1. LENGUAJES DE PROGRAMACIÓN.

Para que un computador (hardware) funcione es necesario utilizar programas (software), los cuales le indican cuál es la tarea que se tiene que hacer. Un *lenguaje de programación* es el que se utiliza para escribir dichos programas. Posteriormente estos se introducirán en la memoria del computador y éste último ejecutará todas las operaciones que se incluyen.

Los lenguajes de programación constan de:

A) Un conjunto finito de *símbolos*, a partir del cual se define el ***léxico*** o vocabulario del lenguaje.
B) Un conjunto finito de *reglas*, la ***gramática*** del lenguaje, para la construcción de las sentencias "correctas" del lenguaje. (***Sintaxis***).
C) ***Semántica***, que asocia un significado (la acción que debe llevarse a cabo) a cada posible construcción del lenguaje.

Así, podemos decir que un *lenguaje de programación* consta de un *conjunto de símbolos* y un *conjunto de reglas* válidas para componerlos, de forma que formen un mensaje con significado para el computador.

En este capítulo se presenta una visión general de los distintos tipos de lenguajes de programación de computadoras y sus características.

1.1. LENGUAJE MÁQUINA.

Los ordenadores sólo entienden un lenguaje específico para cada máquina, que se denomina *CÓDIGO MÁQUINA* o *Lenguaje Máquina*. Este lenguaje utiliza un código binario (símbolos "0" y "1"). Las órdenes que se dan a un ordenador han de ir codificadas en *instrucciones*, y estas forman los *programas*. Las instrucciones tienen dos partes diferenciadas: código de operación y código(s) de operando(s):

CODOP	CODOPERANDO(S)

En la primera, se codifica la operación que realiza la instrucción. Este código de operación siempre es único para cada instrucción. En la segunda se indica(n) la(s) dirección(es) de memoria en la que se encuentra el operando, hasta un máximo de tres, sobre el/(los) que se aplicará la operación.

Puesto que cada tipo de ordenador tiene su código máquina específico, para programar en este lenguaje el programador debe conocer la arquitectura física de la computadora con cierto detalle (registros de la CPU, palabras de memora,...). La estructura del lenguaje máquina está totalmente adaptada a los circuitos de la computadora y muy alejada del lenguaje que empleamos normalmente para expresar y analizar los problemas que hoy día son resolubles con la computadora. Por ejemplo, para hacer cálculos aritméticos disponemos de un "lenguaje" matemático fácil de comprender y claro, que no se parece en nada al código máquina necesario para hacer dichos cálculos.

Las **ventajas** de los lenguajes máquina son:

- Un programa escrito en lenguaje máquina es directamente interpretable por el procesador central. Una vez introducido el programa en la memoria principal de la computadora, no se necesitan transformaciones previas para ser ejecutado (como más adelante veremos que sí ocurre con los programas escritos en lenguajes de alto nivel).

- Los programas escritos en lenguaje máquina se ejecutan muy eficientemente (con rapidez), debido a que el usuario lo redacta específicamente para los circuitos que lo han de interpretar y ejecutar, y a que desde el código máquina se puede utilizar la totalidad de los recursos de la máquina.

Por contra, los lenguajes máquina tienen las siguientes características-**inconvenientes**:

- Las instrucciones son cadenas de ceros y unos, aunque estas cadenas se pueden introducir en la computadora mediante un código intermedio (octal o hexadecimal).

- Los datos se utilizan por medio de las direcciones de memoria donde se encuentran. En las instrucciones no aparecen nombres de variables (i, j aux...), sino que el programador debe *asignar* las direcciones de memoria para las variables y constantes del programa. Para realizar esta asignación se debe tener en cuenta la zona de memoria que ocupa el programa, para que no se solape con la zona en la que se almacenan las variables.

- El repertorio de instrucciones suele ser muy reducido y las instrucciones realizan operaciones muy simples.

En lenguaje máquina, hay varios tipos de instrucciones:

- De transferencia de información.
- De tratamiento o aritmético-lógicas y de desplazamiento
- De transferencias de control: bifurcaciones, saltos, llamadas a procedimientos y retornos de procedimientos.

Muchas computadoras, por ejemplo, no disponen de instrucciones específicas de multiplicar y dividir; en su lugar, el programador debe ingeniárselas para descomponer cada una de las operaciones que desee realizar en términos de las instrucciones elementales del repertorio máquina. Por ejemplo, para multiplicar, se deberá codificar un algoritmo que realice la multiplicación por medio de sumas, comparaciones, desplazamientos, etcétera.

- Existe muy poca elasticidad y versatilidad para la redacción de instrucciones. Estas tienen un formato rígido en cuanto a posición de los distintos campos que configuran la instrucción (código de operación, dirección o direcciones de memoria, códigos de puertos, etc.). El código de operación debe seleccionarse estrictamente entre los que figuran en una tabla o repertorio fijo. Además, un programa máquina no permite el uso de sentencias declarativas, existiendo sólo las instrucciones.

- El lenguaje máquina depende y está ligado íntimamente a la CPU del computador. Si dos computadoras tienen CPU's diferentes, tendrán distintos lenguajes máquina. En particular, dos microcomputadoras con el mismo microprocesador e iguales circuitos de control, tienen igual lenguaje máquina. La dependencia del lenguaje máquina de la configuración de la CPU hace que los programas redactados en este lenguaje de programación sean poco *transferibles* o *transportables* de una computadora a otra.

- En un programa en código máquina, no pueden incluirse comentarios que faciliten la legibilidad del mismo. Además, debido a su representación totalmente numérica, es muy difícil de reconocer o interpretar por el usuario.

1.2. TRADUCTORES.

Para facilitar el trabajo de programación y hacer los programas más legibles, resolviendo las limitaciones que presentan los lenguajes máquina, se han desarrollado lenguajes de programación

que permiten utilizar una simbología y terminología próximas a las tradicionalmente utilizadas en la descripción de problemas. Estos lenguajes se denominan genéricamente *lenguajes simbólicos*.

Como la computadora únicamente puede interpretar y ejecutar código máquina, existen programas *traductores*, que traducen o trasladan programas desde el lenguaje simbólico al lenguaje máquina. En general existen traductores para convertir programas escritos en un lenguaje a otro. El programa inicial se denomina *programa fuente*, y el programa obtenido tras el proceso de traducción *programa objeto*. La figura 5.1 ilustra el mecanismo: el programa fuente es utilizado por el traductor como conjunto de datos que debe procesar, y a partir del mismo genera como resultado el programa objeto.

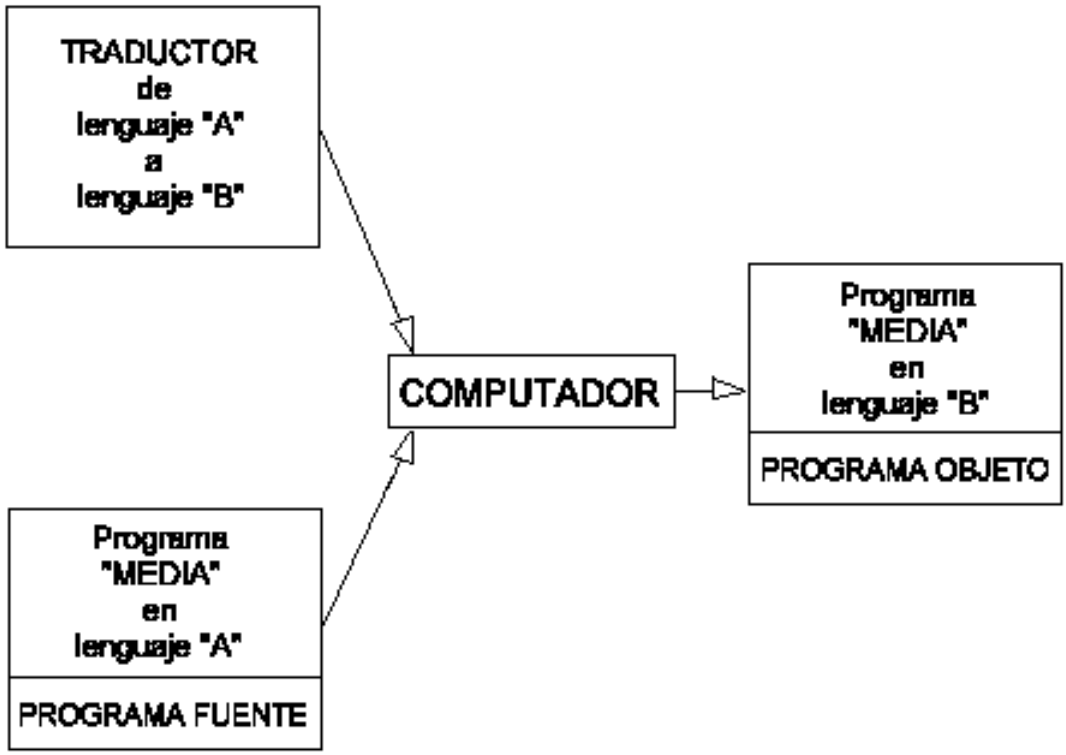

Figura 5.1 Proceso de Traducción de un Programa

Existen dos tipos claramente diferenciados de lenguajes que necesitan de un traductor: *los lenguajes ensambladores y los lenguajes de alto nivel*.

1.2.1. Ensambladores y macroensambladores.

Los lenguajes ensambladores permiten al programador:

- Escribir las instrucciones utilizando, en vez de códigos binarios o intermedios, una notación simbólica o mnemotécnica para representar los códigos de operación. Normalmente los códigos mnemotécnicos están constituidos por tres o cuatro letras que, en forma abreviada, indican la operación a realizar. Usualmente, debido al origen anglosajón de los fabricantes de computadoras, los mnemotécnicos son abreviaturas en inglés. Por ejemplo: la Suma se suele representar como ADD (suma en inglés), SUB (sustracción), MOV (movimiento), NOT (negación lógica), CALL (llamada a un procedimiento), RET (retorno de procedimiento), etc.

- Utilizar, para los datos, direcciones simbólicas de memoria en lugar de direcciones binarias absolutas. Así, por ejemplo, los datos pueden ser referenciados mediante identificadores, tales como

I, SUMA, PROD, CONT, AUX, etc. Existen sentencias declarativas (también denominadas seudoinstrucciones o directivas) para indicar al traductor la correspondencia entre direcciones simbólicas y direcciones de memoria. Con estas seudoinstruciones, el traductor crea una tabla con cuya ayuda, al generar las instrucciones máquina, sustituye las direcciones simbólicas por las direcciones binarias correspondientes.

- Las instrucciones escritas en este lenguaje, guardan una estrecha relación con las instrucciones del lenguaje máquina en que posteriormente serán traducidas. Este tipo de lenguajes hace corresponder a cada instrucción en ensamblador una instrucción en código máquina.

- Incluir líneas de comentarios entre las líneas de instrucciones. El traductor las elimina automáticamente, no incluyéndolas en el código máquina que genera.

Como ejemplo, veamos un programa sencillo para el microprocesador Motorola 6800. Realiza la operación de suma que describe la siguiente operación en seudocódigo:

AUX1<---- CONT + 1

Instrucción		Comentario
AUX1	EQU 5	Iguala el símbolo AUX1a la posición de memoria principal de dirección 5.
CONT	EQU 6	Iguala el símbolo CONT a la posición de memoria principal de dirección 10.
LDA	A CONT	Copia el contenido de la posición de memoria M[CONT] en el acumulador A
ADD	A # 1	Sumar el número 1 al contenido de A y almacenar el resultado en A.
STA	A AUX	Copiar el contenido de A en la posición de memoria principal M[AUX1]

Un programa en ensamblador no puede ejecutarse directamente por la computadora, siendo necesario ser traducido (ensamblado) previamente, según el esquema de traducción que hemos visto antes.

El traductor de lenguaje ensamblador a lenguaje máquina se denomina *ensamblador*. El ensamblador mejora o resuelve algunos de los problemas de los lenguajes máquina (como son el que las instrucciones sean cadenas de 1's y 0's, que no existen nombres de variables sino direcciones de memoria, o que no permiten comentarios), pero siguen persistiendo otras limitaciones (repertorio de instrucciones reducido, poca elasticidad para la redacción de instrucciones, o que está íntimamente ligado a la CPU de la computadora).

Hay unos lenguajes evolucionados de los ensambladores, que se denominan *macroensambladores*. Con ellos se solventa en cierta medida la limitación de tener un repertorio de instrucciones muy reducido. Los lenguajes macro ensambladores disponen de *macroinstrrucciones*, como por ejemplo transferir un bloque de datos de memoria principal a disco, multiplicar, dividir, etc. La macroinstrucción es una llamada a un módulo o rutina, llamada *macro*, que el traductor inserta, antes de realizar el proceso de generación del código máquina definitivo, en el lugar de la llamada correspondiente. Obviamente, a cada macroinstrucción, a diferencia de las instrucciones, le corresponden varias instrucciones máquina y no sólo una.

1.2.2. Lenguajes de alto nivel: Compiladores e intérpretes.

Los lenguajes ensambladores se suelen denominar lenguajes de segunda generación, en contraposición a los lenguajes máquina, que fueron los de la primera generación. Los *lenguajes de alto nivel* se consideran lenguajes de tercera generación.

Los lenguajes de alto nivel no obligan al usuario a conocer los detalles de la computadora que utiliza. Con estos lenguajes las operaciones se expresan con sentencias o frases muy parecidas

al lenguaje matemático o al lenguaje natural, utilizados habitualmente por las personas (igual que para los ensambladores, predomina el uso de palabras o términos en inglés).

Las características de los lenguajes de alto nivel son:

- Las instrucciones se expresan por medio de caracteres alfabéticos, numéricos y caracteres especiales (+, =, /, etc.).

- El usuario puede definir las variables que desee, dándoles los nombres que considere oportuno (TOTAL, CONTADOR, RESULTADO, etc.), siendo las reglas para denominación de las mismas muy poco restrictivas. La asignación de memoria para variables y constantes las hace directamente el traductor.

- El repertorio de instrucciones es muy amplio, conteniendo operadores y funciones de una gran diversidad: aritméticas, especiales (seno, coseno, módulo, etc.), lógicas (comparaciones, operaciones del álgebra de Boole, etc.), de tratamiento de caracteres (como buscar una subcadena en una cadena de caracteres), etc..

- El programador puede definir sus instrucciones con una gran versatilidad, siendo las reglas o gramáticas de los lenguajes muy abiertas.

- Los lenguajes de alto nivel apenas dependen de la máquina. Hay lenguajes más trasladables que otros. Por ejemplo, al existir muchos dialectos del BASIC, con frecuencia es necesario adaptar partes de los programas para pasarlos de una computadora a otra. Hay lenguajes casi "independientes de la computadora". En general, cuanto mejor definido (sin ambigüedades) y diseñado esté un lenguaje, menos dependiente de la máquina será. Entre sus actividades, el American National Standard Institute (ANSI) se encarga de realizar normalizaciones de lenguajes para garantizar la translabilidad de los programas.

- Pueden incluirse comentarios en las líneas de instrucciones, o puede haber líneas específicas de comentarios. Esto facilita la legibilidad de los programas, tanto para el propio programador, como para otras personas.

- Un programa escrito en un lenguaje de alto nivel no puede ser directamente interpretado por la computadora, siendo necesario realizar previamente su traducción a lenguaje máquina. Usualmente la traducción se hace en dos etapas: primero a ensamblador, y posteriormente a código máquina. Por lo general, una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en ensamblador o lenguaje máquina.

Existen dos tipos de traductores para los lenguajes de alto nivel:

Los *compiladores* traducen el código fuente a código objeto, para todo el programa a la vez (todo el programa completo). A su vez llevan a cabo *optimizaciones* del programa (como eliminar variables o constantes no utilizadas, o ciclos inútiles) que permiten que el programa ocupe menos espacio o sea más rápido.

Los *intérpretes* traducen el código fuente línea por línea, sin generar programa objeto, y traduciendo las instrucciones en comandos para el hardware. Son más lentos que los compiladores, puesto que tienen que interpretar una línea cada vez que pasan por ella (en los compiladores cada línea solo se traduce una vez). Los lenguajes *semicompilados* son intérpretes que traducen cada línea a un lenguaje intermedio, mejor comprensible por la máquina (por ejemplo, pasan los números a representación interna, o sustituyen los números de línea por su posición de memoria correspondiente).

1.2.2.1. Compiladores.

Un compilador traduce un programa fuente, escrito en un lenguaje de alto nivel, a un programa objeto, escrito en lenguaje ensamblador o máquina. El programa fuente suele estar contenido en un fichero, y el programa objeto pasa a ocupar otro fichero. El fichero objeto puede almacenarse en memoria masiva para ser procesado posteriormente.

La traducción por un compilador (la compilación) consta de dos etapas fundamentales, que a veces no están claramente diferenciadas a lo largo del proceso: la etapa de análisis del programa fuente y la etapa de síntesis del programa objeto; es decir, qué pretende realizar el programa fuente, y cómo realizarlo en el programa objeto. A su vez, cada una de estas etapas conlleva la realización de varias fases, y en cada una de las cuales se recorre o analiza completamente el programa fuente. Las fases del proceso de compilación se pueden ver en la Figura 5.2.

Figura 5.2. Fases del Proceso de Compilación

a) Análisis lexicográfico.

Consiste en descomponer el programa fuente en sus elementos constituyentes, es decir, sus símbolos, que son caracteres o secuencias de caracteres con significado especial. En cierto sentido, identifica las palabras del lenguaje. De esta forma, el analizador léxico (también denominado escáner) aísla los símbolos, identifica su tipo y almacena en las tablas de símbolos la información del símbolo que pueda ser necesaria durante el proceso de traducción. La representación obtenida en esta fase contiene la misma información que el programa fuente, pero de forma más compacta.

b) Análisis sintáctico.

La sintaxis de los lenguajes de programación se especifica mediante un conjunto de reglas (la gramática del lenguaje). Esta fase deberá comprobar si un programa es sintácticamente correcto, es decir, si sus estructuras (expresiones, sentencias o asignaciones) están construidas de acuerdo con las reglas del lenguaje.

c) Análisis semántico.

La semántica de un lenguaje de programación define el significado dado a las distintas construcciones sintácticas. En los lenguajes de programación, el significado está ligado a la estructura sintáctica de las sentencias.

En el proceso de traducción, el significado de las sentencias se obtiene de la identificación sintáctica de las construcciones sintácticas y de la información almacenada en la tabla de símbolos.

d) Generación de Código Intermedio.

Si no se han producido errores en algunas de las etapas anteriores, este módulo realiza la traducción a un código interno propio del compilador, denominado Código Intermedio, a fin de permitir la transportabilidad del lenguaje a otros ordenadores.

e) Optimizaciones.

En la fase de optimización se mejora el código intermedio generado anteriormente, analizando el programa de forma global. Un ejemplo de optimización sería encontrar en el código la inicialización de una constante dentro de un bucle, con lo que esta inicialización se produciría tantas veces como se repitiese el bucle. El optimizador sacaría esta inicialización de la constante fuera del bucle para que sólo se realizase una vez.

f) Generación de Código Objeto.

En esta etapa se genera el código objeto final. En algunos casos, este código es directamente ejecutable, y en otros necesita algunos pasos previos a la ejecución (ensamblado, encuadernación y carga).

Para un determinado lenguaje de alto nivel, se hace común todo el proceso de análisis y generación de código intermedio. Y es la generación del código objeto la que se particulariza para cada tipo de microprocesador.

La compilación es un proceso complejo y que consume a veces un tiempo muy superior a la propia ejecución del programa. En cualquiera de las fases de análisis el compilador puede dar mensajes sobre los errores que detecta en el programa fuente, cancelando en ocasiones la compilación para que el usuario realice en el fichero las correcciones oportunas.

Existen compiladores que permiten al usuario omitir o reducir las fases de optimización, disminuyéndose así el tiempo global de la compilación.

1.2.2.2. Intérpretes.

Un intérprete hace que un programa fuente escrito en un lenguaje vaya, sentencia a sentencia, traduciéndose a código objeto y sea ejecutado directamente por la computadora. El intérprete capta una sentencia fuente y la traduce, expandiéndola en una o varias instrucciones máquina, que ejecuta inmediatamente, no creándose, por tanto, un fichero o programa objeto almacenable en memoria masiva para posteriores ejecuciones.

En la práctica el usuario crea un fichero con el programa fuente. Esto suele realizarse con un editor específico del propio intérprete del lenguaje. Según se van almacenando las instrucciones simbólicas, se analizan y se producen los mensajes de error correspondientes: así el usuario puede proceder inmediatamente a su corrección. Una vez creado el fichero fuente el usuario puede dar la orden de ejecución (“RUN”) y el intérprete lo ejecuta línea a línea. Siempre la traducción antecede inmediatamente a la ejecución, de forma que:

- Las optimizaciones sólo se realizan dentro del contexto de cada sentencia, no contemplándose el programa o sus estructuras en conjunto. Si por ejemplo, en seudocódigo, la instrucción que debe traducir es:

A ß 2.0 * B + C

al traducirse, antes de ejecutarse, podríamos obtener:

```
aux1 ß conv_a_real(2)
aux2 ß B * aux1
aux3 ß C + aux2
A ß aux3
```

Pero se podrían realizar los mismos cálculos usando sólo dos instrucciones

aux1 ß B * 2.0
A ß aux1 + C

- Si una sentencia forma parte de un bucle, se traduce tantas veces como tenga que ejecutarse el bucle, y no una sola vez como ocurriría en un compilador.

- Cada vez que utilicemos un programa tenemos que volver a traducirlo, ya que en la traducción no se genera un fichero objeto que poder guardar en memoria masiva (y utilizarlo en cada ejecución). Con un compilador, aunque la traducción es más lenta, esta sólo debe realizarse una vez (ya depurado el programa): cuando deseemos ejecutar un programa ejecutamos el fichero objeto, que se tradujo previamente.

Los intérpretes, a pesar de los inconvenientes anteriores, son preferibles a los compiladores cuando el número de veces que se va a ejecutar el programa es muy bajo. Además con ellos es más fácil desarrollar programas. Esto es así porque un error en una sentencia en ejecución puede corregirse inmediatamente sobre el programa fuente. Con un compilador no sucede lo mismo, ya que es difícil, una vez producido un error en una instrucción máquina, localizar la sentencia fuente, origen de la instrucción errónea. Una vez localizadas las instrucciones erróneas del programa fuente, el programador puede corregirlas sobre el fichero fuente (no sobre el objeto), y volver a compilarlo en su totalidad; y después, en su caso, ensamblarlo, encuadernarlo y cargarlo (este proceso lo veremos más adelante).

Obviamente los lenguajes intérpretes resultan más pedagógicos para aprender a programar, ya que el alumno puede detectar y corregir más fácilmente sus errores. Otra ventaja adicional de los traductores-intérpretes es que ocupan, por lo general, menos memoria que los compiladores (por ejemplo, en los primeros PC el BASIC se encontraba grabado en ROM).

Existen lenguajes cuyos traductores se idearon como intérpretes (BASIC, LISP, ...) Y otros como compiladores (FORTRAN, C, ADA...). No obstante en la actualidad, para un lenguaje dado pueden existir tanto compiladores como intérpretes.

1.3. FASES EN LA CODIFICACIÓN, EJECUCIÓN Y DEPURACIÓN DE UN PROGRAMA ESCRITO EN UN LENGUAJE DE ALTO NIVEL.

Para ejecutar un trabajo (programa principal con sus subrutinas) es necesario realizar una serie de operaciones en las que la propia computadora actúa facilitando herramientas y ayudas, que genéricamente se conocen como *Utilidades*.

A continuación, detallamos estas operaciones.

1.3.1. Escritura o tecleo.

Una vez redactado el programa, debe introducirse en la computadora. Esto, por lo general, se realiza creando un fichero en memoria masiva (disco), con ayuda de un *editor de textos*.

El editor de textos es un programa de utilidad que nos permite cómodamente introducir y modificar (borrar, intercalar, cambiar, ampliar, duplicar, etc.) información (de programas o datos) en un fichero. Podríamos decir que esta fase es la introducción y corrección “mecanográfica” del programa.

1.3.2. Compilación.

Cuando el fichero está creado (supongamos que con el nombre de MEDIAS), pasamos a compilar el programa. Con ello obtenemos el mismo programa en lenguaje ensamblador (suponemos que este compilador no genera directamente código máquina). A continuación se ensambla, generándose un nuevo fichero ("MEDIAS.OBJ" por ejemplo), que contiene el programa en lenguaje máquina. Este programa se suele denominar relocalizable, y aún no es directamente ejecutable.

1.3.3. Enlazado.

La siguiente fase consiste en unir las llamadas a diversas rutinas o módulos del propio usuario o del sistema (funciones de librería, etc.) al programa principal. Todos estos módulos, antes de unirse, deben estar compilados y ensamblados. Para efectuar la unión, denominada encuadernación o colección, se utiliza una utilidad, que genera un nuevo fichero (por ejemplo de nombre "MEDIAS.PR") denominado absoluto, y que puede ser ejecutado directamente.

1.3.4. Carga del programa en memoria principal.

La fase siguiente es introducir o cargar el programa absoluto en memoria, y prepararlo para su ejecución. Estas operaciones las realiza una utilidad denominada *cargador*. Tradicionalmente las funciones que se asignan a un cargador son las siguientes:

- Asignar espacio en la memoria principal para el programa. Obviamente esta asignación debe realizarse en espacios libres.

- Cada programa o módulo relocalizable puede hacer llamadas a otros módulos del programa absoluto. Es necesario ajustar las direcciones de las instrucciones de llamada, de forma que estas queden definidas. Esto no es posible antes de la creación del módulo absoluto, ya que cada módulo relativo no puede conocer a priori dónde se encontrarán los módulos y sus instrucciones. Una de las funciones de los cargadores tradicionales es realizar estos enlaces.

- El programa absoluto tiene sus direcciones numeradas de 0 a n (son direcciones relativas). El cargador, teniendo en cuenta las zonas de memoria asignadas (que no necesariamente han de ser contiguas) reajusta las direcciones para que correspondan a direcciones físicas reales.

- Pasa las instrucciones de memoria masiva a memoria principal; es decir, realiza la operación de carga propiamente dicha.

En los sistemas actuales las cuatro funciones anteriores pueden estar dispersas en diversos módulos y no pertenecer todas al cargador.

1.3.5. Ejecución.

La siguiente fase es la ejecución del programa. Para ello el monitor del sistema operativo pasa el control de la CPU a la dirección de la palabra de memoria donde se encuentra la primera instrucción del programa (es decir, carga en el contador de programa la dirección física de dicha instrucción).

1.3.6. Depuración de errores.

Si el programa tiene errores o se desea optimizarlo, pueden usarse utilidades de rastreo y depuración de errores. Estas permiten efectuar funciones tales como ejecutar el programa instrucción a instrucción, mostrándose después de cada ejecución el contenido de las variables que van cambiando.

1.4. CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN.

Tradicionalmente los lenguajes de programación se pueden clasificar atendiendo a varios factores:

- Según su *"nivel"*. Hace referencia a lo próxima al hombre que esté la forma de expresar las sentencias:
  * Lenguajes de bajo nivel y ensambladores (cercanos a la máquina).
  * Lenguajes de alto nivel (cercanos al hombre).

- Según la *relación traducción-ejecución*.
  * Compiladores.
  * Intérpretes.

- Según su *campo de aplicación*:
  * Aplicaciones Científicas. Predominan los algoritmos de cálculo numérico y matrices.
  * Aplicaciones de Procesamiento de Datos. Sobresalen las tareas relativas a la creación, mantenimiento, consulta y listado de datos. Estos datos se organizan en registros, ficheros y bases de datos.
  * Aplicaciones de Tratamiento de Textos. Llevan a cabo la manipulación de textos en lenguaje natural.
  * Aplicaciones en Inteligencia Artificial. Están constituidas por programas que emulan un comportamiento inteligente. Ej. juegos inteligentes (ajedrez, tres en raya, ...), robótica, sistemas expertos, ...
  * Aplicaciones de Programación de Sistemas. Como por ejemplo aquéllos que se utilizan para desarrollar los módulos de un Sistema Operativo, traductores de lenguajes, ...

- Según el *estilo de programación*:
  * Imperativos.
  * Declarativos.

Nos centramos en esta última clasificación.

1.4.1. Lenguajes imperativos.

Los lenguajes imperativos o procedurales se basan en la asignación de valores. Se fundamentan en la utilización de variables para almacenar valores y en la realización de operaciones con los datos almacenados. La mayoría de los lenguajes son de este tipo (FORTRAN, BASIC, COBOL, PASCAL, C, etc.).

Son los que ya hemos analizado:

a) *Lenguajes de alto nivel*, caracterizados por estar enfocados a la resolución de problemas en campos de aplicación específicos y los programas escritos en ellos ser fácilmente trasladables de una a otra computadora.

b) *Lenguajes ensambladores y máquina*, totalmente adaptados y predeterminados por la CPU de la máquina.

1.4.2. Lenguajes declarativos.

Están basados en la definición de funciones o relaciones. No utilizan instrucciones de asignación (sus variables no almacenan valores). Son los más fáciles de utilizar (no se requieren conocimientos específicos de informática), están muy próximos al hombre. Se suelen denominar también lenguajes de órdenes, ya que los programas están formados por sentencias que ordenan "qué es lo que se quiere hacer", no teniendo el programador que indicar a la computadora el proceso detallado (el algoritmo) de cómo hacerlo".

En este grupo se incluyen ciertos lenguajes especializados en funciones tales como recuperación de la información en bases de datos (NATURAL e IMS), análisis de circuitos electrónicos (SPICE), y realización de cálculos estadísticos (BMDP, SPSS, SAS, etc.).

Se dividen en lenguajes funcionales y lógicos.

1.4.2.1. Lenguajes funcionales.

Los lenguajes funcionales son un tipo de lenguajes declarativos, en los que los programas están formados por una serie de definiciones de funciones. Ejemplos de estos lenguajes son el LISP y el SCHEME. Se suelen aplicar a problemas de Inteligencia Artificial.

1.4.2.2. Lenguajes lógicos.

Los lenguajes lógicos son el otro tipo de lenguajes declarativos, y en ellos los programas están formados por una serie de definiciones de predicados. También se les denomina lenguajes de programación lógica, y el mayor exponente es el lenguaje PROLOG. Se aplican sobre todo en la resolución de problemas de Inteligencia Artificial.