I.1. Esquema de un compilador.

Generalidades.

Traductor: Es una máquina teórica que tiene como entrada un texto escrito en un lenguaje L1 y como salida un texto escrito en un lenguaje L2. Habitualmente se denomina a L1 *lenguaje fuente* y a L2 *lenguaje objeto*.

Las técnicas que se desarrollan en esta asignatura no sólo son válidas para la implementación de compiladores, sino que son aplicables en general a todos los sistemas de procesamiento de lenguajes y de traducción. Estos sistemas pueden ser de distintos tipos:

Traductores de lenguaje natural: Serían los que tradujeran un lenguaje natural en otro, por ejemplo español a inglés. Esto en la actualidad no se ha conseguido debido fundamentalmente a la ambigüedad del lenguaje natural. Los mayores logros en la materia siempre trabajan con un subconjunto del lenguaje natural, limitando las construcciones sintácticas válidas y/o el vocabulario. Este tema se aborda generalmente mediante técnicas de inteligencia artificial.

Compilador: Es un traductor que convierte un texto escrito en un lenguaje fuente de alto nivel en un programa objeto en código máquina.

Intérprete: Es un traductor que realiza la operación de compilación paso a paso. Para cada sentencia que compone el texto de entrada, se realiza una traducción, ejecuta dicha sentencia y vuelve a iniciar el proceso con la sentencia siguiente. La principal ventaja del proceso de compilación frente al de interpretación es que los programas se ejecutan mucho más rápidamente una vez compilados; por el contrario, es más cómodo desarrollar un programa mediante un intérprete que mediante un compilador puesto que en el intérprete las fases de edición y ejecución están más integradas. La depuración de los programas suele ser más fácil en los intérpretes que en los compiladores puesto que el código fuente está presente durante la ejecución. Estas ventajas pueden incorporarse al compilador mediante la utilización de entornos de desarrollo y depuradores simbólicos en tiempo de ejecución.

Preprocesadores: Procesan un texto fuente modificándolo en cierta forma previamente a la compilación. Por ejemplo muchos compiladores admiten un conjunto de macroinstrucciones ajenas al lenguaje en sí que indican al compilador si tiene que incluir algún fichero externo, si ha de hacer o no un listado completo de la compilación, etc.

Conversores Fuente-Fuente: (LCP) Traducen un lenguaje fuente de alto nivel a otro. Por ejemplo PASCAL -> C. Una aplicación interesante de la traducción fuente-fuente es el desarrollo e implementación de prototipos de nuevos lenguajes de programación. Así, por ejemplo, si se desea definir un lenguaje especializado puede implementarse rápidamente mediante su traducción a un lenguaje convencional de alto nivel.

Rutinas de análisis de instrucciones: El conjunto de instrucciones del entorno de un sistema operativo constituye un lenguaje que debe ser analizado previamente para realizar las acciones oportunas. Igualmente ciertos programas como editores de texto, sistemas de diseño asistido, etc. utilizan instrucciones complejas que deben interpretarse adecuadamente.

Ensambladores: Son compiladores cuyo lenguaje de entrada, llamado ensamblador permite la traducción de cada sentencia fuente a una instrucción en código máquina.

Compilador cruzado: Es el que genera un código objeto ejecutable en un ordenador distinto de aquél en el que se realiza la compilación.

Compilación-Montaje-Ejecución: En las aplicaciones grandes es conveniente fragmentar el programa a realizar en módulos que se compilan por separado y una vez que estos estén compilados unirlos mediante un programa denominado montador para formar el módulo ejecutable. El montador se encarga a su vez de incluir las librerías en donde se guardan las funciones predefinidas de uso común.

Compilación en una o varias pasadas: Se llama pasada a cada lectura que hace el compilador del texto fuente.

Compilación incremental. Este compilador actúa de la siguiente manera. Compila un programa fuente. Caso de detectar errores al volver a compilar el programa corregido solo compila las modificaciones que se han hecho respecto al primero.

Autocompilador: Es aquel que está escrito en el mismo lenguaje que se pretende compilar. Supongamos por ejemplo que queremos desarrollar la versión 2.0 de un compilador Pascal. Dicho compilador generara un código mucho más rápido y eficiente que el que generaba la versión anterior 1.0. Sin embargo son ya muchas las máquinas (IBM 370, Serie 1, PDP 11 ..) que disponen del antiguo compilador, o que al menos tienen otro compilador Pascal. La mejor opción consiste en escribir el nuevo compilador en Pascal, ya que así podrá (el compilador) ser compilado en las distintas máquinas por los compiladores Pascal ya existentes.

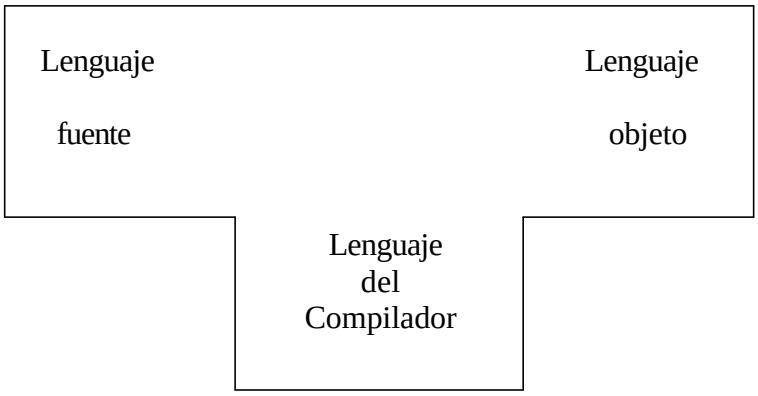
Metacompilador: Es un traductor que tiene como entrada la definición de un lenguaje y como salida el compilador para dicho lenguaje.

Decompilador: Es el que traduce código máquina a lenguaje de alto nivel. Los decompiladores más usuales son los desensambladores, que traducen un programa en lenguaje máquina a otro en ensamblador

Bootstrapping.

Es una técnica muy usada actualmente para el desarrollo de compiladores de lenguajes de alto nivel, en especial si se quiere obtener un autocompilador, o sea un compilador que se compile a sí mismo.

Para describir el proceso de autocompilación se emplea la notación en T que representa gráficamente los tres lenguajes implicados en el proceso de compilación:

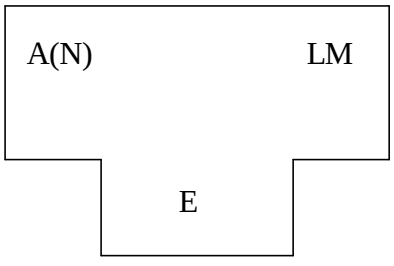


Lenguaje fuente: lenguaje origen que traduce el compilador.

Lenguaje objeto: lenguaje meta, al cual traduce el compilador.

Lenguaje del compilador: lenguaje en el que esta escrito el compilador.

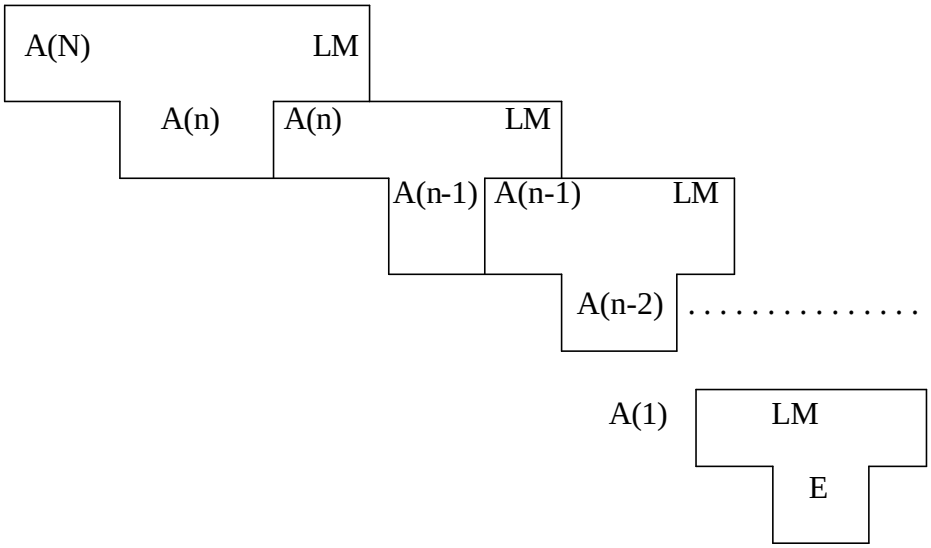
Supongamos que se quiere implementar un nuevo lenguaje A(N) en una máquina determinada. Disponemos solamente de un ensamblador para dicha máquina. En principio parece que la solución es escribir un compilador en lenguaje ensamblador que traduzca desde el lenguaje A(N) al lenguaje máquina LM.



Esto en la práctica resulta bastante complicado dado que programar en ensamblador es muy engorroso.

Lo que se hace en estos casos es desarrollar un lenguaje restringido A(1) parecido al A(N) pero más simple, y para este lenguaje escribir el compilador en ensamblador, o en cualquier otro lenguaje soportado por la máquina.

Una vez construido este compilador, y dado que nuestra máquina es ya capaz de entender el lenguaje A(1), se puede desarrollar un compilador para otro lenguaje A(2) escribiéndolo en el lenguaje A(1), y así sucesivamente hasta llegar a obtener un autocompilador del lenguaje A(N). Esta técnica se conoce como **bootstrapping**.



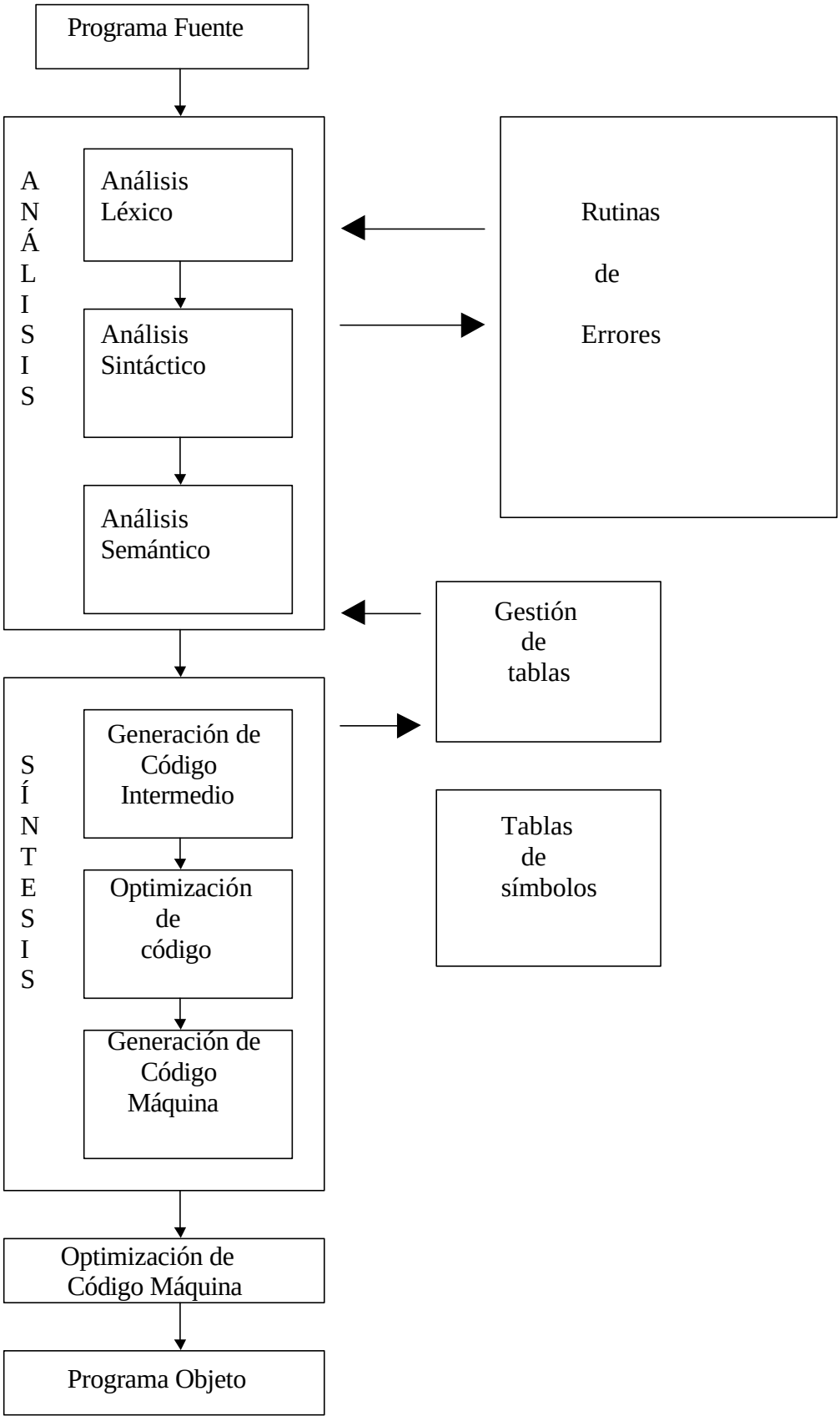
El primer compilador de Pascal desarrollado en Zurich por Wirth fue posible gracias a esta técnica. El conocido compilador C de GNU emplea también este mecanismo en tres pasos.

Estructura de un compilador.

Un compilador es un programa, en el que pueden dirstinguirse dos subprogramas o fases principales: una fase de análisis, en la cual se lee el programa fuente y se estudia la estructura y el significado del mismo; y otra fase de síntesis, en la que se genera el programa objeto.

En un compilador pueden distinguirse además algunas estructuras de datos comunes, la más importante de las cuales es la tabla de símbolos, junto con las funciones de gestión de ésta y de los demás elementos del compilador, y de una serie de rutinas auxiliares para detección de errores.

El esquema general de un compilador podría ser el siguiente:


Programa Fuente
ANÁLISIS
Análisis Léxico
Análisis Sintáctico
Análisis Semántico
Rutinas de Errores
Gestión de tablas
SÍNTESIS
Generación de Código Intermedio
Optimización de código
Generación de Código Máquina
Tablas de símbolos
Optimización de Código Máquina
Programa Objeto

Esquema de un compilador.

Las funciones de estos módulos son las siguientes :

Analizador lexicográfico: Las principales funciones que realiza son:

- ? Identificar los símbolos
- ? Eliminar los blancos, caracteres de fin de línea, etc.
- ? Eliminar los comentarios que acompañan al fuente
- ? Crea unos símbolos intermedios llamados **tokens**
- ? Avisar de los errores que detecte

Ejemplo: A partir de la sentencia en PASCAL siguiente

nuevo := Viejo + RAZON*2

genera un código simplificado para el análisis sintáctico posterior, por ejemplo.

<id1> <:=> <id2> <+> <id3> <*> <ent>

Nota: Cada elemento encerrado entre <> representa un único token. Las abreviaturas id y ent significan identificador y entero respectivamente.

Analizador sintáctico: Comprueba que las sentencias que componen el texto fuente son correctas en el lenguaje, creando una representación interna que corresponde a la sentencia analizada, de esta manera se garantiza que sólo serán procesadas las sentencias que pertenezcan al lenguaje fuente. Durante el análisis sintáctico así como en las demás etapas se van mostrando los errores que se encuentren.

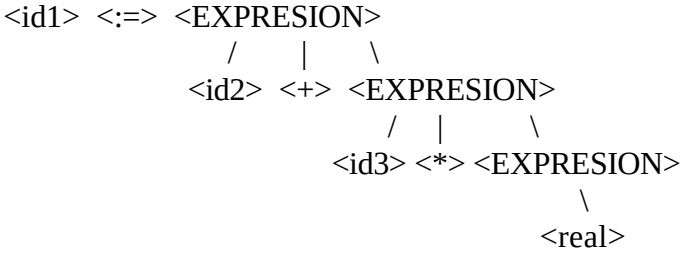
Ejemplo: El esquema de la sentencia anterior corresponde al de una sentencia de asignación del lenguaje Pascal, estas sentencias son de la forma

<id> <:=> <EXPRESION>

y la parte que se denomina <EXPRESION> es de la forma

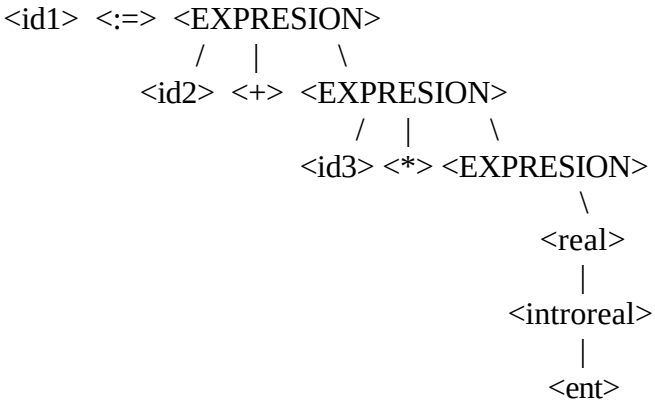
<id> <+> <EXPRESION> o bien
<id> <*> <EXPRESION> o bien
<real>

La estructura de la sentencia queda por tanto de manifiesto mediante el siguiente esquema:



Análisis semántico: Se ocupa de analizar si la sentencia tiene algún significado. Se pueden encontrar sentencias que son sintácticamente correctas pero que no se pueden ejecutar porque carecen de sentido. En general el análisis semántico se hace al par que el análisis sintáctico introduciendo en este unas rutinas semánticas.

Ejemplo: En la sentencia que se ha analizado existe una variable entera, sin embargo las operaciones se realizan entre identificadores reales, por lo que hay dos alternativas: o emitir un mensaje de error "Discordancia de tipos", o realizar una conversión automática al tipo superior, mediante una función auxiliar inttoreal.



Generador de código intermedio: El código intermedio es un código abstracto independiente de la máquina para la que se generara el código objeto que ha de cumplir dos requisitos importantes: ser fácil de producir a partir del análisis sintáctico, y ser fácil de traducir al lenguaje objeto. Esta fase puede no existir si se genera directamente código máquina, pero suele ser conveniente emplearla.

Ejemplo : Consideremos por ejemplo un código intermedio de tercetos, llamado así porque en cada una de sus instrucciones aparecen como máximo tres operandos. La sentencia traducida a este código intermedio quedaría :

```
temp1 := inttoreal (2)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1   := temp3
```

Optimizador de código: A partir de todo lo anterior crea un nuevo código más compacto y eficiente, eliminando por ejemplo sentencias que no se ejecutan nunca,

simplificando expresiones aritméticas, etc. La profundidad con que se realiza esta optimización varía mucho de unos compiladores a otros, en el peor de los casos esta fase se suprime.

Ejemplo: Siguiendo con el ejemplo anterior, es posible evitar la función *inttoreal* mediante el cambio de 2 por 2.0, obviando además una de las operaciones anteriores, quedando por tanto el código optimizado como sigue :

```
temp1 := id3 * 2.0
id1   := id2 + temp1
```

Generador de código: A partir de los análisis anteriores y de las tablas que estos análisis van creando durante su ejecución produce un código o lenguaje objeto que es directamente ejecutable por la máquina. Es la fase final del compilador. Las instrucciones del código intermedio se traducen una a una en código máquina reubicable. (Cada instrucción de código intermedio puede dar lugar a más de una de código máquina)

Ejemplo: El código anterior traducido a ensamblador DLX quedaría:

```
LW R1,id3
MUL R1,R1,2
LW R2,id2
ADD R2,R2,R1
SW id1,R2
```

En donde id1, id2 y id3 representan las posiciones de memoria en las que se hallan almacenadas estas variables; R1,R2 son los registros de la máquina; y las instrucciones LW, SW, MUL y ADD representan las operaciones de colocar un valor de memoria en un registro, colocar un valor de un registro en memoria, multiplicar en coma flotante o sumar.

La tabla de símbolos: Es el medio de almacenamiento de toda la información referente a las variables y objetos en general del programa que se está compilando.

Ejemplo : Hemos visto que en ciertos momentos del proceso de compilación debemos hacer uso de cierta información referente a los identificadores o los números que aparecen en nuestra sentencia, como son su tipo, su posición de almacenamiento en memoria, etc. Esta información es la que se almacena en la tabla de símbolos.

Rutinas de errores : Están incluidas en cada uno de los procesos de compilación (análisis lexicográfico, sintáctico y semántico) y se encargan de informar de los errores que encuentran en texto fuente.

Ejemplo: El analizador semántico podría emitir un error o al menos un aviso cuando detectase una diferencia en los tipos de una operación.