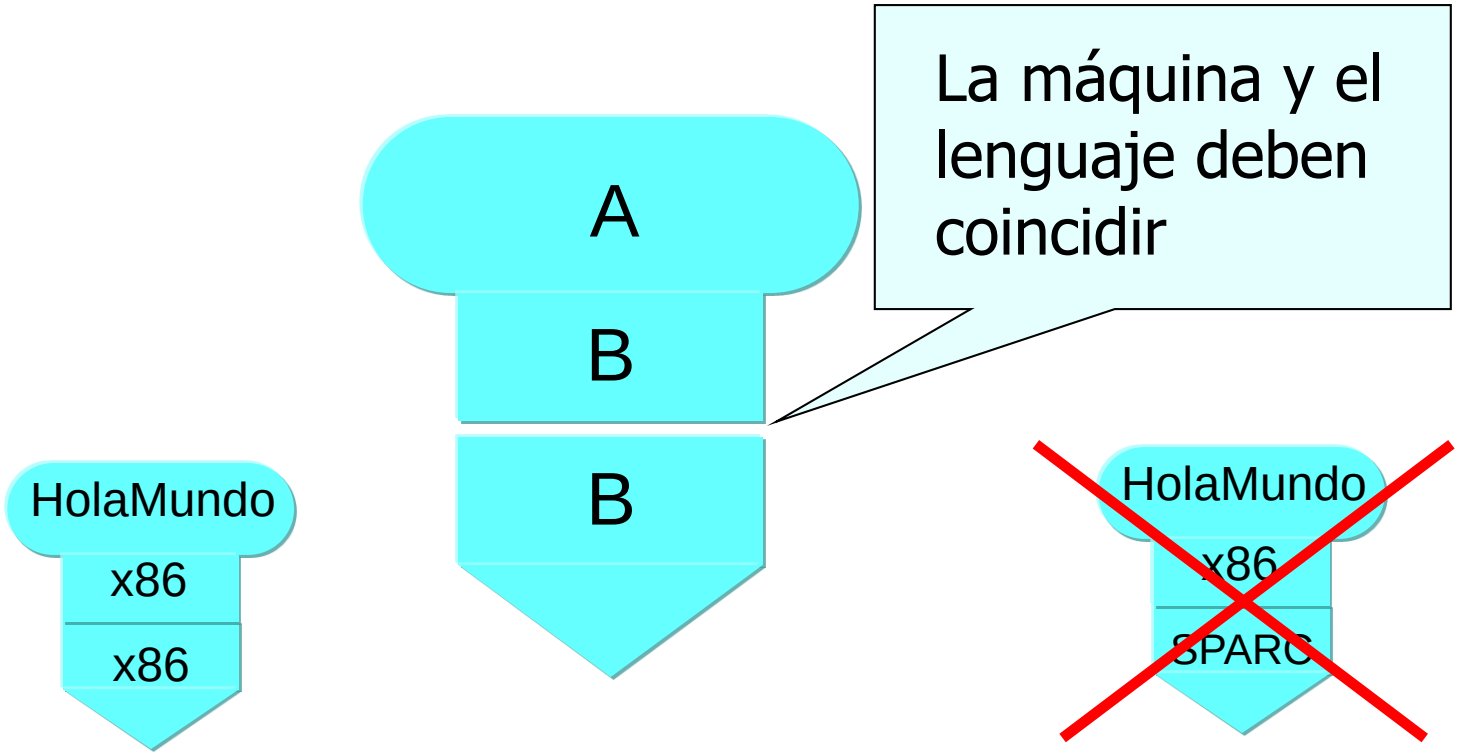
- Diagrama para Máquinas



Máquina M
capaz de ejecutar
programas en el
lenguaje M

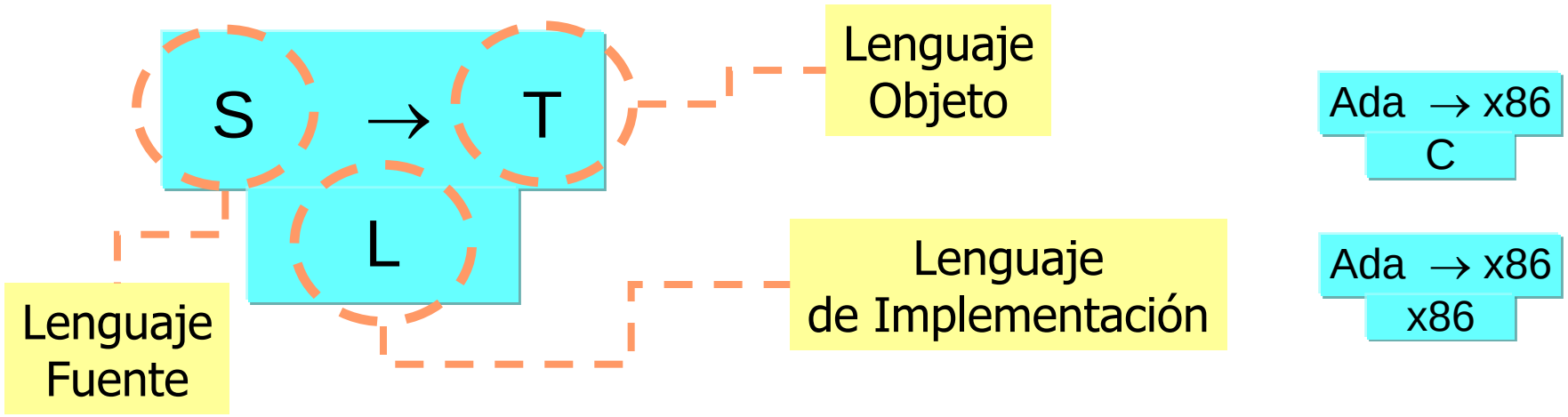
Diagramas de Tombstone

- Ejecución de un programa en una máquina

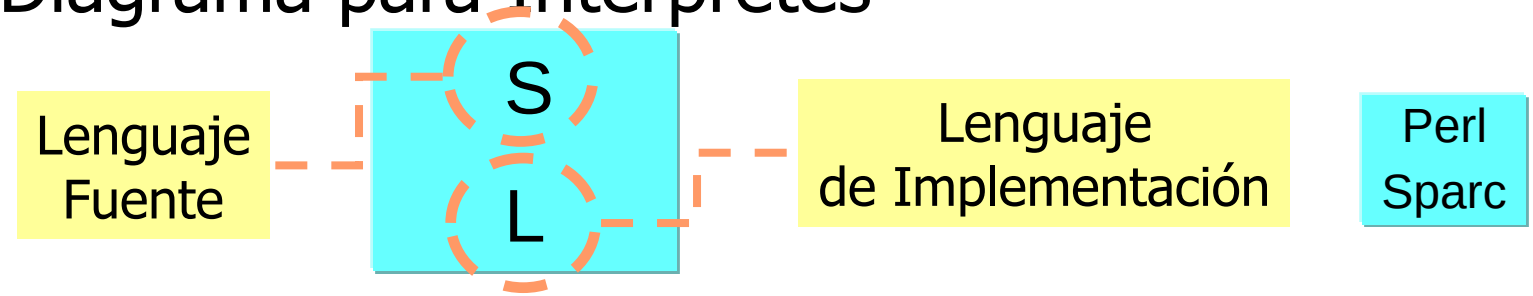


Diagramas de Tombstone

- Diagrama para Traductores/Compiladores

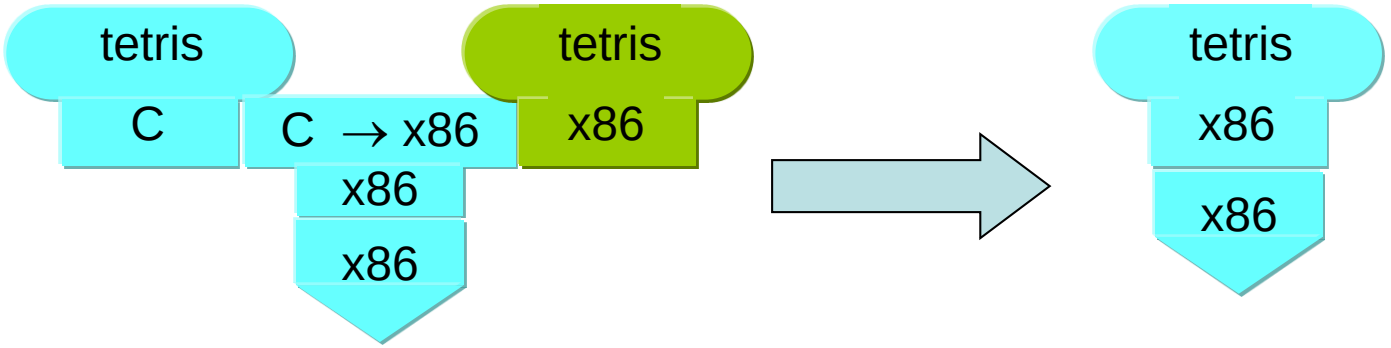


- Diagrama para Intérpretes



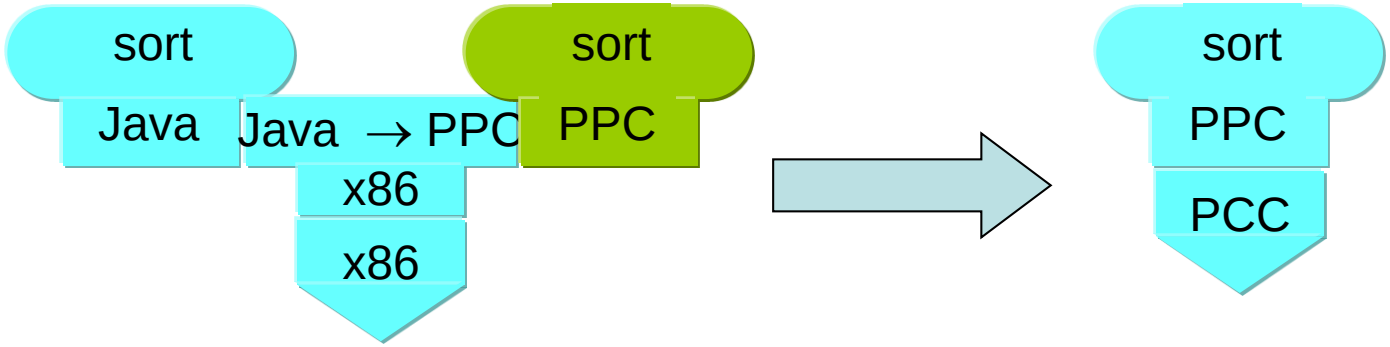
Diagramas de Tombstone

- Ejemplo de compilación



Diagramas de Tombstone

- **Compiladores Cruzados:**
  - Cuando se trabaja con un nuevo tipo de procesador, alguien tiene que escribir el primer compilador.
  - Esta tarea se puede realizar mas cómodamente en una máquina donde ya se disponga de herramientas software → C. cruzado
  - Es un compilador que se ejecuta en una máquina pero el código objeto es para otra máquina

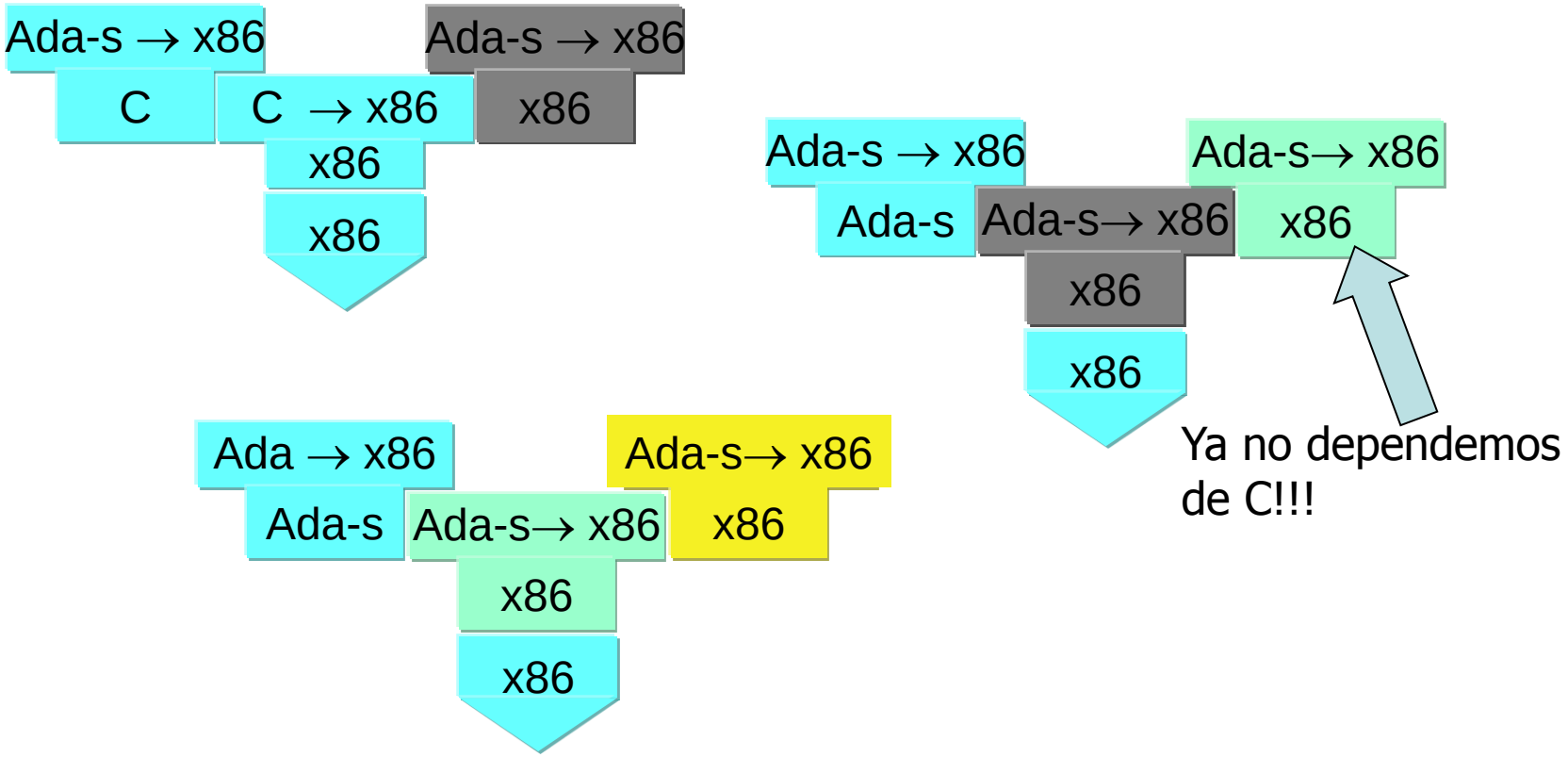


Bootstrapping (“truco o trampa de arranque”)

- Se basa en utilizar las facilidades que ofrece un lenguaje para compilarse a si mismo. Compilar un compilador (una versión antigua) consigo mismo.
  - Se construye el compilador de un lenguaje a partir de un subconjunto de dicho lenguaje.
- Primer paso:
  - Se escribe un pequeño compilador que traduce un subconjunto del código Fuente de un lenguaje L a un lenguaje objeto que es el de la máquina M.
- Segundo paso:
  - Se puede usar ese subconjunto para escribir un compilador en L que se ejecuta sobre M para L

Bootstrapping (“truco o trampa de arranque”)

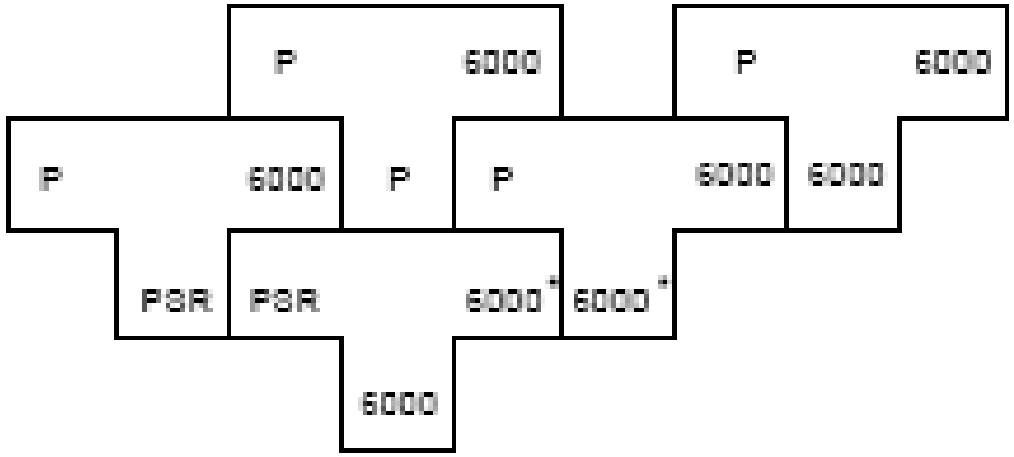
- Las mayores ventajas del bootstraping se producen cuando un compilador se escribe en el lenguaje que se compila



Bootstrapping (“truco o trampa de arranque”)

- Ejemplo: Compilador de PASCAL (1981)
  1. Se eligió un subconjunto básico de Pascal y se programó en SCALLOP (lenguaje de nivel medio de ordenador CDC). Este primer conjunto se llamó Pascal si revisar (SRP). Por medio de un bootstraping se consiguió un compilador PSR para un CDC 6000.→ **$PSR_{6000}6000$**
  2. Se programó un nuevo compilador de Pascal revisado (P), utilizando el PSR para el 6000 → **$P_{PSR}6000$**
  3. Se compila el compilador obtenido en 2 con el obtenido en 1 y se obtiene → **$P_{6000}6000$.**
  4. Se vuelve a programar un compilador de Pascal revisado, pero esta vez en Pascal revisado → **P_P6000**
  5. Se compila el compilador 4 con el obtenido en 3 y → **$P_{6000}6000$**

Bootstrapping (“truco o trampa de arranque”)



Validación

Validación

- Se realiza por medio de un conjunto de pruebas sistematizadas, que verifican cada uno de los módulos del traductor.
- Un criterio clásico consiste en utilizar seis tipos de baterías de test, denominados con una letra (A, B, L, C, D, E).
- Los programas de test se desarrollan utilizando el lenguaje fuente, y se les aplica el traductor desarrollado.
- Un traductor que pase estas baterías de test no garantiza estar libre de fallos.

Validación

- Tests de tipo A.
  - Programas específicos que no contengan errores de compilación y que permitan al traductor generar un código objeto.
  - Se suele crear un programa por instrucción, y posteriormente realizar pruebas con grupos de instrucciones.

- Tests de tipo B.
  - Programas que contienen errores específicos que deben ser detectados en tiempo de compilación.
  - El traductor debe detectar cada error y mostrar el mensaje correspondiente.
  - Debe construirse al menos un programa por cada tipo de error (léxico, sintáctico o semántico).

Validación

- Tests de tipo L.
  - Programas con errores que sólo se pueden detectar en la etapa de carga y enlace.

- Tests de tipo C.
  - Programas similares a los del tipo A, pero comprobados ahora en tiempo de ejecución.
  - Se debe verificar que el ejecutable realiza las instrucciones del programa fuente.
  - Se suelen ampliar los programas de tipo A para comprobar aspectos específicos de la generación de código.

Validación

- Tests de tipo D.
  - Comprueban ciertas capacidades máximas:
    - Nº máximo de identificadores en un bloque.
    - Agotamiento de la pila por recursividad.
    - Máximo nivel de anidamiento de bloques.
  - Suelen ser dependientes de la implementación y del ordenador en que se ejecute.

- Tests de tipo E.
  - Buscan posibles ambigüedades en las instrucciones (resultados ambiguos).
  - Las ambigüedades deberían ser eliminadas en la fase de diseño.

Motivación y Aplicaciones

Motivación (I)

- Para ser buen programador
  - Saber como se obtiene un ejecutable permite saber más sobre corrección y eficiencia
- Para entender más sobre lenguajes
  - Tipificación: estática, dinámica, fuerte, polimorfismo, conversiones, sobrecarga de operadores...
  - Estructura de bloques, ámbitos
  - Paso de parámetros
  - Gestión de memoria, punteros

Aplicaciones

- Desarrollo de interfaces de texto
- Tratamiento de ficheros de texto estructurados
  - Perl, Tcl, comando de Unix (egrep)
- Procesadores de texto
  - vi,emacs
- Formateo de texto y descripción gráfica
  - HTML, TeX, Postscript
- Gestión de bases de datos
- Procesamiento del Lenguaje Natural
- Traducción de formatos de programas
- Cálculo simbólico
- Reconocimiento de formas