FUNDAMENTOS DE PROGRAMACIÓN

TAPAICELA CALUGUILLIN AMANDA CAROLINA

CAPÍTULO 1

Introducción a la ciencia de la computación y a la programación

Contenido

1.1. ¿Qué es una computadora?
1.2. Organización física de una computadora (hardware)
1.3. Representación de la información en las computadoras
1.4. Concepto de algoritmo
1.5. Programación estructurada
1.6. Programación orientada a objetos
1.7. El *software* (los programas)
1.8. Sistema operativo
1.9. Lenguajes de programación
1.10. C: El origen de C++ como lenguaje universal
1.11. El lenguaje C++: Historia y características
1.12. El lenguaje unificado de modelado UML 2.0
REFERENCIAS BIBLIOGRÁFICAS Y LECTURAS RECOMENDADAS

INTRODUCCIÓN

Las computadoras electrónicas modernas son uno de los productos más importantes de los siglos XX y XXI y especialmente la actual década. Son una herramienta esencial en muchas áreas: industria, gobierno, ciencia, educación..., en realidad en casi todos los campos de nuestras vidas. El papel de los programas de computadoras es esencial; sin una lista de instrucciones a seguir, la computadora es virtualmente inútil. Los lenguajes de programación nos permiten escribir esos programas y por consiguiente comunicarnos con las computadoras.

En esta obra, usted comenzará a estudiar la ciencia de la computación o informática a través de uno de los lenguajes de programación más versátiles disponibles hoy día: el lenguaje C++. Este capítulo le introduce a la computadora y sus componentes, así como a los lenguajes de programación, y a la metodología a seguir para la resolución de problemas con computadoras y con una herramienta denominada C++.

En el capítulo se describirá el concepto y organización física (*hardware*) y lógica (*software*) de una computadora junto con las formas diferentes de representación de la información. El concepto de algoritmo como herramienta de resolución de problemas es otro de los temas que se abordan en el capítulo.

Las dos paradigmas más populares y que soporta el lenguaje de programación C++ son: *programación estructurada* y *programación orientada a objetos*. Junto con las características de los diferentes tipos de software —en particular el sistema operativo— y de los lenguajes de programación y, en particular, C++ y UML 2.0 se articula la segunda parte del contenido del capítulo.

CONCEPTOS CLAVE

- Algoritmo.
- CD-ROM, CDR/W.
- Compilador.
- Computadora.
- Diagrama de flujo.
- Diagrama N-S.
- Disquete.
- DVD.
- DVD alta definición.
- *Hardware*.
- Intérprete.
- Lenguaje de máquina.
- Lenguaje de programación.
- Lenguaje ensamblador.
- Memoria.
- Memoria auxiliar.
- Memoria central.
- Microprocesador.
- Módem.
- *Software*.
- Unidad central de proceso.

1.1. ¿QUÉ ES UNA COMPUTADORA?

Una **computadora**[1] es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora por la **entrada** (*input*) y a continuación se procesan para producir una **salida** (*output*, resultados), como se observa en la Figura 1.1. La computadora se puede considerar como una unidad en la que se ponen ciertos datos, *entrada de datos*, procesa estos datos y produce unos *datos de salida*. Los datos de entrada y los datos de salida pueden ser realmente cualquier cosa, texto, dibujos o sonido. El sistema más sencillo de comunicarse una per- sona con la computadora es esencialmente mediante un ratón (*mouse*), un teclado y una pantalla (moni-tor). Hoy día existen otros dispositivos muy populares tales como escáneres, micrófonos, altavoces, cá-maras de vídeo, cámaras digitales, etc.; de igual manera, mediante *módems*, es posible conectar su computadora con otras computadoras a través de redes, siendo la más importante, la red **Internet**.

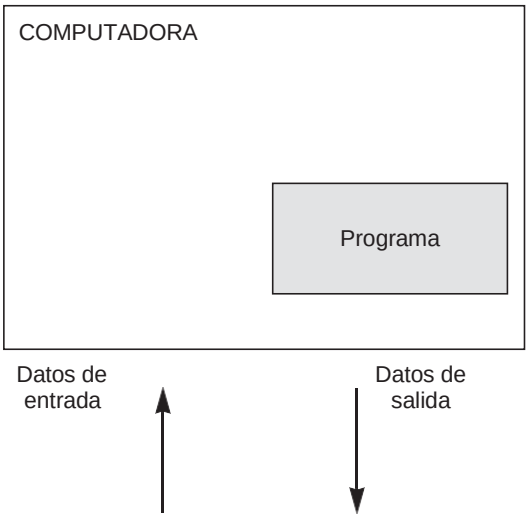


Figura 1.1. Proceso de información en una computadora.

Los componentes físicos que constituyen la computadora, junto con los dispositivos que realizan las tareas de entrada y salida, se conocen con el término ***hardware***. El conjunto de instrucciones que hacen funcionar a la computadora se denomina ***programa***, que se encuentra almacenado en su memoria; a la persona que escribe programas se llama ***programador*** y al conjunto de programas escritos para una computadora se llama ***software***. Este libro se dedicará casi exclusivamente al *software*, pero se hará una breve revisión del *hardware* como recordatorio o introducción según sean los conocimientos del lector en esta materia. En el Anexo A de la página oficial del libro `(www.mhe.es/joyanes)` puede encontrar una amplia información de "Introducción a las computadoras", si desea ampliar este apartado.

1.2. ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA (HARDWARE)

La mayoría de las computadoras, grandes o pequeñas, están organizadas como se muestra en la Figura 1.2. Constan fundamentalmente de tres componentes principales: *Unidad Central de Proceso* (**UCP**) o *procesador* (compuesta de la **UAL**, **U**nidad **A**ritmética y **L**ógica, y la **UC**, **U**nidad de **C**ontrol); la *memoria principal* o *central* y el *programa*.

[1] En España está muy extendido el término **ordenador** para referirse a la traducción de la palabra inglesa *computer*.

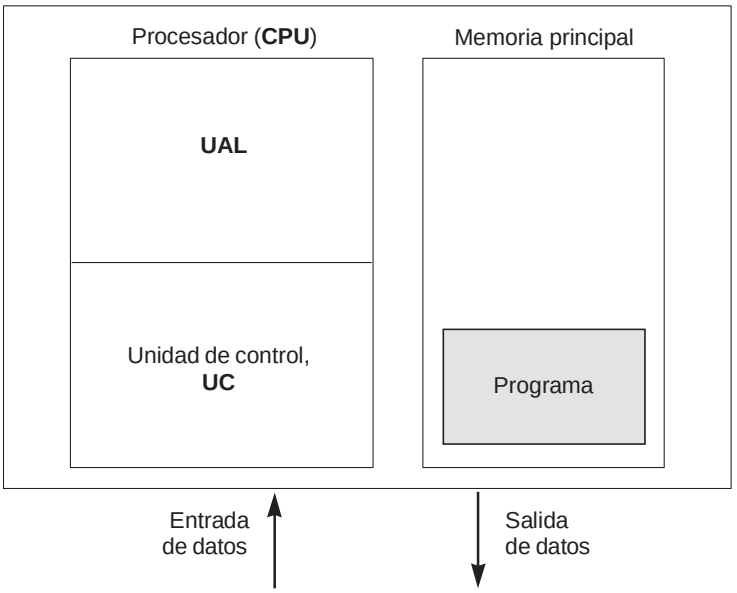


Figura 1.2. Organización física de una computadora.

Si a la organización física de la Figura 1.2 se le añaden los dispositivos para comunicación con la computadora, aparece la estructura típica de un sistema de computadora: dispositivos de entrada, dispositivos de salida, memoria externa y el procesador/memoria central con su programa (Fig.1.3).

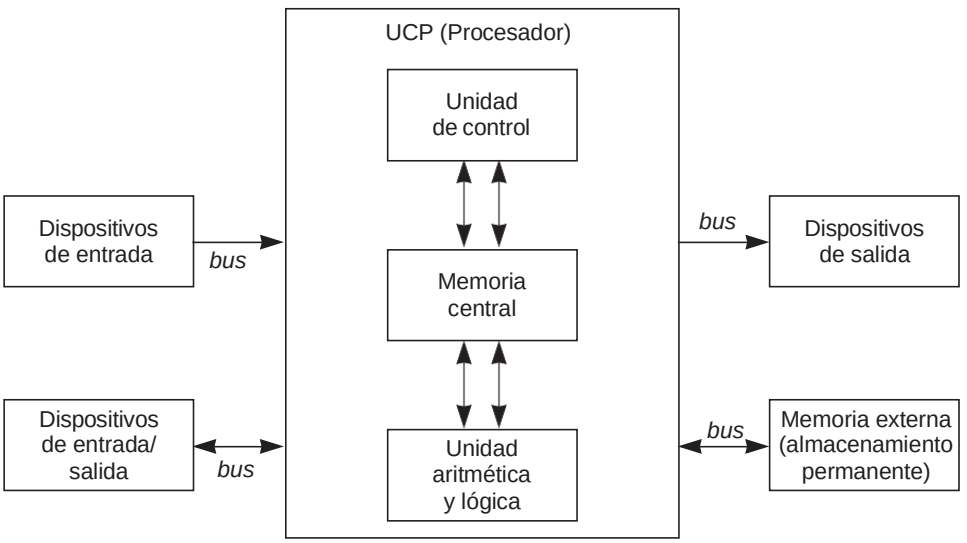


Figura 1.3. Organización física de una computadora.

1.2.1. Dispositivos de Entrada/Salida (E/S)

Los dispositivos de *Entrada/Salida* (**E/S**) (en inglés, *Input/Output* **I/O**) o *periféricos* permiten la comunicación entre la computadora y el usuario.

Los dispositivos de entrada, como su nombre indica, sirven para introducir datos (información) en la computadora para su proceso. Los datos se *leen* de los dispositivos de entrada y se almacenan en la memoria central o interna. Los dispositivos de entrada convierten la información de entrada en señales eléctricas que se almacenan en la memoria central. Dispositivos de entrada típicos son **teclados**, **lectores de tarjetas perforadas** —ya en desuso—, **lápices ópticos**, **palancas de mando** (*joystick*), **lectores de códigos de barras**, **escáneres**, **micrófonos**, **lectores de tarjetas digitales**, **lectores RFID** (tarjetas de identificación por radio frecuencia), etc. Hoy día tal vez el dispositivo de entrada más popular es el **ratón** (***mouse***) que mueve un puntero gráfico (electrónico) sobre la pantalla que facilita la interacción usuario-máquina[2].

Los *dispositivos de salida* permiten representar los resultados (salida) del proceso de los datos. El dispositivo de salida típico es la **pantalla (CRT)**[3] o **monitor**. Otros dispositivos de salida son: **impresoras** (imprimen resultados en papel), **trazadores gráficos** (*plotters*), **reconocedores** *(sintetizadores)* **de voz**, **altavoces**, etc.

Dispositivos de entrada/salida y *dispositivos de almacenamiento* masivo o auxiliar (memoria externa) son: unidad de discos (disquetes, **CD-ROM**, **DVD**, cintas, discos duros, etc.), videocámaras, memorias flash, **USB**, etc.

Figura 1.4. Dispositivo de salida (Impresora HP Color LaserJet 2600n).

1.2.2. La memoria central (interna)

La **memoria central** o simplemente **memoria** (*interna o principal*) se utiliza para almacenar información (**RAM**, **R**andom, **A**ccess **M**emory). En general, la información almacenada en memoria puede ser de dos tipos: *instrucciones*, de un programa y *datos* con los que operan las instrucciones. Por ejemplo, para que un programa se pueda *ejecutar* (correr, rodar, funcionar..., en inglés, *run*), debe ser situado en la memoria central, en una operación denominada *carga* (*load*) del programa. Después, cuando se ejecuta (se realiza, funciona) el programa, cualquier dato a procesar por el programa se debe llevar a la memoria mediante las instrucciones del programa. En la memoria central, hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta a fin de poder funcionar.

[2] Todas las acciones a realizar por el usuario se realizarán con el ratón con la excepción de las que se requieren de la escritura de datos por teclado.

[3] *Cathode Ray Tube:* Tubo de rayos catódicos.

Ejecución

Cuando un programa se ejecuta (realiza, funciona) en una computadora, se dice que se *ejecuta*.

Con el objetivo de que el procesador pueda obtener los datos de la memoria central más rápidamente, normalmente todos los procesadores actuales (muy rápidos) utilizan una *memoria* denominada *caché* que sirve para almacenamiento intermedio de datos entre el procesador y la memoria principal. La memoria *caché* —en la actualidad— se incorpora casi siempre al procesador.

Organización de la memoria

La memoria central de una computadora es una zona de almacenamiento organizada en centenares o millares de unidades de almacenamiento individual o celdas. La memoria central consta de un conjunto de *celdas de memoria* (estas celdas o posiciones de memoria se denominan también *palabras*, aunque no "guardan" analogía con las palabras del lenguaje). El número de celdas de memoria de la memoria central, dependiendo del tipo y modelo de computadora; hoy día el número suele ser millones (512, 1.024, etc.). Cada celda de memoria consta de un cierto número de bits (normalmente 8, un *byte*).

La unidad elemental de memoria se llama *byte* (octeto). Un *byte* tiene la capacidad de almacenar un carácter de información, y está formado por un conjunto de unidades más pequeñas de almacenamiento denominadas *bits*, que son dígitos binarios (0 o 1).

Figura 1.5. Computadora portátil digital.

Por definición, se acepta que un byte contiene ocho bits. Por consiguiente, si se desea almacenar la frase:

```
Hola Mortimer todo va bien
```

la computadora utilizará exactamente 27 bytes consecutivos de memoria. Obsérvese que, además de las letras, existen cuatro espacios en blanco y un punto (un espacio es un carácter que emplea también un byte). De modo similar, el número del pasaporte

```
P57487891
```

ocupará 9 bytes, pero si se almacena como

```
P5-748-7891
```

ocupará 11. Estos datos se llaman *alfanuméricos*, y pueden constar de letras del alfabeto, dígitos o incluso caracteres especiales (símbolos: `$, #, *,` etc.).

Mientras que cada carácter de un dato alfanumérico se almacena en un byte, la información numérica se almacena de un modo diferente. Los datos numéricos ocupan 2, 4 e incluso 8 bytes consecutivos, dependiendo del tipo de dato numérico (se verá en el Capítulo 12).

Existen dos conceptos importantes asociados a cada celda o posición de memoria: su *dirección* y su *contenido*. Cada celda o byte tiene asociada una única *dirección* que indica su posición relativa en memoria y mediante la cual se puede acceder a la posición para almacenar o recuperar información. La información almacenada en una posición de memoria es su *contenido*. La Figura 1.6 muestra una memoria de computadora que consta de 1.000 posiciones en memoria con direcciones de 0 a 999. El contenido de estas direcciones o posiciones de memoria se llaman *palabras*, de modo que existen palabras de 8, 16, 32 y 64 bits. Por consiguiente, si trabaja con una máquina de 32 bits, significa que en cada posición de memoria de su computadora puede alojar 32 bits, es decir, 32 dígitos binarios, bien ceros o unos.

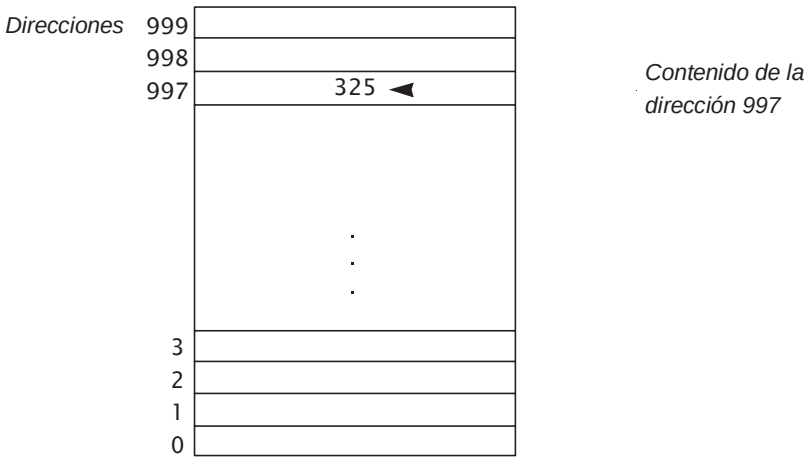


Figura 1.6. Memoria central de una computadora.

Siempre que se almacena una nueva información en una posición, se destruye (desaparece) cualquier información que en ella hubiera y no se puede recuperar. La dirección es permanente y única, el contenido puede cambiar mientras se ejecuta un programa.

La memoria central de una computadora puede tener desde unos centenares de millares de bytes hasta millones de bytes. Como el byte es una unidad elemental de almacenamiento, se utilizan múltiplos de potencia de 2 para definir el tamaño de la memoria central: *Kilo*-byte (**KB** o **Kb**) igual a 1.024 bytes (2^{10}) —prácticamente se consideran 1.000—; *Megabyte* (**MB** o **Mb**) igual a 1.024 × 1.024 bytes = = 1.048.576 (2^{20}) —prácticamente se consideran 1.000.000; *Gigabyte* (**GB** o **Gb**) igual a 1.024 MB (2^{30}), 1.073.741.824 = prácticamente se consideran 1.000 millones de MB.

Tabla 1.1. Unidades de medida de almacenamiento.

Byte	**Byte (B)**	*equivale a*	8 bits
Kilobyte	**Kbyte (KB)**	*equivale a*	1.024 bytes
Megabyte	**Mbyte (MB)**	*equivale a*	1.024 Kbytes
Gigabyte	**Gbyte (GB)**	*equivale a*	1.024 Mbytes
Terabyte	**Tbyte (TB)**	*equivale a*	1.024 Gbytes

1 **Tb** = 1.024 **Gb** = 1.024 × 1.024 **Mb** = 1.048.576 **Kb** = 1.073.741.824 **B**

En la actualidad las computadoras personales tipo PC suelen tener memorias centrales de 512 MB a 2 GB, aunque ya es muy frecuente verlas con memorias de 4 GB y hasta 8 GB.

La memoria principal es la encargada de almacenar los programas y datos que se están ejecutando y su principal característica es que el acceso a los datos o instrucciones desde esta memoria es muy rápido.

En la memoria principal se almacenan:

- Los datos enviados para procesarse desde los dispositivos de entrada.
- Los programas que realizarán los procesos.
- Los resultados obtenidos preparados para enviarse a un dispositivo de salida.

Tipos de memoria principal

En la memoria principal se pueden distinguir dos tipos de memoria: **RAM** y **ROM**. La memoria **RAM** (**R**andom **A**ccess **M**emory, Memoria de acceso aleatorio) almacena los datos e instrucciones a procesar. Es un tipo de memoria volátil (su contenido se pierde cuando se apaga la computadora); esta memoria es, en realidad, la que se suele conocer como memoria principal o de trabajo; en esta memoria se pueden escribir datos y leer de ella. La memoria **ROM** (**R**ead **O**nly **M**emory, Memoria de sólo lectura) es una memoria permanente en la que no se puede escribir (viene pregrabada por el fabricante); es una ***memoria de sólo lectura***. Los programas almacenados en ROM no se pierden al apagar la computadora y cuando se enciende, se lee la información almacenada en esta memoria. Al ser esta memoria de sólo lectura, los programas almacenados en los chips ROM no se pueden modificar y suelen utilizarse para almacenar los programas básicos que sirven para arrancar la computadora.

1.2.3. La Unidad Central de Proceso (UCP): el Procesador

La *Unidad Central de Proceso*, **UCP** (*Central Processing Unit*, **CPU**, en inglés), dirige y controla el proceso de información realizado por la computadora. La **UCP** procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria (esta información son datos o instrucciones: programas). También puede almacenar los resultados de estos procesos en memoria para su uso posterior.

La **UCP** consta de dos componentes: *unidad de control* (**UC**) y *unidad aritmética-lógica* (**UAL**) (Figura 1.7). La **unidad de control** (*Control Unit*, **CU**) coordina las actividades de la computadora y determina qué operaciones se deben realizar y en qué orden; asimismo controla y sincroniza todo el proceso de la computadora. La **unidad aritmético-lógica** (*Arithmetic-Logic Unit*, **ALU**) realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones. Los datos en la memoria central se pueden *leer* (recuperar) o *escribir* (cambiar) por la UCP.

1.2.4. El microprocesador

El **microprocesador** es un *chip* (**un circuito integrado**) que controla y realiza las funciones y operaciones con los datos. Se suele conocer como *procesador* y es el cerebro y corazón de la computadora. En realidad el microprocesador representa a la Unidad Central de Proceso de una computadora.

El primer microprocesador comercial, el Intel 4004 fue presentado el 15 de noviembre de 1971. Existen diferentes fabricantes de microprocesadores, como Intel, Zilog, AMD, Motorola; Cyrix, etc. Microprocesadores históricos de 8 bits son el Intel 8080, Zilog Z80 o Motorola 6800; otros microprocesadores muy populares han sido: 8086, 8088 de Intel o el Motorola MC68000. En la década de los ochenta eran populares: Intel 80286, 80386, 80486; Motorola, MC 68020, MC68400; AMD 80386, 80486.

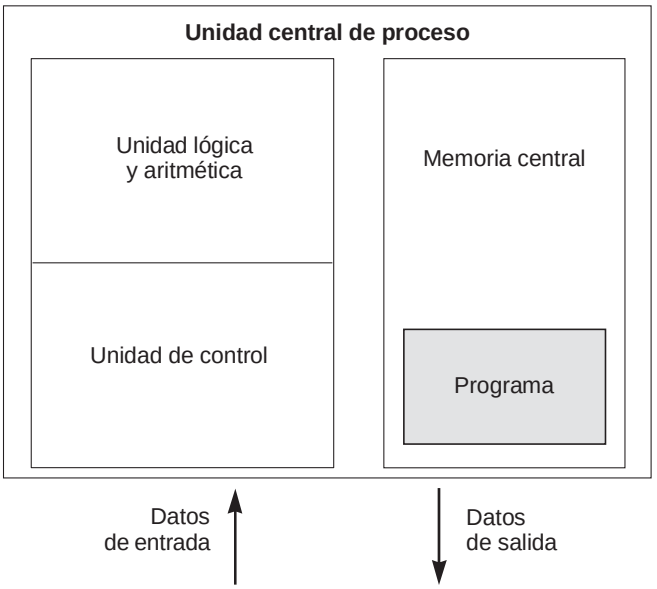


Figura 1.7. Unidad central de proceso.

En el año 1993 aparecieron el Intel Pentium y durante esa década, Intel Pentium Pro, Intel Pentium II/III y AMD K6. En 2000, Intel y **AMD** controlan el mercado con Intel Pentium IV, Intel Titanium, Intel Pentium D o bien AMD Athlon XP, AMD Duxor. En los años 2005 y 2006 aparecen las nuevas tecnologías **Intel Core Duo**, **AMD Athlon 64**, etc.

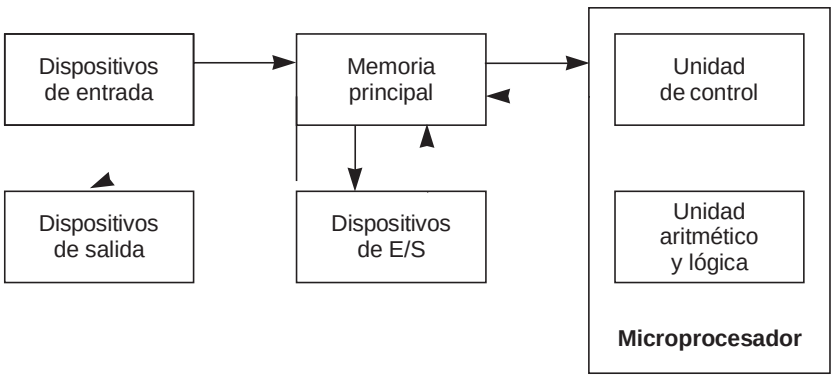


Figura 1.8. Organización física de una computadora con un microprocesador.

1.2.5. Memoria externa: almacenamiento masivo

Cuando un programa se ejecuta, se debe situar primero en memoria central de igual modo que los datos. Sin embargo, la información almacenada en la memoria se pierde (borra) cuando se *apaga* (desconecta de la red eléctrica) la computadora, y por otra parte la memoria central es limitada en capacidad. Por esta razón, para poder disponer de almacenamiento permanente, tanto para programas como para datos, se necesitan *dispositivos de almacenamiento secundario, auxiliar* o *masivo* ("*mass storage*" o "*secondary storage*").

Los **dispositivos de almacenamiento** o **memorias auxiliares** (*externas* o *secundarias*) más comúnmente utilizados son: *cintas magnéticas, discos magnéticos, discos compactos* (**CD-ROM, C**ompact **D**isk **R**ead **O**nly **M**emory) y videodiscos digitales (**DVD**). Las cintas son utilizadas principalmente por

sistemas de computadores grandes similares a las utilizadas en los equipos de audio. Los discos y disquetes magnéticos se utilizan por todas las computadoras, especialmente las medias y pequeñas —las computadoras personales. Los discos pueden ser *duros*, de gran capacidad de almacenamiento (su capacidad actual oscila entre 40 GB y 500 GB), **disquetes** o *discos flexibles* ("floppy disk"), ya casi en desuso. Aunque todavía se suelen comercializar lectoras de disquetes para compatibilidad con equipos antiguos. El disquete, ya casi en desuso, es de 3,5'' y de 1,44 MB de capacidad.

Figura 1.9. Memorias auxiliares: Tarjeta compact flash (izquierda), memoria flash USB (centro) y disco duro (derecha).

Los discos compactos (conocidos popularmente como **CD**) son soportes digitales ópticos utilizados para almacenar cualquier tipo de información (audio, vídeo, documentos.). Se desarrolló en 1980 y comenzó a comercializarse en 1982. Existen diferentes modelos CD-ROM (de sólo lectura), **CD-R** (grabable), **CD-RW** (reescribible). Su capacidad de almacenamiento va de 650 MB a 875 MB e incluso 215 MB.

Los **DVD** constituyen un formato multimedia de almacenamiento óptico y que se puede usar para guardar datos, incluyendo películas de alta calidad de vídeo y audio. Los formatos más populares son: DVD–ROM, DVD±R, DVD±RW, DVD±RAM, y sus capacidades de almacenamiento van desde 4,7 GB y 8,5 GB hasta 17,1 GB, según sean de una cara, de dos caras y de una capa simple o capa doble.

Los últimos discos ópticos presentados en el mercado durante 2006 son: ***Blu-ray*** y **HD DVD**. Estos discos son de alta definición y su capacidad de almacenamiento es muy grande de 15 GB a 50 GB y podrá llegar en el futuro hasta 200 GB.

La información almacenada en la memoria central es *volátil* (desaparece cuando se apaga la computadora) mientras que la información almacenada en la memoria externa (masiva) es *permanente*.

Esta información se organiza en unidades independientes llamadas **archivos** (**ficheros**, *file* en inglés). Los resultados de los programas se pueden guardar como *archivos de datos* y los programas que se escriben se guardan como *archivos de programas*, ambos en la memoria auxiliar. Cualquier tipo de archivo se puede transferir fácilmente desde la memoria auxiliar hasta la memoria central para su proceso posterior.

En el campo de las computadoras es frecuente utilizar la palabra memoria y almacenamiento o memoria externa, indistintamente. En este libro —y recomendamos su uso— se utilizará el término memoria sólo para referirse a la memoria central.

Comparación de la memoria central y la memoria externa

La memoria central o principal es mucho más rápida y cara que la memoria externa. Se deben transferir los datos desde la memoria externa hasta la memoria central, antes de que puedan ser procesados. Los datos en memoria central son: *volátiles* y desaparecen cuando se *apaga* la computadora. Los datos en memoria externa son *permanentes* y no desaparecen cuando se *apaga* la computadora.

Las computadoras modernas necesitan comunicarse con otras computadoras. Si la computadora se conecta con una *tarjeta de red* se puede conectar a una red de datos locales (*red de área local*). De este

modo se puede acceder y compartir a cada una de las memorias de disco y otros dispositivos de entrada y salida. Si la computadora tiene un *módem*, se puede comunicar con computadoras lejanas. Se pueden conectar a una red de datos o *enviar correo electrónioco* a través de las redes corporativas Intranet/Extranet o la propia red Internet. Las redes inalámbricas permiten conexiones a Internet desde números os lugares, siempre que su PC disponga de tarjetas o conexiones inalámbricas.

1.2.6. La computadora personal ideal para programación

Hoy día el estudiante de informática o de computación y mucho más el profesional, dispone de un amplio abanico de computadoras a precios asequibles y con prestaciones altas. En el segundo semestre de 2006 se pueden encontrar computadoras personales (PC) con 1.024 MB de memoria, grabadoras de DVD de doblecapa, procesador Pentium de 3.00/3.20 GH2 y un monitor plano 17", disco duro de 200/400 GB por 800 a 1.000 € e incluso más económicos. En computadoras pórtatíles se pueden encontrar modelos de 1024 MB, 60-80 GB, procesadores Intel Pentium de 1,736 GHz, 1,83 GHz, pantalla de 15,4" y con precios de 800 a 1200 €. La Tabla 1.2 resume nuestra propuesta y recomendación de características medias para un/una PC, a mediados de 2006.

Tabla 1.2. Características medias de una computadora portatil (*laptop*).

Procesador	Intel Centrino 1.6 a 1,73/2.0 GHz; Intel CorneDuo (1.73/1.83 GHz; AMD Turion 64).
Memoria	512 MB a 1.0 GB/2 GB.
Disco duro	60-120 GB.
Internet	Tarjeta de red; modem 56 Kbps; red inalámbrica 802,11 b/g; *Bluetooth*.
Vídeo	Memoria de vídeo de 128 a 512 MB.
Pantalla	15", 15,4" o 17" (se comercializan también de 11", 12" y 13").
Almacenamiento	Grabadora DVD +/– RW de doble capa.
Puertos	3/4 puertos USB 2.0, 1 IEEE, lector de tarjetas.
Marcas	HP, Compaq, Dell, IBM, EI System, Gateways, Acer, Toshiba, Sony, Cofiman...

1.3. REPRESENTACIÓN DE LA INFORMACIÓN EN LAS COMPUTADORAS

Una computadora es un sistema para procesar información de modo automático. Un tema vital en el proceso de funcionamiento de una computadora es estudiar la forma de representación de la información en dicha computadora. Es necesario considerar cómo se puede codificar la información en patrones de bits que sean fácilmente almacenables y procesables por los elementos internos de la computadora.

Las formas de información más significativas son: textos, sonidos, imágenes y valores numéricos y, cada una de ellas presentan peculiaridades distintas. Otros temas importantes en el campo de la programación se refieren a los métodos de detección de errores que se puedan producir en la transmisión o almacenamiento de la información y a las técnicas y mecanismos de comprensión de información al objeto de que ésta ocupe el menor espacio en los dispositivos de almacenamiento y sea más rápida su transmisión.

1.3.1. Representación de textos

La información en formato de texto se representa mediante un código en el que cada uno de los distintos símbolos del texto (tales como letras del alfabeto o signos de puntuación) se asignan a un único patrón de bits. El texto se representa como una cadena larga de bits en la cual los sucesivos patrones representan los sucesivos símbolos del texto original.

En resumen, se puede representar cualquier información escrita (texto) mediante caracteres. Los caracteres que se utilizan en computación suelen agruparse en cinco categorías:

1. **Caracteres alfabéticos** (letras mayúsculas y minúsculas, en una primera versión del abecedario inglés).

   A, B, C, D, E, ... X, Y, Z, a, b, c, ... , X, Y, Z

2. **Caracteres numéricos** (dígitos del sistema de numeración).

   0, 1, 2, 3, 4, 5, 6, 7, 8, 9 *sistema decimal*

3. **Caracteres especiales** (símbolos ortográficos y matemáticos no incluidos en los grupos anteriores).

   { } Ñ ñ ! ? & > # ç ...

4. **Caracteres geométricos y gráficos** (símbolos o módulos con los cuales se pueden representar cuadros, figuras geométricas, iconos, etc.

   | ¯| |¯ ♣ ~ ...

5. **Caracteres de control** (representan órdenes de control como el carácter para pasar a la siguiente línea [NL] o para ir al comienzo de una línea [**RC**, *retorno de carro*, «*carriage* return, **CR**»] emitir un pitido en el terminal [BEL], etc.).

Al introducir un texto en una computadora, a través de un periférico, los caracteres se codifican según un **código de entrada/salida** de modo que a cada carácter se le asocia una determinada combinación de *n* bits.

Los códigos más utilizados en la actualidad son: **EBCDIC**, **ASCII** y **Unicode**.

- **Código EBCDIC** (*Extended Binary Coded Decimal Inter Change Code*).
  Este código utiliza $n = 8$ bits de forma que se puede codificar hasta $m = 2^8 = 256$ símbolos diferentes. Éste fue el primer código utilizado para computadoras, aceptado en principio por IBM.
- **Código ASCII** (*American Standard Code for Information Interchange*).
  El código ASCII básico utiliza 7 bits y permite representar 128 caracteres (letras mayúsculas y minúsculas del alfabeto inglés, símbolos de puntuación, dígitos 0 a 9 y ciertos controles de información tales como retorno de carro, salto de línea, tabulaciones, etc.). Este código es el más utilizado en computadoras, aunque el ASCII ampliado con 8 bits permite llegar a 2^8 (256) caracteres distintos, entre ellos ya símbolos y caracteres especiales de otros idiomas como el español.
- **Código Unicode**
  Aunque ASCII ha sido y es dominante en los caracteres se leen como referencia, hoy día se requiere de la necesidad de representación de la información en muchas otras lenguas, como el portugués, español, chino, el japonés, el árabe, etc. Este código utiliza un patrón único de 16 bits para representar cada símbolo, que permite 2^{16} bits o sea hasta 65.536 patrones de bits (símbolos) diferentes.

Desde el punto de vista de unidad de almacenamiento de caracteres, se utiliza el archivo (**fichero**). Un **archivo** consta de una secuencia de símbolos de una determinada longitud codificados utilizando ASCII o Unicode y que se denomina **archivo de texto**. Es importante diferenciar entre archivos de texto simples que son manipulados por los programas de utilidad denominados **editores de texto** y los archivos de texto más elaborados que se producen por los procesadores de texto, tipo Microsoft Word. Ambos constan de caracteres de texto, pero mientras el obtenido con el editor de texto, es un archivo de texto puro que codifica carácter a carácter, el archivo de texto producido por un procesador de textos contiene números, códigos que representan cambios de formato, de tipos de fuentes de letra y otros, e incluso pueden utilizar códigos propietarios distintos de ASCII o Unicode.

1.3.2. Representación de valores númericos

El almacenamiento de información como caracteres codificados es ineficiente cuando la información se registra como numérica pura. Veamos esta situación con la codificación del número 65; si se almacena como caracteres ASCII utilizando un byte por símbolo, se necesita un total de 16 bits, de modo que el número mayor que se podía almacenar en 16 bits (dos bytes) sería 99. Sin embargo, si utilizamos *notación binaria* para almacenar enteros, el rango puede ir de 0 a 65.535 ($2^{16} - 1$) para números de 16 bits. Por consiguiente, la notación binaria (o variantes de ellas) es la más utilizada para el almacenamiento de datos numéricos codificados.

La solución que se adopta para la representación de datos numéricos es la siguiente: al introducir un número en la computadora se codifica y se almacena como un texto o cadena de caracteres, pero dentro del programa a cada dato se le envía un tipo de dato específico y es tarea del programador asociar cada dato al tipo adecuado correspondiente a las tareas y operaciones que se vayan a realizar con dicho dato.

El método práctico realizado por la computadora es que una vez definidos los datos numéricos de un programa, una rutina (función interna) de la biblioteca del compilador (traductor) del lenguaje de programación se encarga de transformar la cadena de caracteres que representa el número en su notación binaria.

Existen dos formas de representar los datos numéricos: números enteros o números reales.

Representación de enteros

Los datos de tipo entero se representan en el interior de la computadora en notación binaria. La memoria ocupada por los tipos enteros depende del sistema, pero normalmente son dos, bytes (en las versiones de MS-DOS y versiones antiguas de Windows y cuatro bytes en los sistemas de 32 bits como Windows o Linux). Por ejemplo, un entero almacenado en 2 bytes (16 bits):

```
1000 1110 0101 1011
```

Los enteros se pueden representar con signo (`signed`, en C++) o sin signo (`unsigned`, en C++); es decir, números positivos o negativos. Normalmente, se utiliza un bit para el signo. Los enteros sin signo al no tener signo pueden contener valores positivos más grandes. Normalmente, si un entero no se especifica «con/sin signo» se suele asignar con signo por defecto u omisión.

El rango de posibles valores de enteros depende del tamaño en bytes ocupado por los números y si se representan con signo o sin signo (la Tabla 1.3 resume características de tipos estándar en C++).

Representación de reales

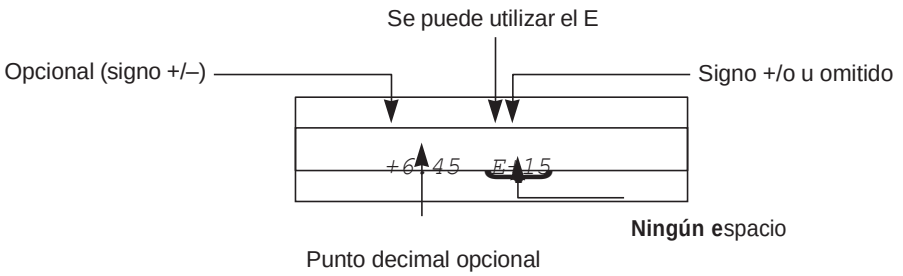
Los números reales son aquellos que contienen una parte decimal como `2,6` y `3,14152`. Los reales se representan en *notación científica* o en *coma flotante*; por esta razón en los lenguajes de programación, como C++, se conocen como números en coma flotante.

Existen dos formas de representar los números reales. La primera se utiliza con la notación del punto decimal (*ojo* en el formato de representación español de números decimales, la parte decimal se representa por coma).

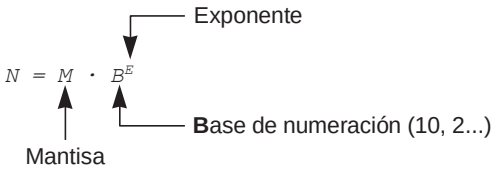
Ejemplos

```
12.35  99901.32  0.00025  9.0
```

La segunda forma para representar números en coma flotante en la notación científica o exponencial, conocida también como notación `E`. Esta notación es muy útil para representar números muy grandes o muy pequeños.



Notación exponencial



Ejemplos

```
2.52 e + 8
8.34 E - 4      equivale a      8.34/10⁴ = 0.000834
7E5
-18.35e15
5.95E25
9.11e - 31      equivale a      0.000000000000000000000000000000911
```

Representación de caracteres

Un documento de texto se escribe utilizando un conjunto de caracteres adecuado al tipo de documento. En los lenguajes de programación se utilizan, principalmente, dos códigos de caracteres. El más común es **ASCII** (American Standard Code for Information Interchange) y algunos lenguajes, tal como Java, utilizan **Unicode** (*www.unicode.org*). Ambos códigos se basan en la asignación de un código numérico a cada uno de los tipos de caracteres del código.

En C++, los *caracteres* se procesan normalmente usando el tipo `char`, que asocia cada carácter a un código numérico que se almacena en un byte.

El código ASCII básico que utiliza 7 bits (128 caracteres distintos) y el ASCII *ampliado* a 8 bits (256 caracteres distintos) son los códigos más utilizados. Así se pueden representar caracteres tales como `'A'`, `'B'`, `'c'`, `'$'`, `'4'`, `'5'`, etc.

En el caso de C++ se considera un dato de tipo carácter denominado `wchar_t` que sirve para representar caracteres de tipo internacional para manejar lenguajes extranjeros.

La Tabla 1.3, recoge los tipos enteros, reales y carácter utilizados en C++, la memoria utilizada (número de bytes ocupados por el dato) y el rango de números.

1.3.3. Representación de imágenes

Las imágenes se adquieren mediante periféricos especializados tales como escáneres, cámaras digitales de vídeo, cámaras fotográficas, etc. Una imagen, al igual que otros tipos de información, se representan por patrones de bits, generados por el periférico correspondiente. Existen dos métodos básicos para representar imágenes: *mapas de bits* y *mapas de vectores*.

Tabla 1.3. Tipos enteros reales, en C++.

		Carácter y bool
Tipo	**Tamaño**	**Rango**
`short (short int)`	2 bytes	`-32.738..32.767`
`int`	4 bytes	`-2.147.483.648` a `2.147.483.647`
`long (long int)`	4 bytes	`-2.147.483.648` a `2.147.483.647`
`float (real)`	4 bytes	10^{-38} a 10^{38} (*aproximadamente*)
`double`	8 bytes	10^{-308} a 10^{308} (*aproximadamente*)
`long double`	10 bytes	10^{-4932} a 10^{4932} (*aproximadamente*)
`char (carácter)`	1 byte	Todos los caracteres ASCII
`bool`	1 byte	`True` (verdadero) y `false` (falso)

En las técnicas de *mapas de bits*, una imagen se considera como una colección de puntos, cada uno de los cuales se llama *pixel* (abreviatura de «*picture element*»). Una imagen en blanco y negro se representa como una cadena larga de bits que representan las filas de píxeles en la imagen, donde cada bit es bien `1` o bien `0`, dependiendo de que el pixel correspondiente sea blanco o negro. En el caso de imágenes en color, cada pixel se representa por una combinación de bits que indican el color de los pixel. Cuando se utilizan técnicas de mapas de bits, el patrón de bits resultante se llama *mapa de bits*, significando que el patrón de bits resultante que representa la imagen es poco más que un mapa de la imagen. Muchos de los periféricos de computadora —tales como cámaras de vídeo, escáneres, etc.— convierten imágenes de color en formato de mapa de bits. Los formatos más utilizados en la representación de imágenes se muestran en la Tabla 1.4.

Tabla 1.4. Mapas de bits.

Formato	**Origen y descripción**
`BMP`	**Microsoft**. Formato sencillo con imágenes de gran calidad pero con el inconveniente de ocupar mucho (no útil para la web).
`JPEG`	Grupo **JPEG**. Calidad aceptable para imágenes naturales. Incluye compresión. Se utiliza en la web.
`GIF`	**CompuServe**. Muy adecuado para imágenes no naturales (logotipos, banderas, dibujos anidados...). Muy usuado en la web.

Mapas de vectores. Otros métodos de representar una imagen se fundamentan en descomponer la imagen en una colección de objetos tales como líneas, polígonos y textos con sus respectivos atributos o detalles (grosor, color, etc.).

Tabla 1.5. Mapas de vectores.

Formato	**Descripción**
`IGES`	ASME/ANSI. Estándar para intercambio de datos y modelos de (AutoCAD,...).
`Pict`	Apple Computer. Imágenes vectoriales.
`EPS`	Adobe Computer.
`TrueType`	Apple y Microsoft para EPS.

1.3.4. Rrepresentación de sonidos

La representación de sonidos ha adquirido una importancia notable debido esencialmente a la infinidad de aplicaciones multimedia tanto autónomas como en la *web*.

El método más genérico de codificación de la información de audio para almacenamiento y manipulación en computadora es mostrar la amplitud de la onda de sonido en intervalos regulares y registrar las series de valores obtenidos. La señal de sonido se capta mediante micrófonos o dispositivos similares y produce una señal analógica que puede tomar cualquier valor dentro de un intervalo continuo determinado. En un intervalo de tiempo continuo se dispone de infinitos valores de la señal analógica, que es necesario almacenar y procesar, para lo cual se recurre a una *técnica de muestreo*. Las muestras obtenidas se digitalizan con un conversor analógico-digital, de modo que la señal de sonido se representa por secuencias de bits (por ejemplo, 8 o 16) para cada muestra. Esta técnica es similar a la utilizada, históricamente, por las comunicaciones telefónicas a larga distancia. Naturalmente, dependiendo de la calidad de sonido que se requiera, se necesitarán más números de bits por muestra, frecuencias de muestreo más altas y lógicamente más muestreos por períodos de tiempo[4].

Como datos de referencia puede considerar que para obtener reproducción de calidad de sonido de alta fidelidad para un disco CD de música, se suele utilizar, al menos, una frecuencia de muestreo de 44.000 muestras por segundo. Los datos obtenidos en cada muestra se codifican en 16 bits (32 bits para grabaciones en estéreo). Como dato anecdótico, cada segundo de música grabada en estéreo requiere más de un millón de bits.

Un sistema de codificación de música muy extendido en sintetizadores musicales es MIDI (***M**usical **I**nstruments **D**igital **I**nterface*) que se encuentra en sintetizadores de música para sonidos de videojuegos, sitios web, teclados electrónicos, etc.

1.4. CONCEPTO DE ALGORITMO

El objetivo fundamental de este texto es enseñar a resolver problemas mediante una computadora. El programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. A lo largo de todo el libro nos referiremos a la *metodología necesaria para resolver problemas mediante programas*, concepto que se denomina **metodología de la programación**. El eje central de esta metodología es el concepto, ya tratado, de algoritmo.

Un algoritmo es un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, **algoritmo** proviene de *Mohammed al-Khowârizmi*, matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra *algorismus* derivó posteriormente en algoritmo. Euclides, el gran matemático griego (del siglo IV antes de Cristo) que inventó un método para encontrar el máximo común divisor de dos números, se considera con Al-Khowârizmi el otro gran padre de la algoritmia (ciencia que trata de los algoritmos).

El profesor Niklaus Wirth —inventor de Pascal, Modula-2 y Oberon— tituló uno de sus más famosos libros, *Algoritmos + Estructuras de datos = Programas*, significándonos que sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos. Esta ecuación será una de las hipótesis fundamentales consideradas en esta obra.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.

---

[4] En las obras del profesor Alberto Prieto, Schaum «*Conceptos de Informática* e *Introducción a la Informática*», publicadas en McGraw-Hill, puede encontrar una excelente referencia sobre estos conceptos y otros complementarios de este capítulo introductorio.

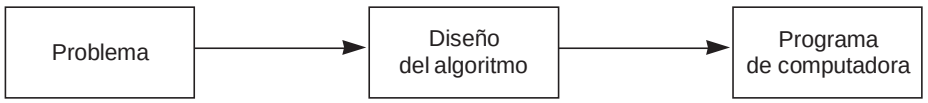


Figura 1.10. Resolución de un problema.

Los pasos para la resolución de un problema son:

1. *Diseño del algoritmo*, que describe la secuencia ordenada de pasos —sin ambigüedades— que conducen a la solución de un problema dado. (*Análisis del problema y desarrollo del algoritmo*.)
2. Expresar el algoritmo como un *programa* en un lenguaje de programación adecuado. (*Fase de codificación*.)
3. *Ejecución y validación* del programa por la computadora.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el idioma del cocinero.

En la ciencia de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Dada la importancia del algoritmo en la ciencia de la computación, un aspecto muy importante será *el diseño de algoritmos*. A la enseñanza y práctica de esta tarea denominada *algoritmia* se dedica gran parte de este libro.

El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia, *la solución de un problema se puede expresar mediante un algoritmo*.

1.4.1. Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser *preciso* e indicar el orden de realización de cada paso.
- Un algoritmo debe estar *definido*. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser *finito*. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: *Entrada, Proceso* y *Salida*. En el algoritmo de receta de cocina citado anteriormente se tendrá:

Entrada: ingredientes y utensilios empleados.
Proceso: elaboración de la receta en la cocina.
Salida: terminación del plato (por ejemplo, cordero).

Ejemplo 1.1

Un cliente ejecuta un pedido a una fábrica. La fábrica examina en su banco de datos la ficha del cliente; si el cliente es solvente entonces la empresa acepta el pedido; en caso contrario, rechazará el pedido. Redactar el algoritmo correspondiente.

Los pasos del algoritmo son:

```
1. Inicio.
2. Leer el pedido.
3. Examinar la ficha del cliente.
4. Si el cliente es solvente, aceptar pedido; en caso contrario, rechazar
   pedido.
5. Fin.
```

Ejemplo 1.2

Se desea diseñar un algoritmo para saber si un número es primo o no.

Un número es primo si sólo puede dividirse por sí mismo y por la unidad (es decir, no tiene más divisores que él mismo y la unidad). Por ejemplo, 9, 8, 6, 4, 12, 16, 20, etc., no son primos, ya que son divisibles por números distintos a ellos mismos y a la unidad. Así, 9 es divisible por 3, 8 lo es por 2, etc.

El algoritmo de resolución del problema pasa por dividir sucesivamente el número por 2, 3, 4, etc.

```
1. Inicio.
2. Poner X igual a 2 (x = 2, x variable que representa a los divisores
   del número que se busca N).
3. Dividir N por X (N/X).
4. Si el resultado de N/X es entero, entonces N es un número primo y bi-
   furcar al punto 7; en caso contrario, continuar el proceso.
5. Suma 1 a X (X ← X + 1).
6. Si X es igual a N, entonces N es un número primo; en caso contrario,
   bifurcar al punto 3.
7. Fin.
```

Por ejemplo, si `N` es `131`, los pasos anteriores serían:

```
1. Inicio.
2. X = 2.
3 y 4. 131/X. Como el resultado no es entero, se continúa el proceso.
5. X ← 2 + 1, luego X = 3.
6. Como X no es 131, se bifurca al punto 3.
3 y 4. 131/X resultado no es entero.
5. X ← 3 + 1, X = 4.
6. Como X no es 131 bifurca al punto 3.
3 y 4.131/X..., etc.
7. Fin.
```

Ejemplo 1.3

Realizar la suma de todos los números pares entre 2 y 1.000.

El problema consiste en sumar 2 + 4 + 6 + 8 ... + 1.000. Utilizaremos las palabras `SUMA` y `NUMERO` (*variables*, serán denominadas más tarde) para representar las sumas sucesivas (2+4), (2+4+6), (2+4+6+8), etcétera. La solución se puede escribir con el siguiente algoritmo:

```
1. Inicio.
2. establecer SUMA a 0.
3. establecer NUMERO a 2.
4. Sumar NUMERO a SUMA. El resultado será el nuevo valor de la suma
   (SUMA).
5. Incrementar NUMERO en 2 unidades.
6. Si NUMERO =< 1.000 bifurcar al paso 4;
7. en caso contrario, escribir el último valor de SUMA y terminar el pro-
   ceso.
8. Fin.
```

1.5. PROGRAMACIÓN ESTRUCTURADA

La programación orientada a objetos se desarrolló para tratar de paliar diversas limitaciones que se encontraban en anteriores enfoques de programación. Para apreciar las ventajas de la POO, es preciso constatar las limitaciones citadas y cómo se producen con los lenguajes de programación tradicionales.

C, Pascal y FORTRAN, y lenguajes similares, se conocen como *lenguajes procedimentales* (por procedimientos). Es decir, cada sentencia o instrucción señala al compilador para que realice alguna tarea: obtener una entrada, producir una salida, sumar tres números, dividir por cinco, etc. En resumen, un programa en un lenguaje procedimental es un conjunto de instrucciones o sentencias. En el caso de pequeños programas, estos principios de organización (denominados *paradigma)* se demuestran eficientes. El programador sólo tiene que crear esta lista de instrucciones en un lenguaje de programación, compilar en la computadora y ésta, a su vez, ejecuta estas instrucciones.

Cuando los programas se vuelven más grandes, cosa que lógicamente sucede cuando aumenta la complejidad del problema a resolver, la lista de instrucciones aumenta considerablemente, de modo tal que el programador tiene muchas dificultades para controlar ese gran número de instrucciones. Los programadores pueden controlar, de modo normal, unos centenares de líneas de instrucciones. Para resolver este problema los programas se descompusieron en unidades más pequeñas que adoptaron el nombre de *funciones* (*procedimientos*, *subprogramas* o *subrutinas* en otros lenguajes de programación). De este modo en un programa orientado a procedimientos se divide en funciones, de modo que cada función tiene un propósito bien definido y resuelve una tarea concreta, y se diseña una interfaz claramente definida (el prototipo o cabecera de la función) para su comunicación con otras funciones.

Con el paso de los años, la idea de romper en programa en funciones fue evolucionando y se llegó al agrupamiento de las funciones en otras unidades más grandes llamadas *módulos* (normalmente, en el caso de C, denominadas **archivos** o **ficheros**); sin embargo, el principio seguía siendo el mismo: agrupar componentes que ejecutan listas de instrucciones (sentencias). Esta característica hace que a medida que los programas se hacen más grandes y complejos, el paradigma estructurado comienza a dar señales de debilidad y resultando muy difícil terminar los programas de un modo eficiente. Existen varias razones de la debilidad de los programas estructurados para resolver problemas complejos. Tal vez las dos razones más evidentes son éstas. Primero, las funciones tienen acceso ilimitado a los datos globales. Segundo, las funciones inconexas y datos, fundamentos del paradigma procedimental proporcionan un modelo pobre del mundo real.

1.5.1. Datos locales y datos globales

En un programa procedimental, por ejemplo escrito en C, existen dos tipos de datos. *Datos locales* que son ocultos en el interior de la función y son utilizados, exclusivamente, por la función. Estos datos locales están estrechamente relacionados con sus funciones y están protegidos de modificaciones por otras funciones.

Otro tipo de datos son los *datos globales* a los cuales se puede acceder desde *cualquier* función del programa. Es decir, dos o más funciones pueden acceder a los mismos datos siempre que estos datos sean globales. En la Figura 1.11 se muestra la disposición de variables locales y globales en un programa procedimental.

Un programa grande (Figura 1.12) se compone de numerosas funciones y datos globales y ello conlleva una multitud de conexiones entre funciones y datos que dificulta su comprensión y lectura.

Todas estas conexiones múltiples originan diferentes problemas. En primer lugar, hacen difícil conceptuar la estructura del programa. En segundo lugar, el programa es difícil de modificar ya que cambios en datos globales pueden necesitar la reescritura de todas las funciones que acceden a los mismos. También puede suceder que estas modificaciones de los datos globales pueden no ser aceptadas por todas o algunas de las funciones.

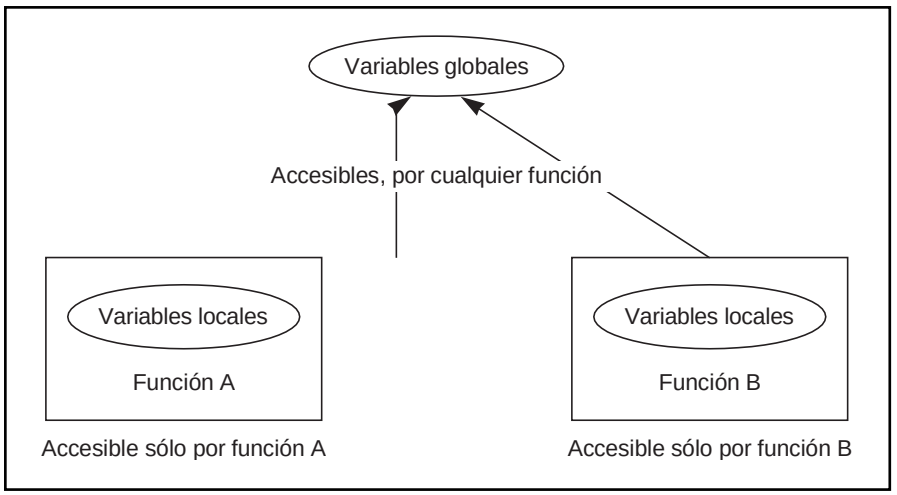


Figura 1.11. Datos locales y globales.

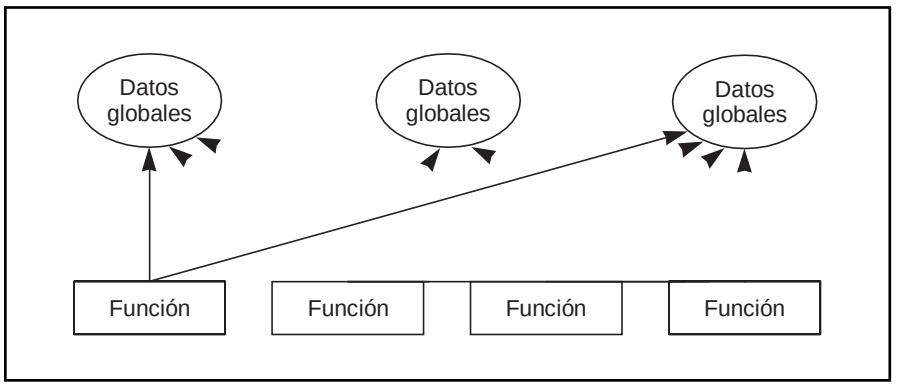


Figura 1.12. Un programa procedimental.

1.5.2 Modelado del mundo real

Un segundo problema importante de la programación estructurada reside en el hecho de que la disposición separada de datos y funciones no se corresponden con los modelos de las cosas del mundo real. En el mundo físico se trata con objetos físicos tales como personas, autos o aviones. Estos objetos no son como los datos ni como las funciones. Los objetos complejos o no del mundo real tienen *atributos* y *comportamiento*.

Los **atributos** o características de los objetos son, por ejemplo: en las personas, su edad, su profesión, su domicilio, etc.; en un auto, la potencia, el número de matrícula, el precio, número de puertas, etc; en una casa, la superficie, el precio, el año de construcción, la dirección, etc. En realidad, los atributos del mundo real tienen su equivalente en los datos de un programa; tienen un valor específico, tal como 200 metros cuadrados, 20.000 dólares, cinco puertas, etc.

El **comportamiento** es una acción que ejecutan los objetos del mundo real como respuesta a un determinado estímulo. Si usted pisa los frenos en un auto, el coche (carro) se detiene; si acelera, el auto aumenta su velocidad, etc. El comportamiento, en esencia, es como una función: se llama a una función para hacer algo (visualizar la nómina de los empleados de una empresa).

Por estas razones, ni los datos ni las funciones, por sí mismas, modelan los objetos del mundo real de un modo eficiente.

> La programación estructurada mejora la claridad, fiabilidad y facilidad de mantenimiento de los programas; sin embargo, para programas grandes o a gran escala, presentan retos de difícil solución.

1.6. PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos, tal vez el paradigma de programación más utilizado en el mundo del desarrollo de software y de la ingeniería de software del siglo XXI, trae un nuevo enfoque a los retos que se plantean en la programación estructurada cuando los problemas a resolver son complejos. Al contrario que la programación *procedimental* que enfatiza en los algoritmos, la POO enfatiza en los datos. En lugar de intentar ajustar un problema al enfoque *procedimental* de un lenguaje, POO intenta ajustar el lenguaje al problema. La idea es diseñar formatos de datos que se correspondan con las características esenciales de un problema.

La idea fundamental de los lenguajes orientados a objetos es combinar en una única unidad o módulo, tanto los datos como las funciones que operan sobre esos datos. Tal unidad se llama un **objeto**.

Las funciones de un objeto se llaman *funciones miembro* en C++ o *métodos* (éste es el caso de Smalltalk, uno de los primeros lenguajes orientados a objetos), y son el único medio para acceder a sus datos. Los datos de un objeto, se conocen también como *atributos* o *variables de instancia*. Si se desea leer datos de un objeto, se llama a una función miembro del objeto. Se accede a los datos y se devuelve un valor. No se puede acceder a los datos directamente. Los datos son ocultos, de modo que están protegidos de alteraciones accidentales. Los datos y las funciones se dice que están *encapsulados en una única entidad*. *El encapsulamiento de datos y la ocultación* de los datos son términos clave en la descripción de lenguajes orientados a objetos.

Si se desea modificar los datos de un objeto, se conoce exactamente cuáles son las funciones que interactúan con las funciones miembro del objeto. Ninguna otra función puede acceder a los datos. Esto simplifica la escritura, depuración y mantenimiento del programa. Un programa C++ se compone normalmente de un número de objetos que se comunican unos con otros mediante la llamada a otras funciones miembro. La organización de un programa en C++ se muestra en la Figura 1.13. La llamada a una función miembro de un objeto se denomina *enviar un mensaje* a otro objeto.

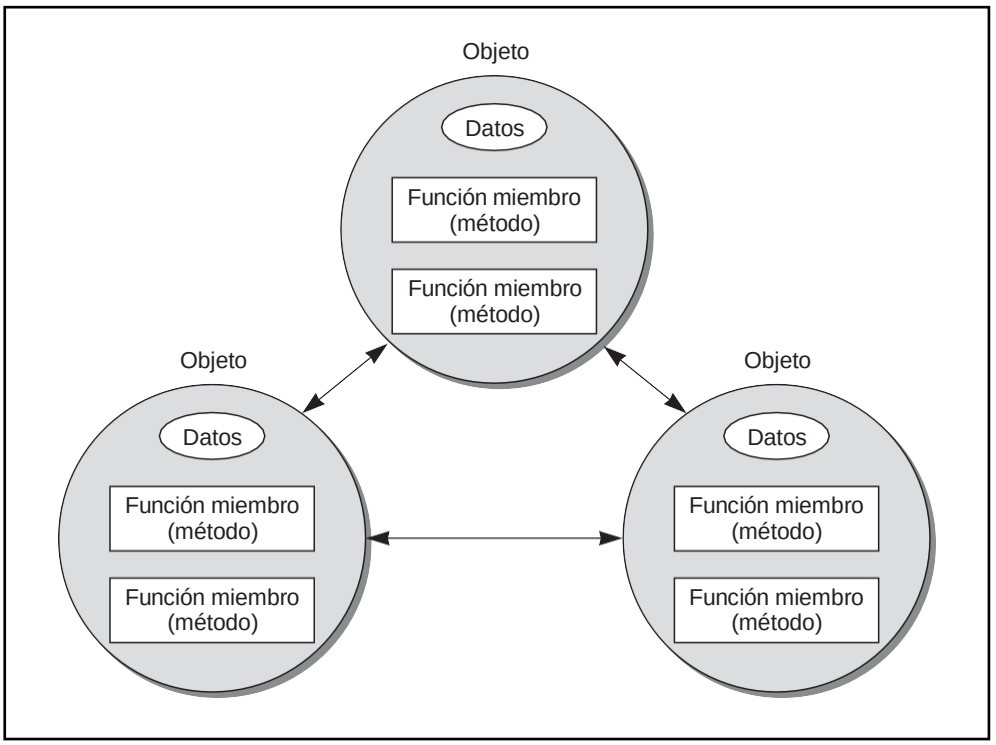


Figura 1.13. Organización típica de un programa orientado a objetos

En el paradigma orientado a objetos, el programa se organiza como un conjunto finito de objetos que contiene datos y operaciones (funciones miembro en C++) que llaman a esos datos y que se comunican entre sí mediante mensajes.

1.6.1. Propiedades fundamentales de la orientación a objetos

Existen diversas características ligadas a la orientación a objetos. Todas las propiedades que se suelen considerar, no son exclusivas de este paradigma, ya que pueden existir en otros paradigmas, pero en su conjunto definen claramente los lenguajes orientados a objetos. Estas propiedades son:

- **Abstracción (tipos abstractos de datos y clases)**.
- **Encapsulado de datos**.
- **Ocultación de datos**.
- **Herencia**.
- **Polimorfismo**.

C++ soporta todas las características anteriores que definen la orientación a objetos, aunque hay numerosas discusiones en torno a la consideración de C++ como lenguaje orientado a objetos. La razón es que en contraste con lenguajes tales como Smalltalk, Java o C#, C++ no es un lenguaje orientado a objetos puro. C++ soporta orientación a objetos pero es compatible con C y permite que programas C++ se escriban sin utilizar características orientadas a objetos. De hecho, C++ es un lenguaje *multiparadigma* que permite programación estructurada, *procedimental*, orientada a objetos y genérica.

1.6.2. Abstracción

La abstracción es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos. El término **abstracción** que se suele utilizar en programación se refiere al hecho de diferenciar entre las propiedades externas de una entidad y los detalles de la composición interna de dicha entidad. Es la abstracción la que permite ignorar los detalles internos de un dispositivo complejo tal como una computadora, un automóvil, una lavadora o un horno de microondas, etc., y usarlo como una única unidad comprensible. Mediante la abstracción se diseñan y fabrican estos sistemas complejos en primer lugar y, posteriormente, los componentes más pequeños de los cuales están compuestos. Cada componente representa un nivel de abstracción en el cual el uso del componente se aísla de los detalles de la composición interna del componente. La abstracción posee diversos grados denominados niveles de abstracción.

En consecuencia, la abstracción posee diversos grados de complejidad que se denominan *niveles de abstracción* que ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. En el modelado orientado a objetos de un sistema esto significa centrarse en *qué* es *y qué hace* un objeto y no en *cómo* debe implementarse. Durante el proceso de abstracción es cuando se decide qué características y comportamiento debe tener el modelo.

Aplicando la abstracción se es capaz de construir, analizar y gestionar sistemas de computadoras complejos y grandes que no se podrían diseñar si se tratara de modelar a un nivel detallado. En cada nivel de abstracción se visualiza el sistema en términos de componentes, denominados **herramientas abstractas**, cuya composición interna se ignora. Esto nos permite concentrarnos en cómo cada componente interactúa con otros componentes y centrarnos en la parte del sistema que es más relevante para la tarea a realizar en lugar de perderse a nivel de detalles menos significativos.

En estructuras o registros, las propiedades individuales de los objetos se pueden almacenar en los miembros. Para los objetos es de interés *cómo* están organizados sino también *qué* se puede hacer con ellos. Es decir, las operaciones que forman la internan de un objeto son también importantes. El primer concepto en el mundo de la orientación a objetos nació con los tipos abstractos de datos (TAD). Un tipo abstracto de datos describe no sólo los atributos de un objeto, sino también su comportamiento (las operaciones). Esto puede incluir también una descripción de los estados que puede alcanzar un objeto.

Un medio de reducir la complejidad es la abstracción. Las características y los procesos se reducen a las propiedades esenciales, son resumidas o combinadas entre sí. De este modo, las características complejas se hacen más manejables.

Ejemplo 1.4

Diferentes modelos de abstracción del término coche (carro).

- Un `coche (carro)` es la combinación (o composición) de diferentes partes, tales como motor, carrocería, cuatro ruedas, cinco puertas, etc.
- Un `coche (carro)` es un concepto común para diferentes tipos de coches. Pueden clasificarse por el nombre del fabricante (Audi, BMW, SEAT, Toyota, Chrisler...), por su categoría (turismo, deportivo, todoterreno...), por el carburante que utilizan (gasolina, gasoil, gas, híbrido…).

La abstracción `coche` se utilizará siempre que la marca, la categoría o el carburante no sean significativos. Así, un `carro (coche)` se utilizará para transportar personas o ir de Carchelejo a Cazorla.

1.6.3. Encapsulación y ocultación de datos

El *encapsulado* o *encapsulación de datos* es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación. En el caso de los objetos que poseen las mismas características y

comportamiento se agrupan en clases, que no son más que unidades o módulos de programación que encapsulan datos y operaciones.

La ocultación de datos permite separar el aspecto de un componente, definido por su *interfaz* con el exterior, de sus detalles internos de implementación. Los términos ocultación de la información (*information hiding*) y encapsulación de datos (*data encapsulation*) se suelen utilizar como sinónimos, pero no siempre es así, y muy al contrario, son términos similares pero distintos. En C++ no es lo mismo, ya los datos internos están protegidos del exterior y no se pueden acceder a ellos más que desde su propio interior y por tanto, no están ocultos. El acceso al objeto está restringido sólo a través de una interfaz bien definida.

El diseño de un programa orientado a objetos contiene, al menos, los siguientes pasos;

1. Identificar los *objetos* del sistema.
2. Agrupar en *clases* a todos objetos que tengan características y comportamiento comunes.
3. Identificar los *datos* y *operaciones* de cada una de las clases.
4. Identificar las *relaciones* que pueden existir entre las clases.

En C++, un **objeto** es un elemento individual con su propia identidad; por ejemplo, un libro, un automóvil. Una **clase** puede describir las propiedades genéricas de un ejecutivo de una empresa (nombre, título, salario, cargo.) mientras que un objeto representará a un ejecutivo específico (Luis Mackoy, director general). En general, una clase define qué datos se utilizan para representar un objeto y las operaciones que se pueden ejecutar sobre esos datos.

Cada clase tiene sus propias características y comportamiento; en general, una clase define los datos que se utilizan y las operaciones que se pueden ejecutar sobre esos datos. Una clase describe un objeto. En el sentido estricto de programación, una clase es un tipo de datos. Diferentes variables se pueden crear de este tipo. En programación orientada a objetos, éstas se llaman *instancias*. Las instancias son, por consiguiente, la realización de los objetos descritos en una clase. Estas instancias constan de datos o atributos descritos en la clase y se pueden manipular con las operaciones definidas dentro de ellas.

Los términos *objeto* e *instancia* se utilizan frecuentemente como sinónimos (especialmente en C++). Si una variable de tipo `Carro` se declara, se crea un objeto `Carro` (una instancia de la clase `Carro`).

Las operaciones definidas en los objetos se llaman *métodos*. Cada operación llamada por un objeto se interpreta como un *mensaje* al objeto, que utiliza un método específico para procesar la operación.

En el diseño de programas orientados a objetos se realiza en primer lugar el diseño de las clases que representan con precisión aquellas cosas que trata el programa. Por ejemplo, un programa de dibujo, puede definir clases que representan rectángulos, líneas, pinceles, colores, etc. Las definiciones de clases, incluyen una descripción de operaciones permisibles para cada clase, tales como desplazamiento de un círculo o rotación de una línea. A continuación se prosigue el diseño de un programa utilizando objetos de las clases.

El diseño de clases fiables y útiles puede ser una tarea difícil. Afortunadamente, los lenguajes POO facilitan la tarea ya que incorporan clases existentes en su propia programación. Los fabricantes de software proporcionan numerosas bibliotecas de clases, incluyendo bibliotecas de clases diseñadas para simplificar la creación de programas para entornos tales como Windows, Linux, Macintosh o Unix. Uno de los beneficios reales de C++ es que permite la reutilización y adaptación de códigos existentes y ya bien probados y depurados.

1.6.4. Objetos

El objeto es el centro de la programación orientada a objetos. Un objeto es algo que se visualiza, se utiliza y juega un rol o papel. Si se programa con enfoque orientado a objetos, se intentan descubrir e implementar los objetos que juegan un rol en el dominio del problema y en consecuencia programa. La estructura interna y el comportamiento de un objetivo, en una primera fase, no tiene prioridad. Es importante que un objeto tal como un carro o una casa juegan un rol.

Dependiendo del problema, diferentes aspectos de un aspecto son relevantes. Un carro puede ser ensamblado de partes tales como un motor, una carrocería, unas puertas o puede ser descrito utilizando propiedades tales como su velocidad, su kilometraje o su fabricante. Estos atributos indican el objeto. De modo similar una persona, también se puede ver como un objeto, del cual se disponen de diferentes atributos. Dependiendo de la definición del problema, esos atributos pueden ser el nombre, apellido, dirección, número de teléfono, color del cabello, altura, peso, profesión, etc.

Un objeto no necesariamente ha de realizar algo concreto o tangible. Puede ser totalmente abstracto y también puede describir un proceso. Por ejemplo, un partido de baloncesto o de rugby puede ser descrito como un objeto. Los atributos de este objeto pueden ser los jugadores, el entrenador, la puntuación y el tiempo transcurrido de partido.

Cuando se trata de resolver un problema con orientación a objetos, dicho problema no se descompone en funciones como en programación estructurada tradicional, caso de C, sino en objetos. El pensar en términos de objetos tiene una gran ventaja: se asocian los objetos del problema a los objetos del mundo real.

¿Qué tipos de cosas son objetos en los programas orientados a objetos? La respuesta está limitada por su imaginación aunque se pueden agrupar en categorías típicas que facilitarán su búsqueda en la definición del problema de un modo más rápido y sencillo.

- RecursosHumanos:
  - Empleados.
  - Estudiantes.
  - Clientes.
  - Vendedores.
  - Socios.
- Colecciones de datos:
  - Arrays (arreglos).
  - Listas.
  - Pilas.
  - Árboles.
  - Árboles binarios.
  - Grafos.
- Tipos de datos definidos por usuarios:
  - Hora.
  - Números complejos.
  - Puntos del plano.
  - Puntos del espacio.
  - Ángulos.
  - Lados.
- Elementos de computadoras:
  - Menús.
  - Ventanas.
  - Objetos gráficos (rectángulos, círculos, rectas, puntos,…).
  - Ratón (mouse).
  - Teclado.
  - Impresora.
  - USB.
  - Tarjetas de memoria de cámaras fotográficas.
- Objetos físicos:
  - Carros.
  - Aviones.

— Trenes.
— Barcos.
— Motocicletas.
— Casas.

• Componentes de videojuegos:

— Consola.
— Mandos.
— Volante.
— Conectores.
— Memoria.
— Acceso a Internet.

La correspondencia entre objetos de programación y objetos del mundo real es el resultado eficiente de combinar datos y funciones que manipulan esos datos. Los objetos resultantes ofrecen una mejor solución al diseño del programa que en el caso de los lenguajes orientados a procedimientos.

Un **objeto** se puede definir desde el punto de vista conceptual como una entidad individual de un sistema y que se caracteriza por un estado y un comportamiento. Desde el punto de vista de implementación un **objeto** es una entidad que posee un conjunto de *datos* y un conjunto de *operaciones* (*funciones* o *métodos*).

El estado de un objeto viene determinado por los valores que toman sus datos, cuyos valores pueden tener las restricciones impuestas en la definición del problema. Los datos se denominan también *atributos* y componen la estructura del objeto y las operaciones —también llamadas *métodos*— representan los servicios que proporciona el objeto.

La representación gráfica de un objeto en **UML** se muestra en la Figura 1.14.

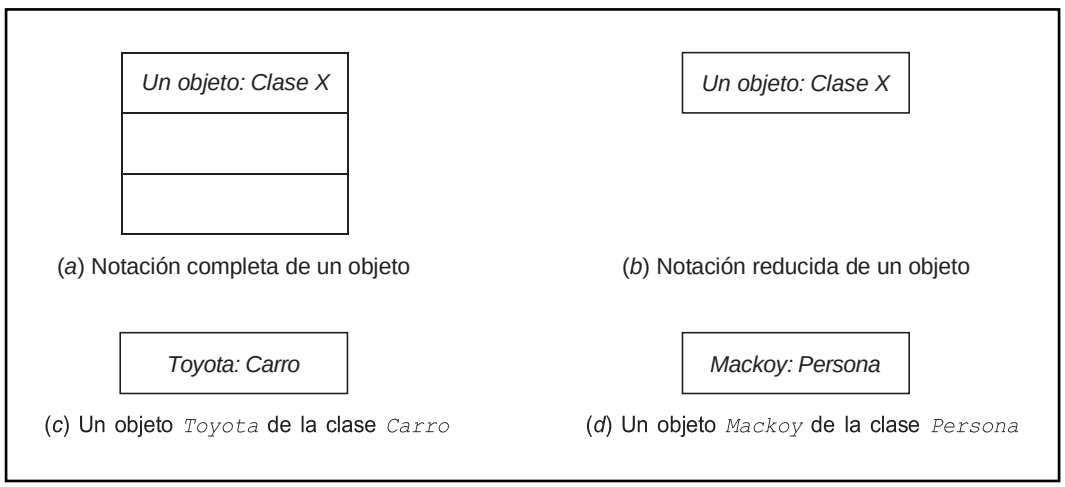


Figura 1.14. Representación de objetos en UML (Lenguaje Unificado de Modelado).

1.6.5. Clases

En POO los objetos son miembros de **clases**. En esencia, una clase es un tipo de datos al igual que cualquier otro tipo de dato definido en un lenguaje de programación. La diferencia reside en que la clase es un tipo de dato que contiene datos y funciones. Una clase contiene muchos objetos y es preciso definirla, aunque su definición no implica creación de objetos.

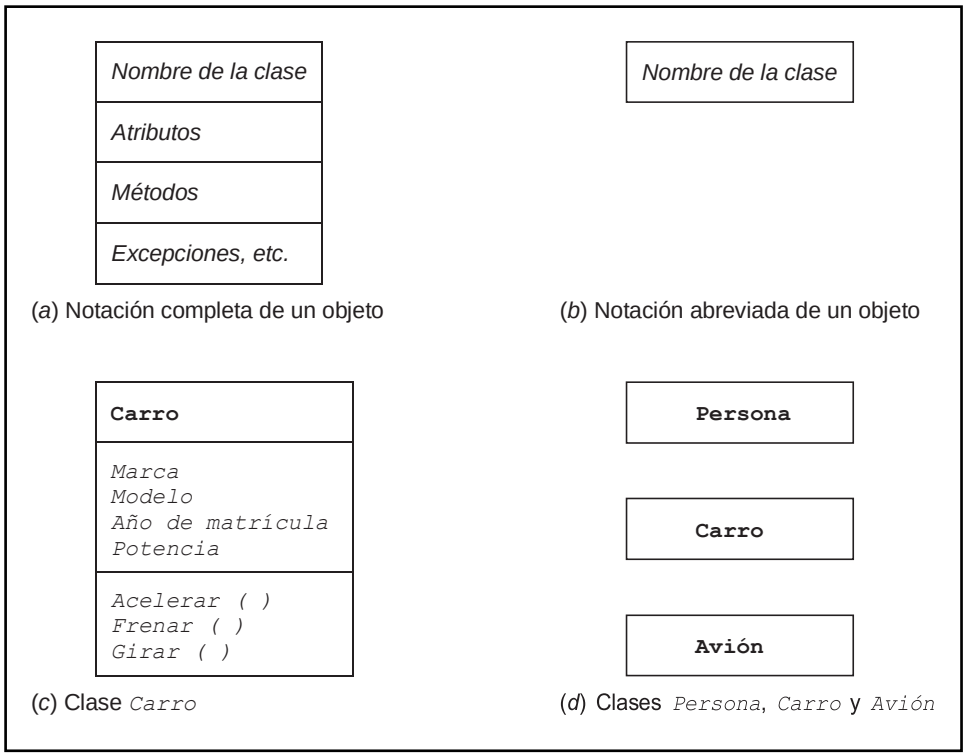


Figura 1.15. Representación de clases en UML.

Una clase es, por consiguiente, una descripción de un número de objetos similares. `Madonna`, `Sting`, `Prince`, `Juanes`, `Carlos Vives` o `Juan Luis Guerra` son miembros u objetos de la clase "músicos de rock". Un objeto concreto, `Juanes` o `Carlos Vives`, son *instancias* de la clase "`músicos de rock`".

En **C++** una clase es una estructura de dato o tipo de dato que contiene funciones (métodos) como miembros y datos. Una clase es una descripción general de un conjunto de objetos similares. Por definición todos los objetos de una clase comparten los mismos atributos (datos) y las mismas operaciones (métodos). Una clase encapsula las abstracciones de datos y operaciones necesarias para describir una entidad u objeto del mundo real.

Una clase se representa en **UML** mediante un rectángulo que contiene en una banda con el nombre de la clase y opcionalmente otras dos bandas con el nombre de sus atributos y de sus operaciones o métodos (Figura 1.16).

1.6.6. Generalización y especialización: herencia

La *generalización* es la propiedad que permite compartir información entre dos entidades evitando la redundancia. En el comportamiento de objetos existen con frecuencia propiedades que son comunes en diferentes objetos y esta propiedad se denomina generalización.

Por ejemplo, máquinas lavadoras, frigoríficos, hornos de microondas, tostadoras, lavavajillas, etc., son todos electrodomésticos (aparatos del hogar). En el mundo de la orientación a objetos, cada uno de estos aparatos es un **`subclase`** de la clase `Electrodoméstico` y a su vez `Electrodoméstico` es una **`superclase`** de todas las otras clases `(máquinas lavadoras, frigoríficos, hornos de microondas, tostadoras, lavavajillas,...)`. El proceso inverso de la generalización por el cual se definen nuevas clases a partir de otras ya existentes se denomina *especialización*

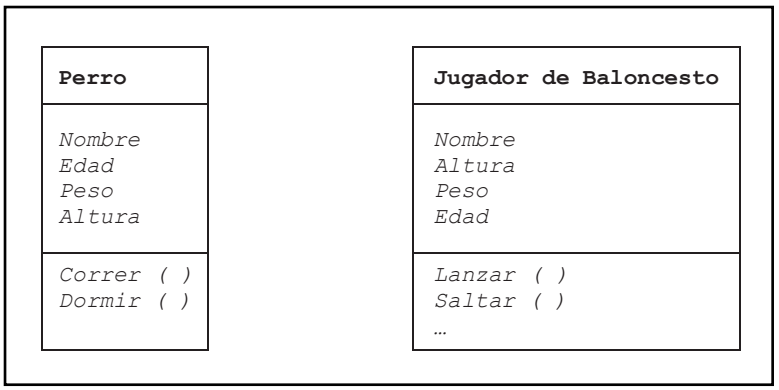


Figura 1.16. Representación de clases en UML con atributos y métodos.

En orientación a objetos, el mecanismo que implementa la propiedad de generalización se denomina **herencia.** La herencia permite definir nuevas clases a partir de otras clases ya existentes, de modo que presentan las mismas características y comportamiento de éstas, así como otras adicionales.

La idea de clases conduce a la idea de herencia. Clases diferentes se pueden conectar unas con otras de modo jerárquico. Como ya se ha comentado anteriormente con las relaciones de generalización y especialización, en nuestras vidas diarias se utiliza el concepto de clases divididas en subclases. La clase `animal` se divide en `anfibios`, `mamíferos`, `insectos`, `pájaros`, etc., y la clase `vehículo` en `carros`, `motos`, `camiones`, `buses`, etc.

El principio de la división o clasificación es que cada subclase comparte características comunes con la clase de la que procede o se deriva. Los carros, motos, camiones y buses tiene ruedas, motores y carrocerías; son las características que definen a un vehículo. Además de las características comunes con los otros miembros de la clase, cada subclase tiene sus propias características. Por ejemplo los camiones tienen una cabina independiente de la caja que transporta la carga; los buses tienen un gran número de asientos independientes para los viajeros que ha de transportar, etc. En la Figura 1.17 se muestran clases pertenecientes a una jerarquía o herencia de clases.

De modo similar una clase se puede convertir en padre o raíz de otras subclases**.** En C++ la clase original se denomina *clase base* y las clases que se derivan de ella se denominan *clases derivadas* y siempre son una especialización o *concreción* de su clase base. A la inversa, la clase base es la generalización de la clase derivada. Esto significa que todas las propiedades (atributos y operaciones) de la clase base se heredan por la clase derivada, normalmente suplementada con propiedades adicionales.

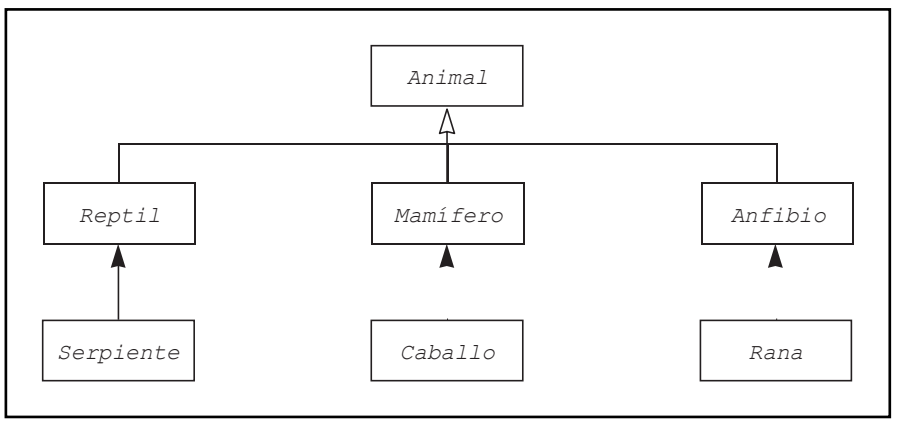


Figura 1.17. Herencia de clases en UML.

1.6.7 Reusabilidad

Una vez que una clase ha sido escrita, creada y depurada, se puede distribuir a otros programadores para utilizar en sus propios programas. Esta propiedad se llama *reusabilidad* [5] o *reutilización.* Su concepto es similar a las funciones incluidas en las bibliotecas de funciones de un lenguaje procedimental como C que se pueden incorporar en diferentes programas.

En C++, el concepto de herencia proporciona una extensión o ampliación al concepto de *reusabilidad.* Un programador puede considerar una clase existente y sin modificarla, añadir competencias y propiedades adicionales a ella. Esto se consigue derivando una nueva clase de una ya existente. La nueva clase heredará las características de la clase antigua, pero es libre de añadir nuevas características propias.

La facilidad de reutilizar o reusar el software existente es uno de los grandes beneficios de la POO: muchas empresas consiguen con la reutilización de clase en nuevos proyectos la reducción de los costes de inversión en sus presupuestos de programación. ¿En esencia cuales son las ventajas de la herencia? Primero, se utiliza para consistencia y reducir código. Las propiedades comunes de varias clases sólo necesitan ser implementadas una vez y sólo necesitan modificarse una vez si es necesario. La otra ventaja es que el concepto de abstracción de la funcionalidad común está soportada.

1.6.8. Polimorfismo

Además de las ventajas de consistencia y reducción de código, la herencia, aporta también otra gran ventaja: facilitar el polimorfismo. Polimorfismo es la propiedad de que un operador o una función actúen de modo diferente en función del objeto sobre el que se aplican. En la practica, el polimorfismo significa la capacidad de una operación de ser interpretada sólo por el propio objeto que lo invoca. Desde un punto de vista práctico de ejecución del programa, el polimorfismo se realiza en tiempo de ejecución ya que durante la compilación no se conoce qué tipo de objeto y por consiguiente que operación ha sido llamada. En el capítulo 14 se describirá en profundidad la propiedad de polimorfismo y los diferentes modos de implementación del polimorfismo.

La propiedad de **polimorfismo** es aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase. Así, por ejemplo, la operación de abrir se puede dar en diferentes clases: abrir una puerta, abrir una ventana, abrir un periódico, abrir un archivo, abrir una cuenta corriente en un banco, abrir un libro, etc. En cada caso se ejecuta una operación diferente aunque tiene el mismo nombre en todos ellos “abrir”. El polimorfismo es la propiedad de una operación de ser interpretada sólo por el objeto al que pertenece. Existen diferentes formas de implementar el polimorfismo y variará dependiendo del lenguaje de programación.

Veamos el concepto con ejemplos de la vida diaria.

En un taller de reparaciones de automóviles existen numerosos carros, de marcas diferentes, de modelos diferentes, de tipos diferentes, potencias diferentes, etc. Constituyen una clase o colección heterogénea de carros (coches). Supongamos que se ha de realizar una operación común “cambiar los frenos del carro”. La operación a realizar es la misma, incluye los mismos principios, sin embargo, dependiendo del coche, en particular, la operación será muy diferente, incluirá diferentes acciones en cada caso. Otro ejemplo a considerar y relativo a los operadores “+” y “*” aplicados a números enteros o números complejos; aunque ambos son números, en un caso la suma y multiplicación son operaciones simples, mientras que en el caso de los números complejos al componerse de parte real y parte imaginaria, será necesario seguir un método específico para tratar ambas partes y obtener un resultado que también será un número complejo.

El uso de operadores o funciones de forma diferente, dependiendo de los objetos sobre los que están actuando se llama polimorfismo (una cosa con diferentes formas). Sin embargo, cuando un operador

[5] El término proviene del concepto ingles *reusability*. La traducción no ha sido aprobada por la RAE, pero se incorpora al texto por su gran uso y difusión entre los profesionales de la informática.

existente, tal como + o =, se le permite la posibilidad de operar sobre nuevos tipos de datos, se dice entonces que el operador está sobrecargado. La sobrecarga es un tipo de polimorfismo y una característica importante de la POO. En el Capítulo 10 se ampliará, también en profundidad, este nuevo concepto.

1.7. EL *SOFTWARE* (LOS PROGRAMAS)

El *software* de una computadora es un conjunto de instrucciones de programa detalladas que controlan y coordinan los componentes *hardware* de una computadora y controlan las operaciones de un sistema informático. El auge de las computadoras en el siglo pasado y en el actual siglo XXI, se debe esencialmente, al desarrollo de sucesivas generaciones de software potentes y cada vez más *amistosas* ("fáciles de utilizar").

Las operaciones que debe realizar el *hardware* son especificadas por una lista de instrucciones, llamadas programas, o *software*. Un programa de software es un conjunto de **sentencias** o **instrucciones** al computador. El proceso de escritura o codificación de un programa se denomina **programación** y las personas que se especializan en esta actividad se denominan **programadores**. Existen dos tipos importantes de software: software del sistema y software de aplicaciones. Cada tipo realiza una función diferente.

***Software* del sistema** es un conjunto generalizado de programas que gestiona los recursos del computador, tal como el procesador central, enlaces de comunicaciones y dispositivos periféricos. Los programadores que escriben software del sistema se llaman **programadores de sistemas**. ***Software* de aplicaciones** son el conjunto de programas escritos por empresas o usuarios individuales o en equipo y que instruyen a la computadora para que ejecute una tarea específica. Los programadores que escriben software de aplicaciones se llaman **programadores de aplicaciones**.

Los dos tipos de software están relacionados entre sí, de modo que los usuarios y los programadores pueden hacer así un uso eficiente del computador. En la Figura 1.18 se muestra una vista organizacional de un computador donde muestran los diferentes tipos de software a modo de capas de la computadora desde su interior (el hardware) hasta su exterior (usuario): Las diferentes capas funcionan gracias a las instrucciones específicas (instrucciones máquina) que forman parte del software del sistema y llegan al software de aplicación, programado por los programadores de aplicaciones, que es utilizado por el usuario que no requiere ser un especialista.

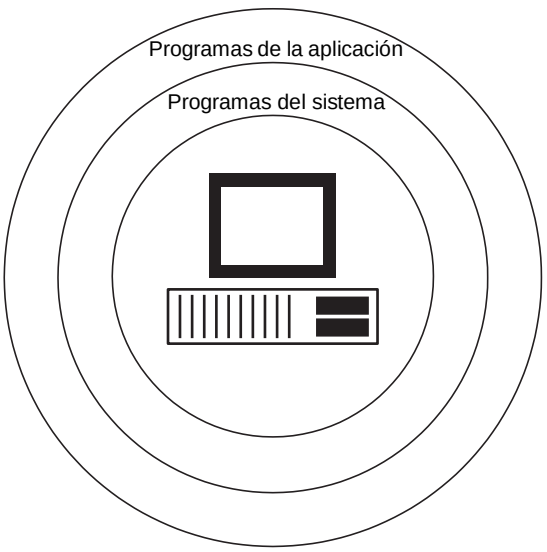


Figura 1.18. Relación entre programas de aplicación y programas del sistema.

1.7.1 Software del sistema

El software del sistema coordina las diferentes partes de un sistema de computadora y conecta e interactúa entre el software de aplicación y el hardware de la computadora. Otro tipo de software del sistema que gestiona controla las actividades de la computadora y realiza tareas de proceso comunes, se denomina *utility* o **utilidades** (en algunas partes de Latinoamérica, **utilerías**). El software del sistema que gestiona y controla las actividades del computador se denomina **sistema operativo**. Otro software del sistema son los programas traductores o de traducción de lenguajes de computador que convierten los lenguajes de programación, entendibles por los programadores, en lenguaje máquina que entienden las computadoras

El **software del sistema** es el conjunto de programas indispensables para que la máquina funcione; se denominan también *programas del sistema*. Estos programas son, básicamente, *el sistema operativo, los editores de texto,* los *compiladores/intérpretes* (lenguajes de programación) y los *programas de utilidad*.

1.7.2. Software de aplicación

El software de aplicación tiene como función principal asistir y ayudar a un usuario de un computador para ejecutar tareas específicas. Los programas de aplicación se pueden desarrollar con diferentes lenguajes y herramientas de software. Por ejemplo, una aplicación de procesamiento de textos (*word processing)* tal como Word o Word Perfect que ayuda a crear documentos, una hoja de cálculo tal como Lotus 1-2-3 o Excel que ayudan a automatizar tareas tediosas o repetitivas de cálculos matemáticos o estadísticos, a generar diagramas o gráficos, presentaciones visuales como PowerPoint, o a crear bases de datos como Acces u Oracle que ayudan a crear archivos y registros de datos.

Los usuarios, normalmente, compran el software de aplicaciones en discos CDs o DVDs (antiguamente en disquetes) o los descargan (bajan) de la Red Internet y han de instalar el software copiando los programas correspondientes de los discos en el disco duro de la computadora. Cuando compre estos programas asegúrese que son compatibles con su computador y con su sistema operativo. Existe una gran diversidad de programas de aplicación para todo tipo de actividades tanto de modo personal, como de negocios, navegación y manipulación en Internet, gráficos y presentaciones visuales, etc.

Los *lenguajes de programación* sirven para escribir programas que permitan la comunicación usuario/máquina. Unos programas especiales llamados *traductores* (**compiladores** o **intérpretes**) convierten las instrucciones escritas en lenguajes de programación en instrucciones escritas en lenguajes máquina (0 y 1, *bits*) que ésta pueda entender.

Los *programas de utilidad* [6] facilitan el uso de la computadora. Un buen ejemplo es un *editor de textos* que permite la escritura y edición de documentos. Este libro ha sido escrito en un editor de textoso *procesador de palabras* ("***word procesor***").

Los programas que realizan tareas concretas, nóminas, contabilidad, análisis estadístico, etc., es decir, los programas que podrá escribir en Turbo Pascal, se denominan *programas de aplicación*. A lo largo del libro se verán pequeños programas de aplicación que muestran los principios de una buena programación de computadora.

Se debe diferenciar entre el acto de crear un programa y la acción de la computadora cuando ejecuta las instrucciones del programa. La creación de un programa se hace inicialmente en papel y, a continuación, se introduce en la computadora y se convierte en lenguaje entendible por la computadora. La Figura 1.19 muestra el proceso general de ejecución de un programa con una entrada (*datos*) al programa y la obtención de una salida (*resultados*). La entrada puede tener una variedad de formas, tales como texto, información numérica, imágenes o sonido. La salida puede también tener formas, tales como datos numéricos o caracteres, señales para controlar equipos o robots, etc.

[6] *Utility*: programa de utilidad o *utilería*.

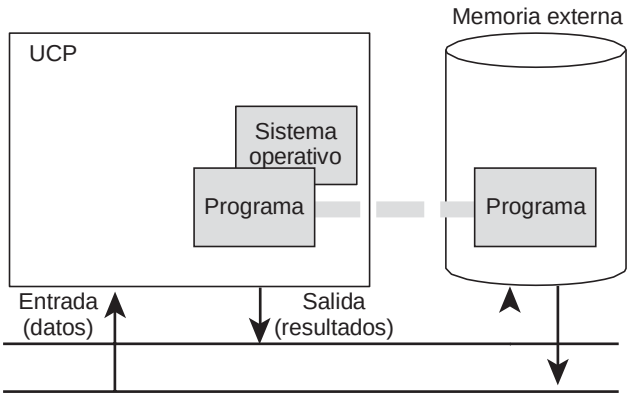


Figura 1.19. Ejecución de un programa.

1.8 SISTEMA OPERATIVO

Un sistema operativo **SO** (*Operating System, OS)* es tal vez la parte más importante del software del sistema y es el software que controla y gestiona los recursos del computador. En la práctica, el sistema operativo es la colección de programas de computador que controla la interacción del usuario y el hardware del computador. El sistema operativo es el administrador principal del computador, y por ello a veces, se le compara con el director de una orquesta ya que este software es el responsable de dirigir todas las operaciones del computador y gestionar todos sus recursos.

El sistema operativo asigna recursos, planifica el uso de recursos y tareas del computador, y monitoriza a las actividades del sistema informático. Estos recursos incluyen memoria, dispositivos de **E/S** (Entrada/Salida), y la **UCP** (**U**nidad **C**entral de **P**roceso). El sistema operativo proporciona servicios tales como asignar memoria a un programa y manipulación del control de los dispositivos de E/S tales como el monitor el teclado o las unidades de disco. La Tabla 1.6 muestra algunos de los sistemas operativos más populares utilizados en enseñanza y en informática profesional.

Cuando un usuario interactúa con un computador, la interacción está controlada por el sistema operativo. Un usuario se comunica con un sistema operativo a través de una interfaz de usuario de ese sistema operativo. Los sistemas operativos modernos utilizan una interfaz gráfica de usuario, **IGU** (*Graphical User Interface, GUI*) que hace uso masivo de iconos, botones, barras y cuadros de diálogo para realizar tareas que se controlan por el teclado o el ratón (mouse) entre otros dispositivos.

Normalmente, el sistema operativo se almacena de modo permanente en un chip de memoria de sólo lectura (ROM) de modo que esté disponible tan pronto el computador se pone en marcha ("se enciende" o "se prende"). Otra parte del sistema operativo puede residir en disco que se almacena en memoria RAM en la inicialización del sistema por primera vez en una operación que se llama c*arga* del sistema (*booting*).

Uno de los programas más importante es el **sistema operativo**, que sirve, esencialmente, para facilitar la escritura y uso de sus propios programas. El sistema operativo dirige las operaciones globales de la computadora, instruye a la computadora para ejecutar otros programas y controla el almacenamiento y recuperación de archivos (programas y datos) de cintas y discos. Gracias al sistema operativo es posible que el programador pueda introducir y grabar nuevos programas, así como instruir a la computadora para que los ejecute. Los sistemas operativos pueden ser: *monousuarios* (un solo usuario) y *multiusuarios*, o tiempo compartido (diferentes usuarios); atendiendo al número de usuarios y *monocarga* (una sola tarea) o *multitarea* (múltiples tareas) según las tareas (procesos) que puede realizar simultáneamente. C++ corre prácticamente en todos los sistemas operativos, Windows XP, Windows 95, Windows NT, Windows 2000, UNIX, Linux, Vista..., y en casi todas las computadoras personales actuales PC, Mac, Sun, etc.

Tabla 1.6. Sistemas operativos más utilizados en educación y en la empresa.

Sistema operativo	Características
Windows Vista[7]	Nuevo sistema operativo de Microsoft presentado en 2006, pero que se lanzará comercialmente en 2007.
Windows XP	Sistema operativo más utilizado en la actualidad, tanto en el campo de la enseñanza, como en la industria y negocios. Su fabricante es Microsoft.
Windows 98/ME/2000	Versiones anteriores de Windows pero que todavía hoy son muy utilizados.
UNIX	Sistema operativo abierto, escrito en C y todavía muy utilizado en el campo profesional.
Linux	Sistema operativo de software abierto, gratuito y de libre distribución, similar a UNIX, y una gran alternativa a Windows. Muy utilizado actualmente en servidores de aplicaciones para Internet.
Mac OS	Sistema operativo de las computadoras Apple Macintosh.
DOS y OS/2	Sistemas operativos creados por Microsoft e IBM respectivamente, ya poco utilizados pero que han sido la base de los actuales sistemas operativos.
CP/M	Sistema operativo de 8 bits para las primeras microcomputadoras nacidas en la década de los setenta.
Symbian	Sistema operativo para teléfonos móviles apoyado fundamentalmente por el fabricante de teléfonos celulares Nokia.
PalmOS	Sistema operativo para agendas digitales, PDA, del fabricante Palm.
Windows Mobile, CE	Sistema operativo para teléfonos móviles con arquitectura y apariencias similares a Windows XP.

1.8.1. Tipos de sistemas operativos

Las diferentes características especializadas del sistema operativo permiten a los computadores manejar muchas diferentes tareas así como múltiples usuarios de modo simultáneo o en paralelo, bien de modo secuencial... En base a sus características específicas los sistemas operativos se pueden clasificar en varios grupos:

Multiprogramación/Multitarea

La multiprogramación permite a múltiples programas compartir recursos de un sistema de computadora en cualquier momento a través del uso concurrente de una UCP. Sólo un programa utiliza realmente la UCCP en cualquier momento dado, sin embargo, las necesidades de entrada/salida pueden ser atendidas en el mismo momento. Dos o más programas están activos al mismo tiempo, pero no utilizan los recursos del computador simultáneamente. Con multiprogramación, un grupo de programas se ejecutan alternativamente y se alternan en el uso del procesador. Cuando se utiliza un sistema operativo de un único usuario, la multiprogramación toma el nombre de **multitarea**.

Multiprogramación

Método de ejecución de dos o más programas concurrentemente utilizando la misma computadora. La UCO ejecuta sólo un programa pero puede atender los servicios de entrada/salida de los otros al mismo tiempo.

[7] Microsoft tiene previsto presentar en el año 2006, un nuevo sistema operativo llamado Windows Vista, actualización de Windows XP pero con numerosas funcionalidades, especialmente de Internet y de seguridad, incluyendo en el sistema operativo programas que actualmente se comercializan independientes, tales como programas de reproducción de música, vídeo, y fundamentalmente un sistema de representación gráfica muy potente que permitirá construir aplicaciones en tres dimensiones, así como un buscador, un sistema antivirus y otras funcionalidades importantes.

Tiempo compartido (múltiples usuarios, time sharing)

Un sistema operativo multiusuario es un sistema operativo que tiene la capacidad de permitir que muchos usuarios compartan simultáneamente los recursos de proceso de la computadora. Centenas o millares de usuarios se pueden conectar al computador que asigna un tiempo de computador a cada usuario, de modo que a medida que se libera la tarea de un usuario, se realiza la tarea del siguiente, y así sucesivamente. Dada la alta velocidad de transferencia de las operaciones, la sensación es de que todos los usuarios están conectados simultáneamente a la UCP con cada usuario recibiendo únicamente un tiempo de máquina.

Multiproceso

Un sistema operativo trabaja en multiproceso cuando puede enlazar a dos o más UCPs para trabajar en paralelo en un único sistema de computadora. El sistema operativo puede asignar múltiples UCPs para ejecutar diferentes instrucciones del mismo programa o de programas diferentes simultáneamente, dividiendo el trabajo entre las diferentes UCP.

> La multiprogramación utiliza proceso concurrente con una CPU; el multiproceso utiliza proceso simultáneo con múltiples CPUs.

1.9. LENGUAJES DE PROGRAMACIÓN

Como se ha visto en el apartado anterior, para que un procesador realice un proceso se le debe suministrar en primer lugar un algoritmo adecuado. El procesador debe ser capaz de *interpretar* el algoritmo, lo que significa:

- comprender las instrucciones de cada paso,
- realizar las operaciones correspondientes.

Cuando el procesador es una computadora, el algoritmo se ha de expresar en un formato que se denomina *programa,* ya que el pseudocódigo o el diagrama de flujo no son comprensibles por la computadora, aunque pueda entenderlos cualquier programador. Un programa se escribe en un *lenguaje de programación* y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman *programación*. Así pues, los lenguajes utilizados para escribir programas de computadoras son los *lenguajes de programación* y **programadores** son los escritores y diseñadores de programas. El proceso de traducir un algoritmo en *pseudocódigo* a un lenguaje de programación se denomina **codificació**n, y el algoritmo escrito en un lenguaje de programación se denomina **código fuente**.

En la realidad la computadora no entiende directamente los lenguajes de programación sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente, pero muy complejo para las personas; este lenguaje se conoce como **lenguaje máquina** y el código correspondiente **código máquina**. Los programas que traducen el código fuente escrito en un lenguaje de programación —tal como C++— a código máquina se denominan **traductores**. El proceso de conversión de un algoritmo escrito en pseudocódigo hasta un programa ejecutable comprensible por la máquina, se muestra en la Figura 1.20.

Hoy en día, la mayoría de los programadores emplean lenguajes de programación como C++, C, C#, Java, Visual Basic, XML, HTML, Perl, PHP, JavaScript..., aunque todavía se utilizan, sobre todo profesionalmente, los clásicos *COBOL, FORTRAN, Pascal* o el mítico BASIC. Estos lenguajes se denominan **lenguajes de alto nivel** y permiten a los profesionales resolver problemas convirtiendo sus algoritmos en programas escritos en alguno de estos lenguajes de programación.

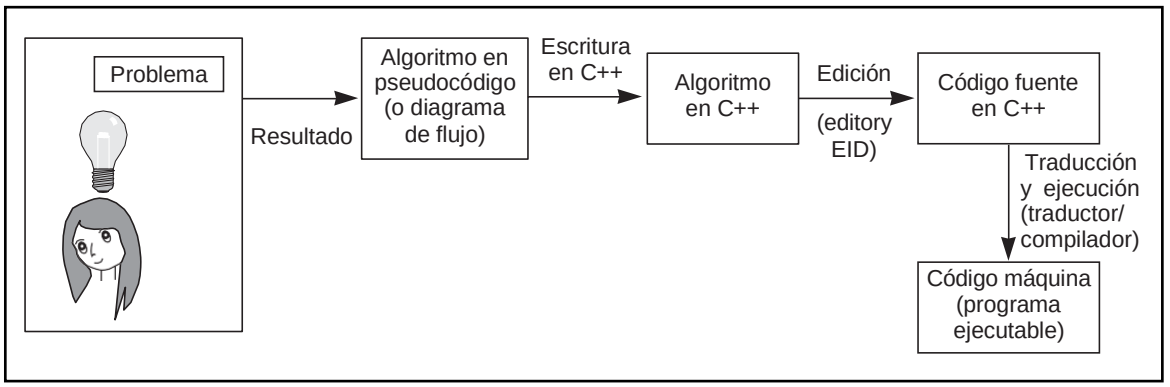


Figura 1.20. Proceso de transformación de un algoritmo en pseudocódigo en un programa ejecutable.

1.9.1. Traductores de lenguaje: el proceso de traducción de un programa

El proceso de traducción de un programa fuente escrito en un lenguaje de alto nivel a un lenguaje máquina comprensible por la computadora, se realiza mediante programas llamados "traductores". Los **traductores de lenguaje** son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en **compiladores** e **interpretes**.

Intérpretes

Un *intérprete* es un traductor que toma un programa fuente, lo traduce y, a continuación, lo ejecuta. Los programas intérpretes clásicos como BASIC, prácticamente ya no se utilizan, más que en circunstancias especiales. Sin embargo, está muy extendida la versión interpretada del lenguaje Smalltalk, un lenguaje orientado a objetos puro. El sistema de traducción consiste en: traducir la primera sentencia del programa a lenguaje máquina, se detiene la traducción, se ejecuta la sentencia; a continuación, se traduce la siguiente sentencia, se detiene la traducción, se ejecuta la sentencia y así sucesivamente hasta terminar el programa.

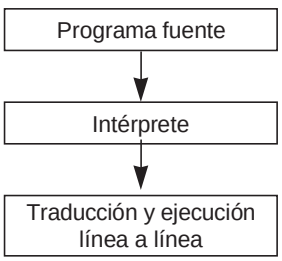


Figura 1.21. Intérprete.

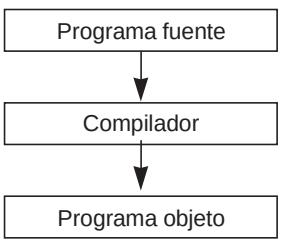


Figura 1.22. La compilación de programas.

Compiladores

Un *compilador* es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina. La traducción del programa completo se realiza en una sola operación denominada **compilación** del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque. El programa compilado y depurado (eliminados los errores del código fuente) se denomina *programa ejecutable* porque ya se puede ejecutar directamente y cuantas veces se desee; sólo deberá volver a compilarse de nuevo en el caso de que se modifique alguna instrucción del programa. De este modo el programa ejecutable no necesita del compilador para su ejecución. Los lenguajes compiladores típicos más utilizados son: **C**, **C++**, **Java**, **C#**, **Pascal**, **FORTRAN** y **COBOL**.

1.9.2. La compilación y sus fases

La *compilación* es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina.

Para conseguir el programa máquina real se debe utilizar un programa llamado *montador* o *enlazador* (*linker*). El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable (Figura 1.23).

El proceso de ejecución de un programa escrito en un lenguaje de programación y mediante un compilador suele tener los siguientes pasos:

1. Escritura del *programa fuente* con un *editor* (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (por ejemplo, un disco).
2. Introducir el programa fuente en memoria.
3. *Compilar* el programa con el compilador C.
4. *Verificar* y *corregir errores de compilación* (listado de errores).
5. Obtención del programa *objeto*.
6. El enlazador (*linker*) obtiene el *programa ejecutable*.
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.

El proceso de ejecución sería el mostrado en las figuras 1.24 y 1.25.

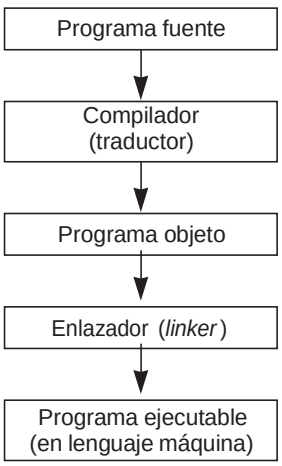


Figura 1.23. Fases de la compilación.

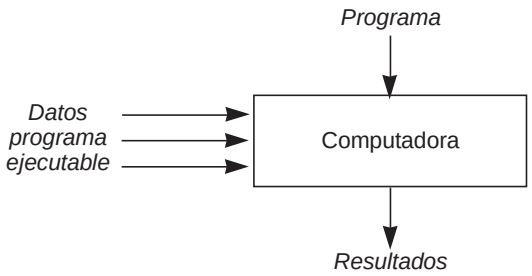


Figura 1.24. Ejecución de un programa.

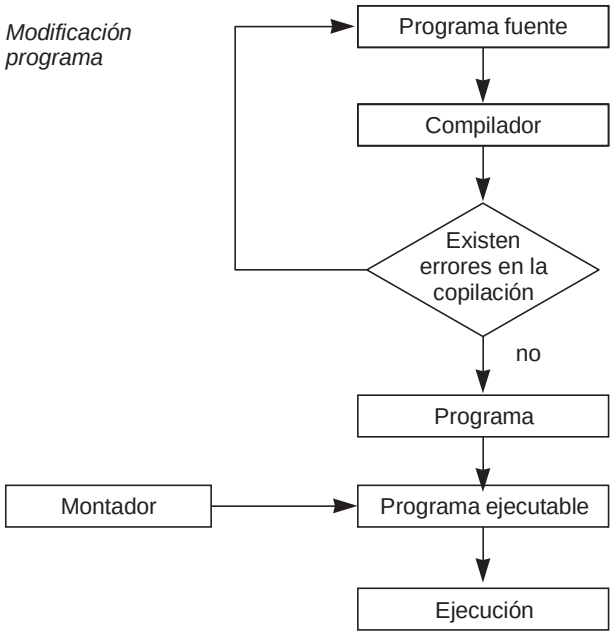


Figura 1.25. Fases de ejecución de un programa.

En el Capítulo 2 se describirá en detalle el proceso completo y específico de ejecución de programas en lenguaje C.

1.10. C: EL ORIGEN DE C++ COMO LENGUAJE UNIVERSAL

C es el lenguaje de programación de propósito general asociado, de modo universal, al sistema operativo UNIX. Sin embargo, la popularidad, eficacia y potencia de C, se ha producido porque este lenguaje no está prácticamente asociado a ningún sistema operativo, ni a ninguna máquina, en especial. Ésta es la razón fundamental, por la cual C, es conocido como el ***lenguaje de programación de sistemas, por excelencia***.

C es una evolución de los lenguajes BCPL —desarrollado por Martin Richards— y B —desarrollado por Ken Thompson en 1970— para el primitivo INIX de la computadora DEC PDP-7.

C nació realmente en 1978, con la publicación de *The C Programming Languaje*, por Brian Kernighan y Dennis Ritchie (Prentice Hall, 1978). Desde su nacimiento, C fue creciendo en popularidad y los

sucesivos cambios en el lenguaje a lo largo de los años junto a la creación de compiladores por grupos no involucrados en su diseño, hicieron necesario pensar en la estandarización de la definición del lenguaje C.

Así, en 1983, el American National Standard Institute (ANSI), una organización internacional de estandarización, creó un comité (el denominado X3J11) cuya tarea fundamental consistía en hacer *"una definición no ambigua del lenguaje C, e independiente de la máquina"*. Había nacido el estándar ANSI del lenguaje C. Con esta definición de C se asegura que cualquier fabricante de software que vende un compilador ANSI C incorpora todas las características del lenguaje, especificadas por el estándar. Esto significa también que los programadores que escriban programas en C estándar tendrán la seguridad de que correrán sus modificaciones en cualquier sistema que tenga un compilador C.

C es un *lenguaje de alto nivel*, que permite programar con instrucciones de lenguaje de propósito general. También, C se define como un lenguaje de programación estructurado de propósito general; aunque en su diseño también primó el hecho que fuera especificado como un lenguaje de programación de Sistemas, lo que proporciona una enorme cantidad de potencia y flexibilidad.

El estándar ANSI C formaliza construcciones no propuestas en la primera versión de C, en especial, asignación de estructuras y enumeraciones. Entre otras aportaciones, se definió esencialmente, una nueva forma de declaración de funciones (prototipos). Pero es, esencialmente, la biblioteca estándar de funciones, otra de las grandes aportaciones.

Hoy, en el siglo XXI, C sigue siendo uno de los lenguajes de programación más utilizados en la industria del software, así como en institutos tecnológicos, escuelas de ingeniería y universidades. Prácticamente todos los fabricantes de sistemas operativos, Windows, UNIX, Linux, MacOS, Solaris. , soportan diferentes tipos de compiladores de lenguaje C y en muchas ocasiones distribuciones gratuitas bajo cualquiera de los sistemas operativos citados. Todos los compiladores de C++ pueden ejecutar programas escritos en lenguaje C, preferentemente si cumplen el estándar ANSI C.[8]

1.11. EL LENGUAJE C++: HISTORIA Y CARACTERÍSTICAS

C++, Java y C#, los tres lenguajes más populares junto con C en esta primera década del siglo XXI son herederos directos del propio C con características orientadas a objetos y a Internet. Actualmente, y aunque C sigue siendo, tal vez, el más utilizado en el mundo de la educación como primer lenguaje de programación y también copa un porcentaje alto de utilización en el campo profesional, los tres lenguajes con características técnicas de orientación a objetos forman con C el *poker* de lenguajes más empleados en el mundo educativo, profesional y científico actual y previsiblemente de los próximos años.

C++ es heredero directo del lenguaje C que, a su vez, se deriva del lenguaje B [Richards, 1980]. C se mantiene como un subconjunto de C++. Otra fuente de inspiración, como señala su autor Bjarne Stroustrup [Stroustrup, 1997][9] fue Simula 67 [Dahl, 1972] del que tomó el concepto de clase (con clases derivadas y funciones virtuales).

El lenguaje de programación C fue desarrollado por **Dennis Ritche** de AT&T Bell Laboratories que se utilizó para escribir y mantener el sistema operativo UNIX (hasta que apareció C, el sistema operativo UNIX fue desarrollado por **Ken Thompson** en AT&T Bell Laboratories mediante en lenguaje ensamblador o en B). C es un lenguaje de propósito general que se puede utilizar para escribir cualquier tipo de programa, pero su éxito y popularidad está especialmente relacionado con el sistema operativo UNIX. (Fue desarrollado como *lenguaje de programación de sistemas*, es decir, un lenguaje de programación para escribir *sistemas operativos* y utilidades (programas) del sistema.) Los sistemas operativos

[8] Opciones gratuitas buenas puede encontrar en el sitio del fabricante de software Borland. También puede encontrar y descargar un compilador excelente Dev-C++ en software libre que puede compilar código C y también código C++, en `www.bloodshed.net` y en `www.download.com` puede asímismo encontrar diferentes compiladores totalmente gratuitos. Otros numerosos sitios puede encontrar en software gratuito en numerosos sitios de la red. Los fabricantes de software y de computadoras (IBM, Microsoft, HP...) ofrecen versiones a sus clientes aunque normalmente no son gratuitos

[9] P. 11

son los programas que gestionan (administran) los *recursos de la computadora*. Ejemplos bien conocidos de sistemas operativos además de UNIX son MS/DOS, OS/2, MVS, Lynux, Windows 95/98, Windows NT, Windows 2000, OS Mac, etc.

La especificación formal del lenguaje C es un documento escrito por Ritchie, titulado *The C Reference Manual*. En 1997, **Ritchie** y **Brian Kernighan**, ampliaron ese documento y publicaron un libro referencia del lenguaje *The C Programming Language* (también conocido por el K&R).

Aunque C es un lenguaje muy potente, tiene dos características que lo hacen inapropiado como una introducción moderna a la programación. Primero, C requiere un nivel de sofisticación a sus usuarios que les obliga a un difícil aprendizaje a los programadores principiantes ya que es de comprensión difícil. Segundo C, fue diseñado al principio de los setenta, y la naturaleza de la programación ha cambiado de modo significativo en la década de los ochenta y noventa.

Para subsanar estas "deficiencias" Bjarne Stroustrup de AT&T Bell Laboratories desarrolló C++ al principio de la década de los ochenta. Stroustrup diseñó C++ como un mejor C. En general, C estándar es un subconjunto de C++ y la mayoría de los programas C son también programas C++ (la afirmación inversa no es verdadera). C++ además de añadir propiedades a C, presenta características y propiedades de *programación orientada a objetos*, que es una técnica de programación muy potente y que se verá en la última parte de este libro.

Se han presentado varias versiones de C++ y su evolución se estudió en [Stroustrup 94]. Las características más notables que han ido incorporándose a C++ son: herencia múltiple, *genericidad*, plantillas, funciones virtuales, excepciones, etc. C++ ha ido evolucionando año a año y como su autor ha explicado: "*evolucionó siempre para resolver problemas encontrados por los usuarios y como consecuencia de conversaciones entre el autor, sus amigos y sus colegas*"[10].

IMPORTANTE: Página oficial de Bjarne Stroustrup

Bjarne Stroustrupo, diseñador e implementador del lenguaje de programación C++ es la referencia fundamental y definitiva para cualquier estudiante y programador de C++. Sus obras *The C++ Programming Language*[11]*,The Design and Evolution of C++ y C++. Reference Manual* son lectura y consulta obligada.

Su sitio web personal de AT&T Labs Researchs debe ser el primer sitio "favorito" que le recomendamos visite con cierta frecuencia.

www.resarch.att.com/~bs

El sitio es actualizado con frecuencia por Stroustrup y contiene gran cantidad de información y una excelente sección de FAQ (*frequently asked questions*).

C++ comenzó su proyecto de estandarización ante el comité ANSI y su primera referencia es *The Annotated C++ Reference Manual* [**Ellis** 89][12]. En diciembre de 1989 se reunió el comité X3J16 del ANSI por iniciativa de Hewlett Packard. En junio de 1991, a la estandarización de ANSI se unió ISO (*International Organization for Standardization*) con su propio comité (ISO-WG-21), creando un esfuerzo común ANSI/ISO para desarrollar un estándar para C++. Estos comités se reunen tres veces al año para aunar sus esfuerzos y llegar a una decisión de creación de un estándar que convirtiera a C++ en un lenguaje importante y de amplia difusión.

En 1995, el Comité publicó diferentes artículos (*working paper*) y en abril de ese año, se publicó un borrador del estándar (Comité Draft) para su examen público. En diciembre de 1996 se lanzó una segunda versión (CD2) a dominio público. Estos documentos no sólo refinaron la descripción de las caracte-

[10] [Stroustrup 98], p. 12

[11] Esta obra ha sido traducida por un equipo de profesores de la universidad pontificia de Salamanca que coordinó y dirigió el autor de este libro.

[12] Existe versión española de Addison-Wesley Díaz de Santos y traducida por los profesores Manuel Katrib y Luis Joyanes.

rísticas existentes, de C++ sino que también se amplió el lenguaje con excepciones, identificación en tiempo de ejecución (*RTII, run_time type identification*), plantillas (templates) y la biblioteca estándar de plantillas STL (*Standard Template Library*). Stroustrup publicó en 1997 la tercera edición de su libro *The C++ Programming Language*. Este libro sigue el estándar ANSI/ISO C++.

C++ es un lenguaje estandarizado. En 1998, un comité de ANSI/ISO (*American Nacional Standard Institute/International Organization for Standardization*) adoptó una especificación del lenguaje conocida como Estándar C++ de 1998, oficialmente tiene el título ISO/ANSI C++ Standard (ISO/IEC 14882:1998) [Standard98]. En 2003 una segunda versión del Estándar vio la luz y fue adoptada también como [Standard03]. El estándar está disponible en Internet como archivo PDF con el número de documento 14882 en *http://www.ansi.org* donde se puede adquirir. La nueva edición es una revisión técnica, significando que se ordena la primera edición —fijación de tipos, reducción de ambigüedades y similares, pero no cambian las características del lenguaje. Este libro se basa en el estándar C++ como un superconjunto válido de C. Existen diferencias entre el estándar ANSI C y las reglas correspondientes de C++, pero son pocas. Realmente, ANSI C incorpora algunas características primitivas introducidas en C++, tal como el prototipado de funciones y el calificador de tipo `const`.

Antes de la emergencia de ANSI C, la comunidad de C fue seguida de un estándar de facto, basado en el libro *The C Programming Language*, de Kernighan y ritchie (Addison-Wesley Publishing Company, Reading, MA, 1978). Este estándar se conoció, en un principio, como K&r; con la emergencia de ANSI C, el k&R más simple se llamó, con frecuencia, C clásico.

El estándar ANSI C no sólo definió el lenguaje C sino que también definió una biblioteca estándar de C que deben soportar todas las implementaciones de ANSI C. C++ también utiliza la biblioteca; este libro se refiere a ella como la biblioteca estándar de C o simplemente *biblioteca estándar*. También el estándar ANSI/ISO C++ proporciona una biblioteca estándar de clases.

Recientemente, el C estándar ha sido revisado; el nuevo estándar, llamado, con frecuencia C99, fue adoptado por ISO en 1999 y por ANSI en 2000. Este estándar añade algunas características de C, tales como un nuevo tipo integer, que soportan algunos compiladores de C++. Aunque, no son parte del estándar C++ actual, estas características se pueden convertir como parte del C++ Standard. Antes de que el comité ANSI/ISO C++ comenzara su trabajo, muchas personas aceptaron como estándar, la versión de C++ más reciente de Bell Labs. Así, un compilador puede describirse como compatible con la versión 2.0 o 3.0 de C++.

Debido a que se lleva mucho tiempo hasta que un fabricante de compiladores implementa las últimas especificaciones de un lenguaje, existen compiladores que todavía no cumplen o conforman el estándar. Esto puede conducir a restricciones en algunos casos. Sin embargo, todos los programas de este libro han sido compilados con las últimas versiones de compiladores diferentes.

C++ es un lenguaje orientado a objetos que se ha hecho muy popular y con una gran difusión en el mundo del software y actualmente constituye un lenguaje estándar para programación orientada a objetos, aunque también puede ser utilizado como lenguaje estructurado al estilo de C si se desea trabajar al modo clásico de programación estructurada, sobre todo si se desea trabajar con algoritmos y estructura de datos.

1.11.1. C versus C++

C++ es una extensión de C con características más potentes. Estrictamente hablando, es un superconjunto de C. Al igual que sucede con Java y C# que son superconjuntos de C++. El ANSI C estándar no sólo define el lenguaje C sino que también define una biblioteca de C estándar que las implementaciones de ANSI C deben soportar. C++ también utiliza esa biblioteca, además de su propia biblioteca estándar de clases. Hace unos años se revisó el nuevo estándar de C, denominado C99 que fue adoptado por ISO en 1999 y por ANSI en 2000. El estándar de C ha añadido algunas características que soportan algunos compiladores de C++.

Por estas razones casi todas las sentencias de C también tienen una sentencia correcta en C++, pero no es cierto a la inversa. Los elementos más importantes añadidos a C para crear clases C, objetos y

programación orientada a objetos (C++ fue llamado originalmente "C con clases"). Sin embargo, a C++ se han añadido nuevas características, incluyendo un enfoque mejorado de la entrada/salida (E/S) y un nuevo medio para escribir comentarios. La Figura 1.26 muestra las relaciones entre C y C++

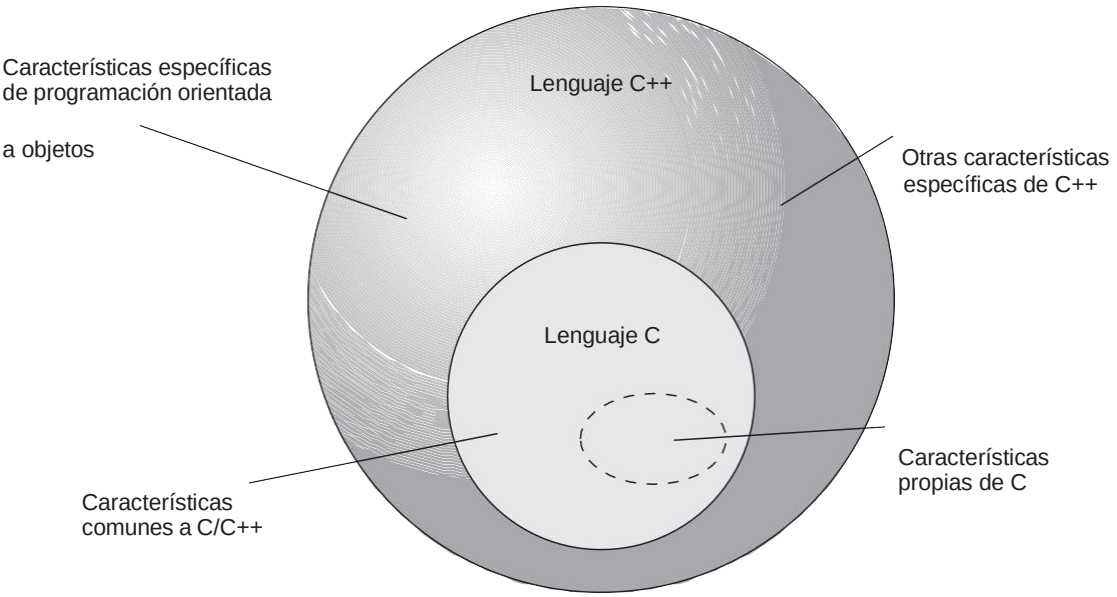


Figura 1.26. Características de C y C++.

De hecho las diferencias prácticas entre C y C+ son mucho mayores que lo que se pueda pensar. Aunque se puede escribir un programa en C++ similar a un programa en C, raramente se hace. C++ proporciona a los programadores las nuevas características de C++ y enfatiza a utilizar estas características, pero también se suele utilizar en partes de los programas las características específicas y potentes de C. Si usted ya conoce C, muchas propiedades le serán familiares y aprenderá de un modo mucho más rápido y eficiente, pero encontrará muchas propiedades nuevas y muy potentes que le harán disfrutar con este nuevo lenguaje y construir sus programas aprovechando la potencia de orientación a objetos y genérica de C++.

Nuestro objetivo es ayudarle a escribir programas POO tan pronto como sea posible, Sin embargo, como ya se ha observado, muchas características de C++ se han heredado de C, de modo que, aunque la estructura global de un programa pueda ser POO, consideramos que usted necesita conocimientos profundos del "viejo estilo *procedimental*". Por ello, los capítulos de la primera parte del libro le van introduciendo lenta y pausadamente en las potentes propiedades orientadas a objetos de las últimas partes, al objeto de conseguir a la terminación del libro el dominio de la Programación en C++.

Este libro describe el estándar ANSI/ISO C++, segunda edición, (ISO/IEC 14882:2003), de modo que los ejemplos funcionarán con cualquier implementación de C++ que sea compatible con el estándar. Sin embargo, pese a los años transcurridos, C++ Standard se suele considerar todavía nuevo y se pueden encontrar algunas discrepancias con el compilador que esté usted utilizando. Por ejemplo, si su compilador no es una versión reciente, puede carecer de espacios de nombres o de las características más novedosas de plantillas, como la clase `"String"` y la *Standard Template Library* que no son compatibles con los compiladores más antiguos. También puede suceder que si un compilador no cumple el estándar, algunas propiedades tales como el número de bytes usados para contener un entero, son dependientes de la implementación.

1.11.2 El futuro de C++

C++ continúa evolucionando y se habían realizado ya trabajos para producir la próxima versión del estándar. La nueva versión se ha llamado de modo informal como **C++0X**, ya que se espera su terminación al final de esta década, alrededor de 2009. Bjarne Stroustrup ha publicado el 2 de enero de 2006, un artículo titulado *The Sc++ Source: A Brief Look at C++0X* donde plantea brevemente los principios del trabajo del nuevo estándar y las esperanzas que existen para que el comité ISO C++ termine sus trabajos en 2008 y se puede convertir de facto en el nuevo estándar **C++09**. En otras palabras, comenta Struoustrup, se trata de reforzar el estándar **C++98** que ya es un lenguaje fuertemente implantado. Se pretende hacer de C++ un lenguaje mejor para programación de sistemas y construcción de bibliotecas, así como un lenguaje C++ que sea más fácil de enseñar y de aprender.

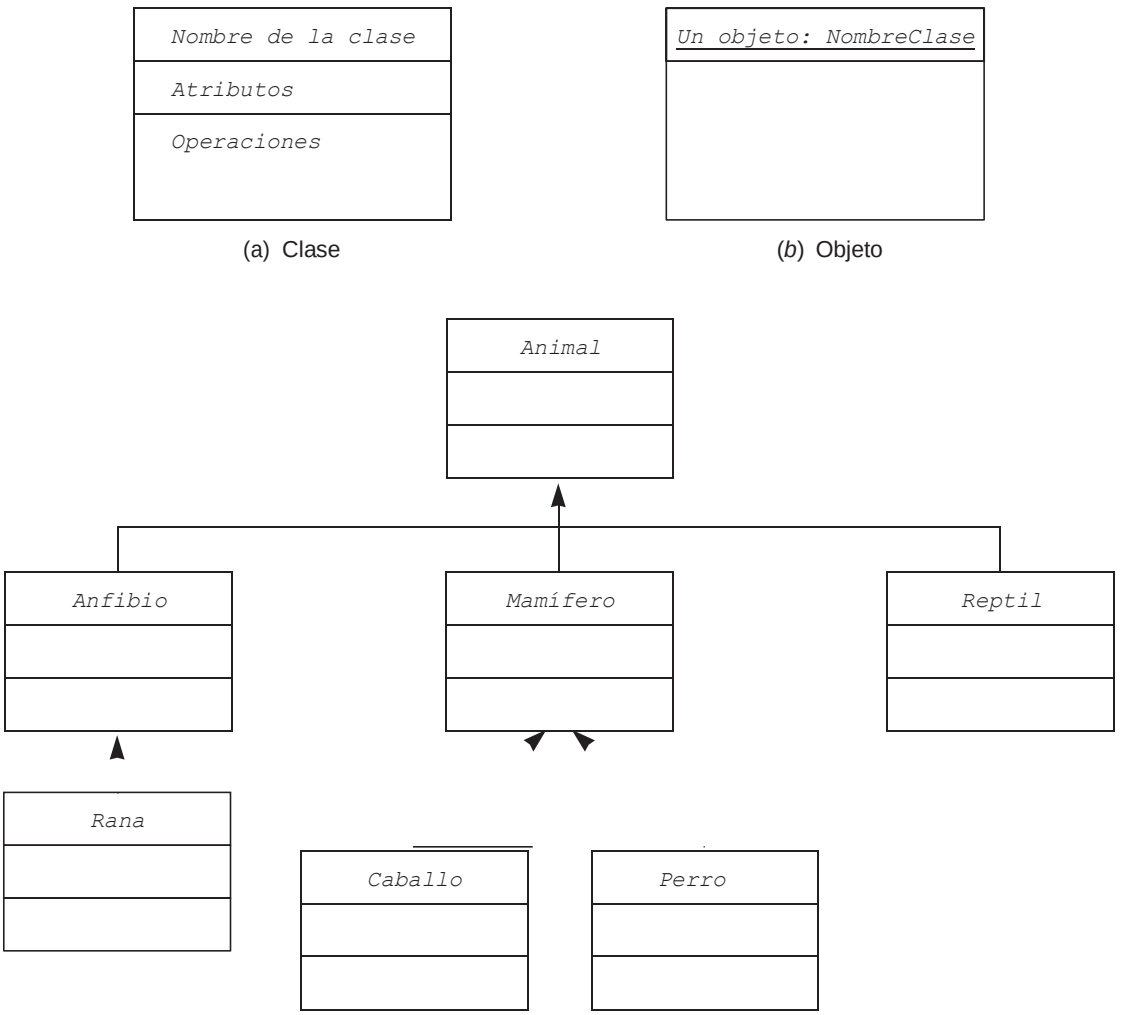
1.12. EL LENGUAJE UNIFICADO DE MODELADO (UML 2.0)

UML es un lenguaje gráfico para modelado de programas de computadoras. "Modelado" significa como su nombre indica, crear modelos o representaciones de "algo", como un plano de una casa o similar. UML proporciona un medio de visualizar la organización de alto nivel de los programas sin fijarse con detenimiento en los detalles del código real.

El lenguaje de modelado está unificado porque está basado en varios modelos previos (métodos de Booch, Rumbaugh y Jacobson). En la actualidad UML está adoptado por OMG (*Object Management Group*), un consorcio de más de 1.000 sociedades y universidades activas en el campo de tecnologías orientadas a objetos y está dedicado especialmente a unificación de estándares. UML tiene una gran riqueza semántica que lo abstrae de numerosos aspectos técnicos y ésa es su gran fortaleza. ¿Porqué necesitamos UML en programación orientada a objetos? En primer lugar, porqué UML nació como herramientas gráficas y metodologías para implementar análisis y diseño orientados a objetos y, en segundo lugar, porque en programas grandes de computadoras es difícil entender el funcionamiento de un programa examinando sólo su código y es necesario ver cómo se relacionan unas partes con otras.

La parte más importante de UML, al menos a un nivel de iniciación o medio en programación, reside en un conjunto rico de diagramas gráficos. Diagramas de clases que muestran las relaciones entre clases, diagramas de objetos que muestran cómo se relaciona objetos específicos entre sí, diagramas de casos de uso que muestran cómo los usuarios de un programa interactúan con el programa, etc. Cuando se modelan clases, UML es de hecho estándar para representaciones gráficas. UML no es un proceso de desarrollo de software sino, simplemente, un medio para examinar el software que se está desarrollando. Aunque, al ser un estándar, UML se puede aplicar a cualquier tipo de lenguaje de programación, está especialmente adaptado a la POO. En la Figura 1.27 se muestran representaciones gráficas de clases, objetos y relaciones de generalización y especialización (herencia) entre clases.

Una breve reseña de UML es la siguiente. En 1994, James Rumbaugh y Grady Booch deciden unirse para unificar sus notaciones y métodos: OMT y Booch. En 1995, Juar Jacobson se une al equipo de «los tres amigos» en la empresa Rational Software. En 1997 se publica UML 1.0 y en ese mismo año OMG adopta la notación y crea una comisión (*Tark Forcs*) encargada de estudiar la evolución de UML. En 2003 se presenta UML 1.5. A lo largo de 2005 se presenta UML 2.0 y en 2006 Rumbaugh, Booch y Jacobson publican las nuevas ediciones de la Guía de Usuario y Manual de Referencia.



(a) Clase (*b*) Objeto

(*c*) Herencia de claves

Figura 1.27. Representaciones gráficas de objetos, clases y herencia en UML 2.0.

RESUMEN

En este capítulo se realiza una breve descripción de la organización física de una computadora. Se describen conceptos fundamentales de la misma y sus bloques componentes más sobresalientes, tales como CPU ALU (UAL), VC, microprocesador, memorias RAM, ROM, dispositivos de E/S y de almacenamiento (cintas, disquetes, discos duros, CDs, DVDs, memorias *flash* con y sin conexiones USB, etc.).

Se hace también una breve introducción a la representación de la información en textos, imágenes, sonidos y datos numéricos. Asimismo, se analiza una primera introducción a la programación orientada a objetos y sus propiedades más importantes: abstracción, encapsulamiento de datos, polimorfismo y herencia (UCP), junto a las clases y objetos.

El capítulo termina con una breve historia y evolución de los lenguajes de programación C y C++.

REFERENCIAS BIBLIOGRÁFICAS Y LECTURAS RECOMENDADAS

BROOKSHEAR, J. Glenn. *Computer Science*, Eigth ed., Boston (USA): Pearson/Addison-Wesley (2005).

Uno de los mejores libros para aprender la ciencia de la computación a nivel de introducción y medio. Trata todos los temas relacionados con el mundo de la programación con una visión académica y científica notable. Desde el estudio de algoritmos hasta lenguajes de programación, ingeniería de software, bases de datos y una excelente introducción a la inteligencia artificial y teoría de la computa- ción.

British Standards Institute. *C++ estándar*. Madrid: Anaya. Prólogo de Bjarne Stroustrup inventor de C++ (2005).

Esta obra es el Estándar Internacional del lenguaje de programación C++. Esta versión traducida por la editorial Anaya del estándar original en inglés, recoge las especificaciones completas del C++ estándar, oficialmente conocido como BS ISO/IEC 14882:1998 con las actualizaciones (Corrigendum Técnico 1) adoptadas entre 1997 y 2001. Obra de referencia indispensable para el trabajo profesional en C++. Las 1.006 páginas de esta enciclopedia recogen las especificaciones oficiales. Es un libro de consulta para resolver cualquier duda de aplicación de sintaxis, especificaciones, bibliotecas de funciones, de clases, etc.

JOYANES AGUILAR, Luis. *Fundamentos de programación. Algoritmos, estructuras de datos y objetos*, 3.ª ed., Madrid: McGraw-Hill (2003).

Libro de referencia para el aprendizaje de la programación con un lenguaje algorítmico. Libro complementario de esta obra y que ha cumplido ya quince años desde la publicación de su primera edición

JOYANES AGUILAR, Luis. *Programación orientada a objetos*, 2.ª ed., Madrid: McGraw-Hill (2003).

Libro de referencia para el aprendizaje de la programación orientada a objetos y que se escribió en la segunda mitad de la década de los noventa cuando UML comenzaba a emerger. Ahora está en fase de revisión y mejora, esperando publicarse la tercera edición en el segundo semestre de 2006.

JOYANES, Luis y SÁNCHEZ, Lucas. *Programación en C++*. Colección Schaum. Madrid: McGraw-Hill, 2006.

Libro complementario de esta obra y con un carácter eminentemente teórico-práctico. Aunque se puede leer y estudiar de modo independiente, también se ha escrito pensando en completar de modo práctico toda la teoría necesaria para iniciarse en algoritmos y estructura de datos, así como en programación orientada a objetos. Contiene un gran número de ejemplos, ejercicios y problemas resueltos.

LAFORE, Robert. *Object-Oriented Programming in C++*, fourth ed., Indianapolis (Indiana): Sams (2002).

Uno de los mejores libros escritos sobre programación orientada a objetos y C++. Su autor reedita cada cierto tiempo el libro, lo cual permite que el lector esté actualizado en todo momento. Esta edición contiene 1.040 páginas y es uno de los libros más completos que se encuentran en el mercado.

PRATA, Stephen. *C++ Primer Plus*, fifth ed., Indianapolis (Indiana): Sams (2005).

Otro gran libro para aprender a programar C++ y métodos de orientación a objetos. Es también un libro enciclopédico (1.075 páginas) muy completo con profusión de ejemplos y ejercicios complejos resueltos.

PRIETO ESPINOSA, Alberto y PRIETO CAMPOS, Beatriz. *Conceptos de Informática*. Colección Schaum. Madrid: McGraw-Hill (2005).

Excelente libro teórico-práctico para conocer todos los fundamentos de la informática y de gran utilidad para el aprendizaje y dominio de la programación.

PRIETO, A., LLORIS, A. y TORRES, *Introducción a la informática*, 3.ª ed., Madrid: McGraw-Hill (2005).

Uno de los mejores libros teóricos para conocer todos los fundamentos de informática necesarios para iniciarse en carreras de ingeniería y ciencias. Lectura imprescindible y consulta obligada, para profundizar en técnicas de programación tanto en el campo del *hardware* como del *software*.