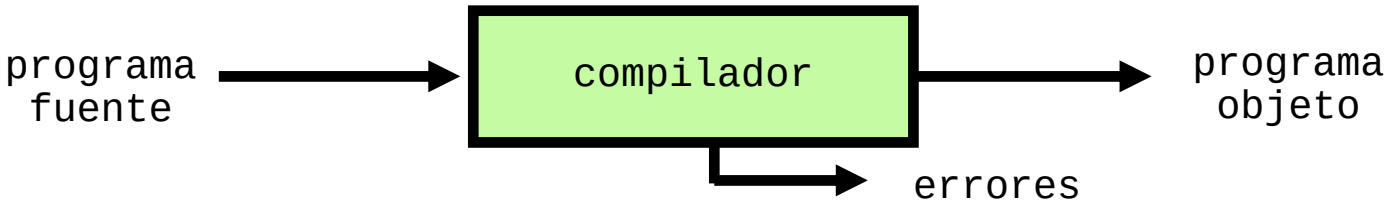
Lección 1: Introducción a los compiladores

1) Introducción
2) Las partes de un compilador
   - 2.1) El analizador léxico
   - 2.2) El analizador sintáctico
   - 2.3) El analizador semántico
   - 2.4) El optimizador
   - 2.5) El generador de código
   - 2.6) La tabla de símbolos
3) ¿Y los intérpretes?

- ¿Qué es un compilador?
  - Programa que lee un programa (fuente) en un lenguaje
  - Lo traduce a un programa EQUIVALENTE en otro lenguaje (objeto)
  - Además:
    - » da mensajes de error
    - » lleva a cabo determinadas "correcciones" (recuperación de errores)
    - » puede optimizar el código generado



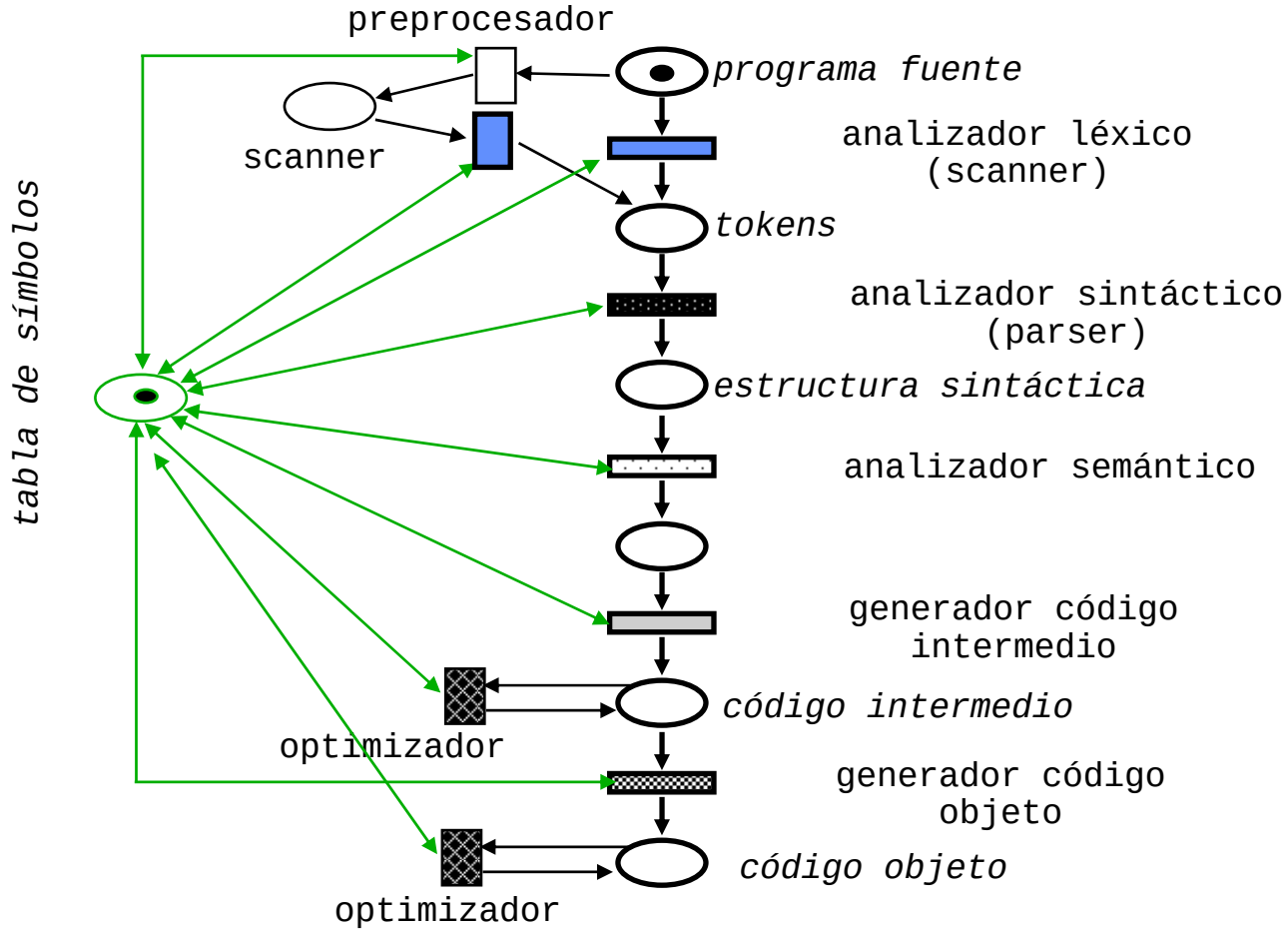
- Permite programar "independientemente" de la máquina
  - Importante, ya que el número de máquinas diferente crece deprisa

- ¿De dónde el nombre "compilador"?
  - Murray Hopper (50's)
  - La traducción se veía como la "compilación" de una secuencia de subprogramas tomados de una librería (biblioteca) de programas
  - Compilación (actual) se llamaba "programación automática"
    - » se veía como algo futurista
  - Primeros compiladores modernos: FORTRAN (finales 50)
    - » "independencia" de la máquina
    - » coste: 18 personas/año
    - » desarrollo de técnicas "ad hoc"

Introducción

- No siempre el lenguaje objeto tiene que ser de "bajo nivel" (traductor)
- Técnicas aplicables para:
  - editores/formateadores de texto
    - » nroff, troff, eqn, tbl, pic de UNIX,TeX
  - lenguajes de consulta
    - » SQL, shells
  - transformación de formatos de ficheros

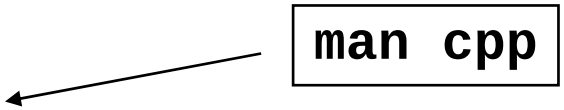
Las partes de un compilador



Las partes de un compilador

- **Primera fase** (precompilador)
  - no siempre se realiza
  - sustituciones de macros
  - eliminación de comentarios
  - inclusión de ficheros
  - extensiones al lenguaje (C+SQL)
- **Segunda fase**
  - es la parte fundamental (y siempre presente)
  - consta de:
    - » **analizador léxico**
    - » **analizador sintáctico**
    - » **generador de código**
  - traduce el código fuente a otro objeto
    - » puede ser el definitivo
    - » puede ser un código intermedio

Preprocesador (precompilador)



etc.

The C preprocessor provides four separate facilities that you can use as you see fit:

- ***Inclusion of header files**. These are files of declarations that can be substituted into your program.*
- ***Macro expansion**. You can define macros, which are abbreviations fo arbitrary fragments of C code, and then the C preprocessor will re place the macros with their definitions throughout the program.*
- ***Conditional compilation**. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.*
- ***Line control**. If you use a program to combine or rearrange source files into an intermediate file which is then compiled, you can us line control to inform the compiler of where each source line originally came from.*

etc.

Las partes de un compilador

- **Tercera fase:**
  - no siempre presente
  - realiza optimizaciones (algunas) sobre el código (intermedio) generado
- **Cuarta fase**:
  - traduce el código intermedio (optimizado) a
    - » ensamblador
    - » Binario
- **Quinta fase**:
  - Optimización ligada a la máquina de destino

- Muchas variaciones posibles:
  - sin preprocesador
  - sin usar código intermedio
  - optimizando directamente sobre el ensamblador de la máquina
  - generar directamente binario, sin pasar por el ensamblador
  - .....

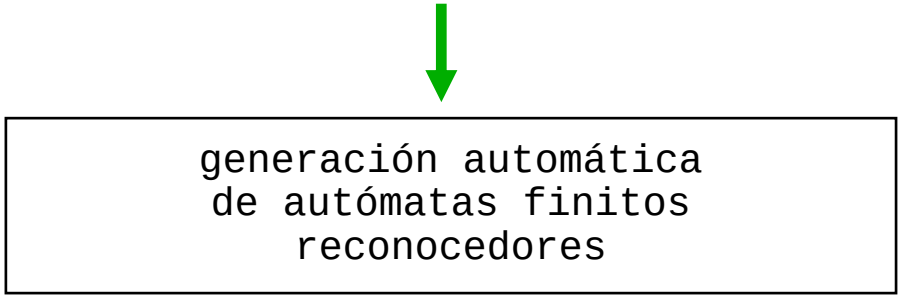
El analizador léxico

- Lo realiza un "scanner"
  - también "tokenizer"
- El scanner recorre los caracteres de entrada (el fuente) hasta reconocer un "*token*"
  - token: unidad léxica indivisible
  - ejemplos: `while`, `if`, `==`, `>=`, `ancho`, `...`
- La secuencia de caracteres correspondiente se llama

  "*lexema*" (componente léxico)
- 1 token <> 1 lexema

El analizador léxico

- Además, suele realizar otras tareas:
  - procesar directivas al compilador (opciones)
  - introducir información preliminar en la tabla de símbolos
  - eliminar separadores innecesarios
  - sustituir macros
  - listar el fuente
- Normalmente, los tokens se describen mediante expresiones regulares



- Ejemplo:

```
posición = inicial + velocidad * 60
```

- El scanner deberá reconocer sucesivamente,

token	lexema
IDENTIFICADOR	posición
'='	=
IDENTIFICADOR	inicial
'+'	+
IDENTIFICADOR	velocidad
'*'	*
CONSTANTE	60

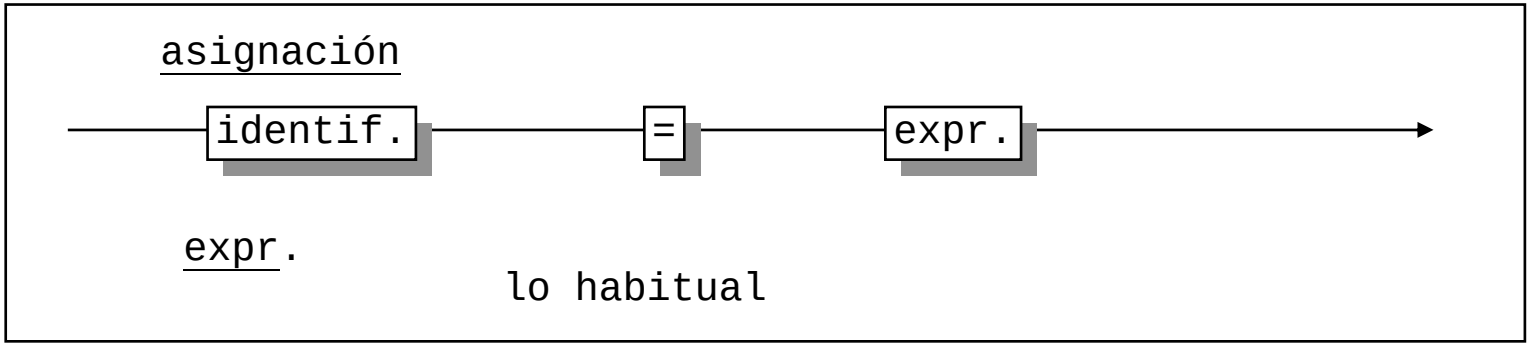
El analizador sintáctico

- O "*parser*"
- Objetivo: agrupar los tokens suministrados por el scanner para reconocer "*frases*"
- ¿Cómo lo hace?
  - La sintaxis se suele especificar formalmente mediante una GLC
    - » también de otros tipos
  - El parser recibe tokens y los agrupa de acuerdo a

    `producciones`

    especificadas por la GLC
- El parser detecta errores sintácticos
- Y si es bueno, puede además realizar algunas correcciones

- Ejemplo:
  - supongamos sintaxis asignación como:

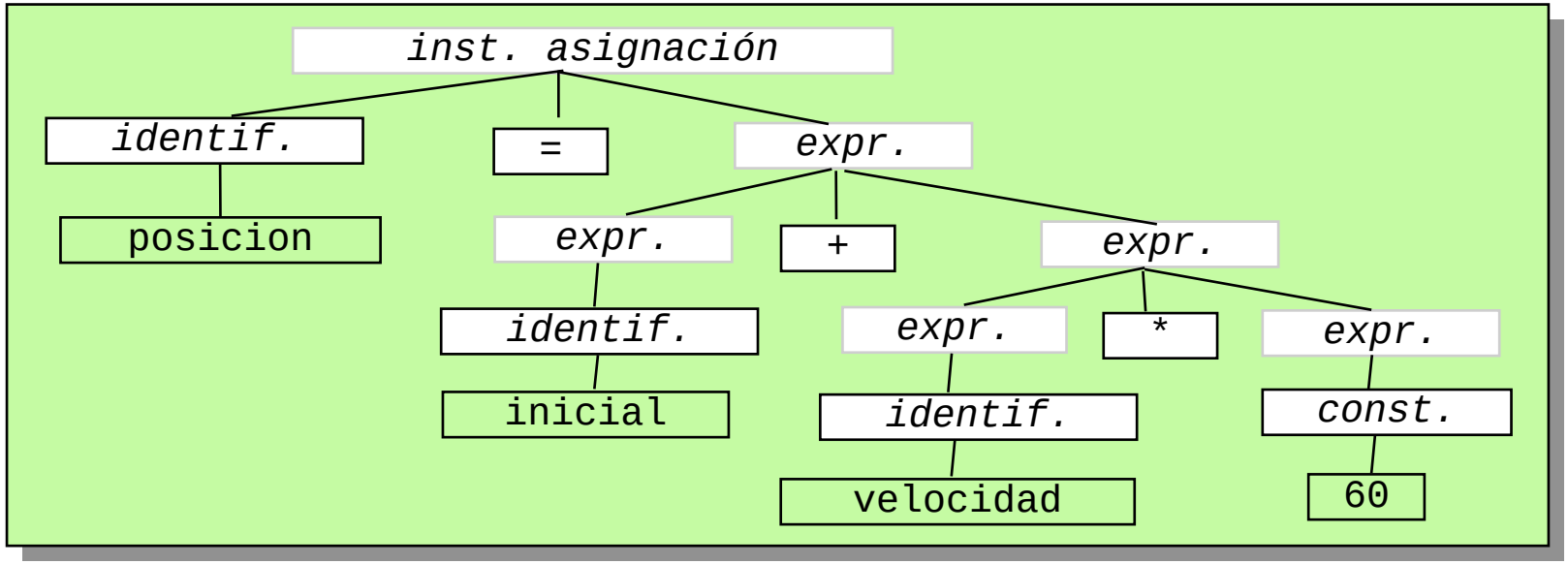


  - y debemos analizar

```
posicion = inicial + velocidad * 60
```

- El árbol sintáctico correspondiente es

```
posicion = inicial + velocidad * 60
```



El analizador semántico

- Realiza dos funciones:
  - Análisis de la semántica estática
    - » ¿Es cada construcción legal y "con sentido"?
      - variables en una expresión definidas
      - del tipo adecuado
      - alcance de los objetos
  - Generación del código (intermedio)
    - » Generalmente se lleva a cabo mediante *gramáticas de atributos*
      - la GLC se completa con atributos necesarios
        - tipo
        - valor
        - acciones a ejecutar cuando se detecta una construcción legal
        - ....

RTL (Register Transfer Language - GNU)

(binop mode operand1 operand2)
Perform a binary arithmetic operation. `binop' is one of add, sub, and, or, xor, mul, div, udiv, mod, umod.

(binop-with-bit mode operand1 operand2 operand3)
Same as `binop', except taking 3 operands. The third operand is always a single bit. `binop-with-bit' is one of addc, add-cflag, add-oflag, subc, sub-cflag, sub-oflag.

(shiftop mode operand1 operand2)
Perform a shift operation. `shiftop' is one of sll, srl, sra, ror, rol.

(boolifop mode operand1 operand2)
Perform a sequential boolean operation. `operand2' is not processed if `operand1' "fails". `boolifop' is one of andif,orif.

(convop mode operand)
Perform a mode->mode conversion operation. `convop' is one of ext, zext, trunc, float, ufloat, fix, ufix.

El optimizador

- El código intermedio generado es analizado y transformado en uno equivalente optimizado
- Es una tarea muy costosa
  - de hecho, generalmente se puede invocar al compilador activando/desactivando esta opción
  - es una tarea difícil
- Otras veces, optimiza el código objeto
  - usual la optimización "peephole":
    - » tomar una porción pequeña de código y hacer una optimización local
  - "desenrrollado" de bucles
  - eliminación de recursividad final
  - ....

El generador de código

- Toma código intermedio y genera código objeto para la máquina considerada
- Es la parte más próxima a la arquitectura de la máquina
- Habitualmente, se escriben "a mano"
  - desarrollo "a medida" para cada máquina específica
- Dada la complejidad, si no se va a realizar optimización, se asocia la generación de código a las rutinas semánticas
  - compiladores de "una pasada" (no hay pasos 1,3,4)

La tabla de símbolos

- Mecanismo para almacenar/acceder la información de los identificadores
- Las informaciones asociadas a un identificador se denominan "***atributos***"
- Cada vez que se usa un identificador, la tabla de símbolos proporciona la información necesaria
- Se usa tanto en la parte de análisis como en la de síntesis

¿Y los intérpretes?

- Un intérprete:
  - ejecuta programas sin una traducción explícita
  - es decir: no hay una fase de traducción y otra de ejecución
  - no se genera código máquina. El intérprete ejecuta
- Ventajas:
  - la propia ejecución puede modificar el programa
    - » datos y programa son datos
  - variables pueden cambiar dinámicamente de tipo
  - facilidades de depuración
    - » fuente presente durante ejecución
- Muchas técnicas comunes con compiladores

¿Y los intérpretes?

- Inconvenientes:
  - lentitud (entre 1/10 y 1/100 de velocidad)
  - sobrecarga de memoria
    - » programa en ejecución e intérprete simultáneamente en memoria
    - » además de tablas de símbolos y otros elementos necesarios para la traducción
- Util: ambas cosas
  - intérprete para la fase de desarrollo
  - compilador para generar versión definitiva