Apéndice

B

Compilador y enlazador

En este apéndice veremos como trabaja el compilador: estructuras de datos, fases de compilación y análisis sintáctico y semántico. También se verá la función del programa enlazador o "*linker*"

Un compilador es un programa que traduce un código fuente de un lenguaje de alto nivel a ensamblador o directamente a código máquina. El compilador puede estar escrito en ensamblador, en un lenguaje de alto nivel (tendencia actual) como por ejemplo C (ya que es un lenguaje de nivel medio) o incluso en un lenguaje especial para construir compiladores (compilador de compiladores), en el caso de UNIX puede ser lex y yacc o bison. No hay que confundirlo con un intérprete que lee línea a línea un programa y lo va ejecutando, por ejemplo, los antiguos BASIC o Java.

El funcionamiento de un compilador está dividido fundamentalmente en dos partes: Una de análisis del código fuente y otra de síntesis del código objeto, para lo cual, al igual que el ensamblador, debe construir y analizar varias tablas (ver Figura 1).

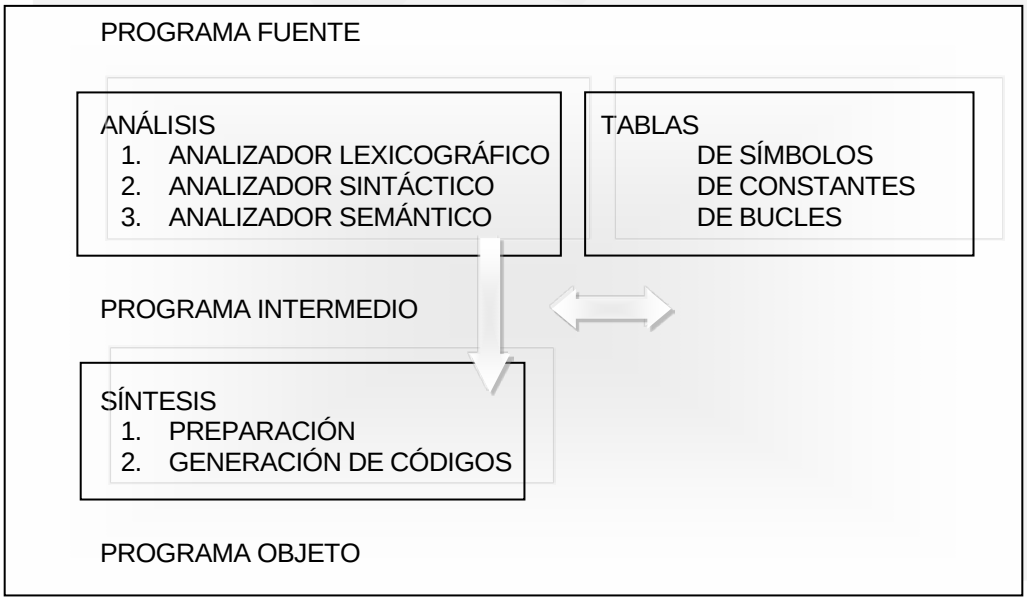

Figura 1. Proceso de compilación.

En la fase de análisis aparecen tres componentes:

1. El analizador lexicográfico (*scanner*). Tiene como misión explorar los caracteres del programa, convertirlo a un formato apropiado para la fase sintáctica (quitar partes innecesarias como comentarios y convertirlo a la notación adecuada) y construir las tablas de datos necesarias. Para ello distingue entre varios tipos de componentes (llamados *tokens* o unidades léxicas):

   - Constantes: De tipo numérico o alfabético.
   - Identificadores: Nombres dados a entidades propias del programa (variables, subprogramas, etc.).
   - Operadores: Los dados en un determinado programa (aritméticos, relacionales, lógicos, etc.).
   - Palabras clave: Predefinidas por el lenguaje para poder construir las sentencias.
   - Delimitadores: Separadores de los anteriores.

   A cada uno de estos grupos se les aplican sus reglas propias, construyéndose para ellos las tablas de símbolos correspondientes (si no lo están ya, como la de palabras reservadas). El *scanner* puede dar errores de escritura.

2. El analizador sintáctico. Tomando el código producido por el analizador lexicográfico, el sintáctico es el encargado de determinar si una sentencia es correcta o no (de acuerdo a las reglas del lenguaje), para ello se sigue un algoritmo reconocedor o "*parser*". Si todo es correcto (tendrá que seguir las reglas de la gramática del lenguaje que se pueden dar con distintas representaciones: BNF, diagramas, etc.) producirá un árbol sintáctico reflejo del programa que es usado por el analizador semántico.

3. El analizador semántico. Tiene como misión construir un código intermedio independiente de la máquina donde se ejecutará el código objeto. Para ello utilizará las rutinas semánticas que reflejan las leyes de la gramática.

En la parte de síntesis se deberá producir el código objeto a partir de este código intermedio. Esta parte es similar al análisis, sólo que ahora se genera código por cada una de las partes de la sentencia que se está evaluando.

Además en esta fase se puede producir una optimización del código objeto resultante, que puede ser de dos tipos:

- Independiente de la máquina.
  - Reducción simple: Sustitución de operaciones aritméticas innecesarias por constantes.
  - Ordenación de instrucciones: El resultado es el mismo pero una expresión reordenada puede producir código más eficiente al eliminar registros intermedios.
  - Reducción de potencia: Sustitución de operaciones complejas (como potenciación por otras más simples).
  - Reducción de invariantes: Quitar de lazos elementos constantes u operaciones constantes que no dependen de la ejecución del bucle.

- Dependiente de la máquina.
  - Utilización de registros eficientemente. Se pueden usar algoritmos para seleccionar registros víctima a la hora de hacer cargas y guardas de valores.

El código objeto producido puede ser absoluto o reubicable. En el primer caso el código se va a ejecutar en una posición fija de la memoria, en el segundo caso el código puede ejecutarse en cualquier posición, pudiendo tener cuatro partes:

- Tabla de símbolos. Define las distintas partes lógicas de un programa como las referencias externas, las áreas de datos comunes, etc.
- Texto. El formado por el propio código y los datos utilizados por él.

- Tabla de reubicación. Define las direcciones que tienen que cambiarse con respecto a la dirección de ejecución inicial.
- Terminación. Indica la terminación del programa.

También hay que tener en cuenta que un programa objeto puede hacer referencia a otros programas objeto. Estas referencias son solucionadas por una fase posterior realizada por el programa montador, enlazador o *linker*, que dependiendo de la complejidad del sistema operativo puede estar unido al programa cargador o no.