Introducción a la ciencia de la computación

de La manipulación de datos
a La teoría de La computación

Diseñado para el curso introductorio a la ciencia de la computación, este libro tiene como objetivo cubrir lo que todo estudiante de ingeniería e informática necesita saber antes de pasar a cursos más especializados. El autor desarrolla el texto en cinco partes: Computadoras y datos, Hardware, Software, Organización de datos y Temas avanzados (compresión de datos, seguridad y teoría de la computación), a fin de abordar, uno a uno, los principales temas básicos. Además de lo anterior, esta obra presenta las siguientes características:

- Desde un principio estudia la representación de datos, esto incluye texto, imágenes, audio, video y números.
- Hace hincapié en los conceptos y no en los modelos matemáticos.
- Las redes y los sistemas operativos se tratan en capítulos aparte para incluir las tendencias actuales.
- Cuenta con ayudas visuales en los temas más difíciles.

THOMSON

MÉXICO Y AMÉRICA CENTRAL
Tel. 52(55)5281-2906
Fax52(55)5280-8970
editor@thomsonlearning.com.mx
México, D.F., MÉXICO

AMÉRICA DEL SUR
thomson@thomsonlearning.com.ar
Buenos Aires, ARGENTINA

EL CARIBE
Tel. (787) 641-1112
Fax(787) 641-1119
thomson@coqui.net
San Juan, PUERTO RICO

PACTO ANDINO
Tel.(571)340-9470
Fax.(571)340-9475
clithomson@andinet.com
Bogotá, COLOMBIA

ESPAÑA
Tel. (3491)446-3350
Fax (3491)445-6218
clientes@paraninfo.es
Madrid, España

ISBN 970-686-285-4

Introducción a la ciencia de la computación

de La manipulación de datos
a La teoría de La computación

Behrouz A. Forouzan

CIENCIAS E INGENIERÍAS

Introducción a la ciencia de la computación

De la manipulación de datos a la teoría de la computación

Introducción a la ciencia de la computación

De la manipulación de datos a la teoría de la computación

Behrouz A. Forouzan
De Anza College

con colaboración de
Sophia Chung Fegan

THOMSON

Australia • Brasil • Canadá • España • Estados Unidos • México • Reino Unido • Singapur

THOMSON

Introducción a la ciencia de la computación
De la manipulación de datos a la teoría de la computación
Behorouz A. Forouzan

Vicepresidente editorial y de producción:
Miguel Ángel Toledo Castellanos

Editor de desarrollo:
Pedro de la Garza Rosales

Traducción:
Lorena Peralta

Gerente de producción:
René Garay Argueta

Editora de producción:
Alma Castrejón Alcocer

Supervisora de manufactura:
Claudia Calderón Valderrama

Revisión técnica:
Jorge Valeriano Assem
UNAM-Facultad de ingeniería

Traducido del libro *Foundations of Computer Science, From Data Manipulation to Theory of Computation* publicado en inglés por Brooks Cole, ©2003
ISBN 0-534-37968-0
Datos para catalogación bibliográfica:
Forouzan, Behrouz A. *Introducción a la ciencia de la computación.*
ISBN 970-686-285-4
1. Introducción a la ciencia de la computación. 2. De la manipulación de datos a la teoría de la computación.

Impreso en México
Printed in Mexico
1 2 3 4 05 04 03

Para mayor información contáctenos en: Séneca 53
Col. Polanco
México, D. F. 11560

Puede visitar nuestro sitio en
http://www.thomsonlearning.com.mx

División Iberoamericana

México y América Central:
Thomson Learning
Séneca 53
Col. Polanco
México, D.F. 11560
Tel 52 (55) 52 81 29 06
Fax 52 (55) 52 81 26 56
editor@thomsonlearning.com.mx

América del Sur:
Thomson Learning
Calle 39 No. 24-09
La Soledad
Bogotá, Colombia
Tel (571) 340 94 70
Fax (571) 340 94 75
clithomson@andinet.com

El Caribe:
Thomson Learning
598 Aldebarán
Altamira San Juan,
Puerto Rico
Tel (787) 641 11 12
Fax (787) 641 11 19
thomson@coqui.net

España:
Thomson Learning
Calle Magallanes núm. 25
28015 Madrid, España
Tel 34 (0)91 446 33 50
Fax 34 (0)91 445 62 18
clientes@paraninfo.es

Cono Sur:
Tel/Fax (54 114) 582 26 84
Buenos Aires, Argentina
sdeluque@thomsonlearning.com.ar

Esta obra se terminó de imprimir en el 2003
en los talleres de Grupo GEO Impresores S.A. de C.V.

A mi esposa Faezeh

— Behrouz Forouzan

Contenido

Prefacio

¡Bienvenidos a las ciencias de la computación! Están a punto de iniciar la exploración de un maravilloso y excitante mundo que ofrece muchas carreras desafiantes y emocionantes. Las computadoras juegan un papel importante en nuestra vida diaria y continuarán haciéndolo en el futuro.

Las ciencias de la computación son una disciplina joven que está evolucionando y progresando. Las redes de computadoras han conectado a personas desde lugares remotos del mundo. La realidad virtual está creando imágenes tridimensionales que sorprenden a la vista. La exploración espacial debe parte de su éxito a las computadoras. Los efectos especiales creados por computadora han cambiado la industria del cine. Y las computadoras han jugado papeles importantes en la genética.

ORGANIZACIÓN DEL LIBRO

Este libro está diseñado para un curso CS0 (primer curso de ciencias de la computación). Cubre todas las áreas de la computación. Dividimos el texto en cinco partes: Computadoras y datos, Hardware de computadoras, Software de computadoras, Organización de los datos y Temas avanzados (figura P.1).

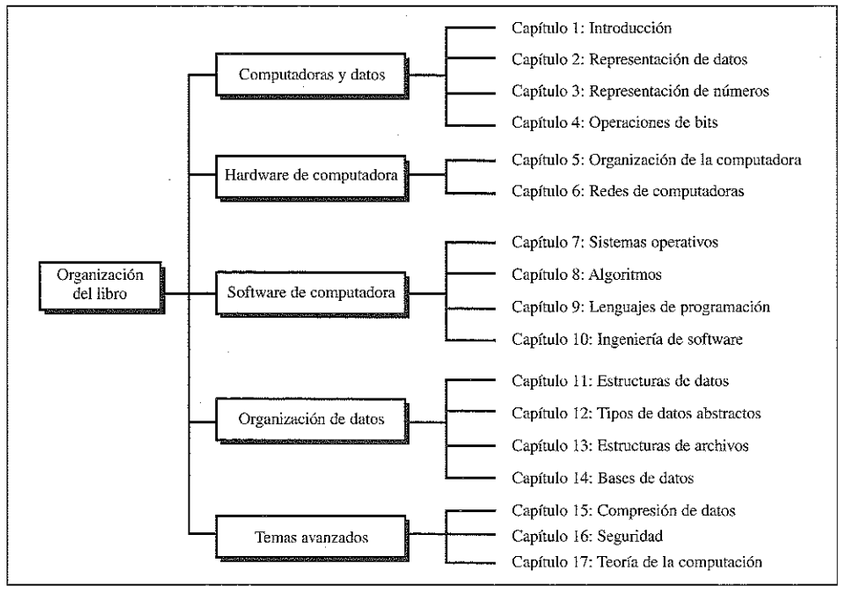


Figura P.1 Esquema del libro

Parte I: Computadoras y datos

En la parte I estudiamos una computadora y los datos que ésta procesa. Esta parte contiene cuatro capítulos.

Capítulo 1: Introducción En este capítulo se considera a la computadora como una entidad procesadora de datos. Presentamos el concepto de von Neumann y analizamos los componentes generales de una computadora. Posponemos el análisis detallado de los componentes de computadoras hasta el capítulo 5.

Capítulo 2: Representación de datos En este capítulo, estudiamos la representación de texto, imágenes, audio y video como patrones de bits. La representación de números se pospone hasta el capítulo 3.

Capítulo 3: Representación de números En este capítulo presentamos la representación de números. Mostramos cómo los enteros y los números de punto flotante se almacenan en una computadora.

Capítulo 4: Operaciones de bits En este capítulo, estudiamos la manipulación de patrones de bits, tanto aritméticos como lógicos.

Parte II: Hardware de computadora

En la parte II, estudiamos el hardware de computadora. Esta parte se divide en dos capítulos.

Capítulo 5: Organización de la computadora En este capítulo, consideramos a la computadora como una máquina independiente. Describimos las partes del hardware de computadora y cómo funcionan.

Capítulo 6: Redes de computadoras En este capítulo comentamos cómo se conectan las computadoras para crear redes e interredes.

Parte III: Software de computadora

En la parte III, estudiamos varios aspectos del software de computadora.

Capítulo 7: Sistemas operativos Este capítulo estudia al sistema operativo como la parte más importante del software del sistema. Presentamos las tareas de un sistema operativo y cómo evolucionan. También analizamos las partes de un sistema operativo moderno.

Capítulo 8: Algoritmos En este capítulo, estudiamos los algoritmos, los cuales juegan un papel tan importante en las ciencias de la computación que algunas personas creen que ciencias de la computación significa el estudio de los algoritmos. Definimos el concepto de algoritmos y usamos algunas herramientas para representar algoritmos. Una discusión completa de estas herramientas se presenta en los apéndices C, D y E.

Capítulo 9: Lenguajes de programación Este capítulo presenta primero los lenguajes en general y luego analiza los elementos de C, un lenguaje generalizado.

Capítulo 10: Ingeniería de software En este capítulo se estudia la ingeniería de software, una disciplina muy importante para el estudio de las computadoras.

Parte IV: Organización de datos

En la parte IV, estudiamos nuevamente los datos, pero desde el punto de vista del usuario.

Capítulo 11: Estructuras de datos En este capítulo comentamos las estructuras de datos. Los datos, en el nivel más alto, están organizados en estructuras. Presentamos estructuras de datos comunes en uso en la actualidad tales como arreglos, registros y listas ligadas.

Capítulo 12: Tipos de datos abstractos En este capítulo analizamos los tipos de datos abstractos (ADT). En el procesamiento de datos, se necesita definir a los datos como un paquete incluyendo las operaciones definidas para el paquete. Se describen listas, pilas, colas, árboles y gráficas de tal manera que los estudiantes piensen en los datos en abstracto.

Capítulo 13: Estructuras de archivos En este capítulo analizamos las estructuras de archivos. Mostramos cómo los archivos se organizan lógicamente y analizamos los archivos de acceso secuencial y de acceso aleatorio. Un estudiante necesita conocer estos conceptos antes de tomar un primer curso en programación.

Capítulo 14: Bases de datos Este capítulo analiza las bases de datos. Los archivos en una organización rara vez se almacenan por separado y de manera aislada. Con frecuencia, éstos se organizan en una entidad llamada base de datos. Presentamos la base de datos relacional y mencionamos un lenguaje (SQL) que puede recuperar información de este tipo de base de datos.

Parte V: Temas avanzados

En esta parte analizamos tres temas avanzados que están ganando importancia en las ciencias de la computación, compresión de datos, seguridad y la teoría de la computación. Estos temas pueden omitirse si el tiempo es un factor importante o si los estudiantes no tienen una formación previa.

Capítulo 15: Compresión de datos En este capítulo presentamos dos categorías de compresión de datos: sin pérdida y con pérdida. Analizamos la codificación de longitud de ejecución, la codificación Huffman y el algoritmo de Lempel Ziv como ejemplos de compresión sin pérdida. Analizamos JPEG y MPEG como ejemplos de compresión con pérdida.

Capítulo 16: Seguridad Este capítulo estudia cuatro aspectos de la seguridad: privacidad, autenticación, integridad y no rechazo. Mostramos cómo utilizar el cifrado/descifrado y la firma digital para crear un sistema seguro.

Capítulo 17: Teoría de la computación En este capítulo exploramos brevemente la teoría de la computación. Mostramos cómo ningún lenguaje es superior a otro en la solución de un problema. Explicamos que hay algunos problemas que no pueden resolverse por ningún programa de computadora escrito en cualquier lenguaje.

MIRADA A VUELO DE PÁJARO

El lector debe tener en mente que este libro no estudia ningún tema sobre las ciencias de la computación a profundidad; para hacerlo se requerirían múltiples volúmenes. El libro intenta cubrir temas relacionados con las ciencias de la computación. Nuestra experiencia muestra que conocer la representación y la manipulación de datos, por ejemplo, ayuda a los estudiantes a comprender mejor la programación en lenguajes de bajo y alto nivel. Conocer información general sobre las ciencias de la computación ayudará a los estudiantes a tener más éxito cuando tomen cursos de conectividad en red e interredes. El libro es una mirada a vuelo de pájaro de las ciencias de la computación.

CARACTERÍSTICAS DEL LIBRO

Hay varias características de este libro que no sólo lo hacen único, sino más comprensible para los estudiantes primerizos.

Conceptos

A través del libro, hemos intentado enfatizar más en el concepto que en el modelo matemático. Creemos que una comprensión del concepto conduce a una comprensión del modelo.

Método visual

Una breve revisión del libro mostrará que nuestro método es muy visual. Hay casi 300 figuras. Aun cuando esto tiende a aumentar el tamaño del libro, las figuras ayudan a la comprensión del texto.

Ejemplos

Siempre que se considera apropiado, se usan ejemplos para demostrar el concepto y el modelo matemático.

Material de fin de capítulo

El material al final de cada capítulo contiene tres partes: términos clave, resumen y práctica.

Términos clave Los términos clave proporcionan una lista de los términos importantes presentados en el capítulo. Cada término clave se define en el glosario.

Resumen Los resúmenes contienen una descripción general concisa de todos los puntos clave del capítulo. Se listaron con viñetas para que sean más fáciles de leer.

Práctica Cada práctica se compone de tres partes: preguntas de repaso, preguntas de opción múltiple y ejercicios.

- Las **preguntas de repaso** evalúan los puntos clave y conceptos generales del capítulo.
- Las **preguntas de opción múltiple** están diseñadas para probar la comprensión de los materiales.
- Los **ejercicios** están diseñados para ver si los estudiantes pueden aplicar los conceptos y las fórmulas.

Apéndices

Se incluyen siete apéndices para una referencia rápida a las tablas o materiales que se estudian en varios capítulos. Los apéndices son:

- Tabla ASCII
- Unicode
- Diagramas de flujo
- Pseudocódigo
- Diagramas de estructura
- Transformación de coseno discreto
- Acrónimos

Glosario

Al final del libro se incluye un glosario con todos los términos clave.

Soluciones de las prácticas

La solución de todas las preguntas de repaso, preguntas de opción múltiples y ejercicios con un número non están disponibles en línea en www.brookscole.com/compsci.

MATERIALES DE INSTRUCCIÓN

La presentación de PowerPoint de todas las figuras y puntos resaltados, además de la solución a todas las preguntas de repaso, preguntas de opción múltiple y ejercicios está disponible en línea en www.brookscole.com/compsci.

RECONOCIMIENTOS

Ningún libro de este campo puede desarrollarse sin el apoyo de muchas personas. Esto es especialmente cierto para este texto.

Nos gustaría agradecer el apoyo del equipo de De Anza por su continuo ánimo y sus comentarios. En particular agradecemos la contribución de Scout DeMouthe por leer el manuscrito y resolver los problemas.

Para cualquiera que no ha pasado por este proceso, el valor de las revisiones de los compañeros y compañeras no puede apreciarse lo suficiente. Escribir un texto rápidamente se vuelve un proceso miope. La orientación importante de los revisores que pueden guardar distancia para revisar el texto como un todo no puede medirse. Para parafrasear un viejo cliché, "No son valiosos, son invaluables". Nos gustaría agradecer especialmente las contribuciones de los siguientes revisores: Essam El-Kwae, Universidad de Carolina del Norte en Charlotte; Norman J. Landis, Universidad Fairleigh Dickinson; John A. Rohr, Universidad de California en Los Ángeles Robert Signorile, Boston Collage, y Robert Statica del Instituto de Tecnología de New Jersey.

Damos las gracias también a nuestro editor, Hill Stenquist; a la editora de adquisiciones, Kallie Swanson; a la asistente editorial, Carla Vera, y al editor de producción, Kesley McGee. También deseamos agradecer a Merrill Peterson de Matriz Productions, al corrector Frank Hubert y a la lectora de pruebas Amy Dorr.

Por último, y obviamente no lo menos importante, está el agradecimiento al apoyo recibido por parte de nuestras familias y amigos. Hace muchos años un autor definió el proceso de escribir un texto como "encerrarse en un cuarto". Mientras los autores sufren a través del proceso de escritura, sus familias y amigos sufren a través de su ausencia. Sólo podemos esperar que en cuanto vean el producto final, sientan que su sacrificio valió la pena.

I

Datos y computadoras

1 Introducción

La frase **ciencia de la computación** tiene un significado muy amplio en la actualidad. Sin embargo, en este libro la definimos como "temas relacionados con la computación". Este capítulo introductorio primero trata de averiguar qué es la computadora y luego descubre otros aspectos relacionados con las computadoras. Consideramos a la computadora como una **caja negra** y tratamos de adivinar su comportamiento. Luego tratamos de penetrar en esta caja para dejar al descubierto lo que es común a todas las computadoras. Esto nos lleva al **modelo de von Neumann**, el cual es universalmente aceptado como la base de la computadora. Enseguida analizamos brevemente las repercusiones y los inconvenientes de aceptar el modelo de Von Neumann. En este punto, nos referimos al capítulo o los capítulos del texto relacionados con estos problemas. El capítulo termina con una breve historia de este recurso cambiante de la cultura: la computadora.

1.1 LA COMPUTADORA COMO UNA CAJA NEGRA

Si usted no está interesado en los mecanismos internos de una computadora, simplemente puede definirla como una caja negra. Sin embargo, aun así necesita definir el trabajo realizado por una computadora para distinguirla de otros tipos de cajas negras. Explicaremos dos modelos comunes de computadoras.

PROCESADOR DE DATOS

Puede pensar en una computadora como un **procesador de datos**. Usando esta definición, una computadora actúa como una caja negra que acepta datos de entrada, procesa los datos y crea datos de salida (figura 1.1). Aunque este modelo puede definir la funcionalidad de una computadora hoy día, es demasiado general. Bajo este modelo, una calculadora de bolsillo también es una computadora (lo cual es cierto, literalmente).

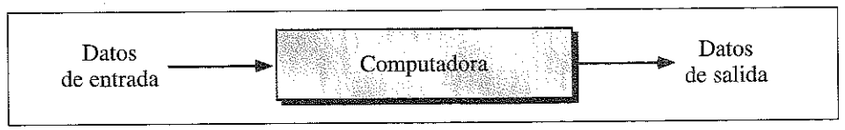


Figura 1.1 Modelo de procesador de datos

Otro problema con este modelo es que no especifica el tipo de procesamiento o si es posible más de un tipo de procesamiento. En otras palabras, no queda claro cuántos tipos de conjuntos de operaciones puede realizar una máquina basada en este modelo. ¿Se trata de una máquina para propósito específico o una máquina para propósito general?

Este modelo podría representar una computadora para propósito específico (o procesador) que está diseñada para realizar alguna tarea en especial como controlar la temperatura de un edificio o controlar el uso de combustible en un automóvil. Sin embargo, las computadoras, según se usa el término en la actualidad, son máquinas de *propósito general*. Pueden realizar muchos tipos distintos de tareas. Esto implica que necesitamos cambiar nuestro modelo para reflejar las computadoras reales de hoy.

PROCESADOR DE DATOS PROGRAMABLE

Un mejor modelo para una computadora de propósito general se muestra en la figura 1.2. Esta figura añade un elemento extra a la computadora: el programa. Un **programa** es un conjunto de instrucciones que indican a la computadora qué hacer con los datos. En los primeros días de las computadoras, las instrucciones se ejecutaban al cambiar el cableado o apagar y encender una serie de interruptores. Actualmente, un programa es una serie de instrucciones escritas en un **lenguaje de computadora**.

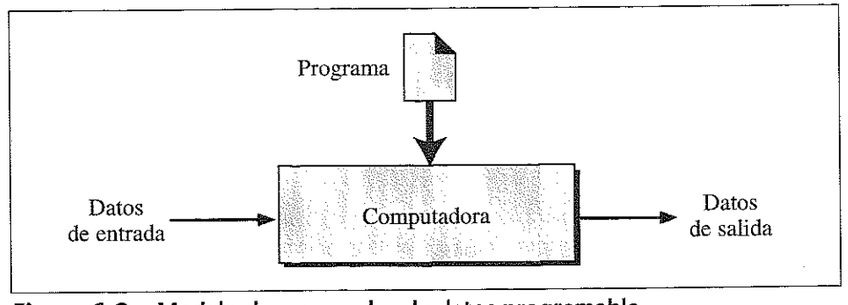


Figura 1.2 Modelo de procesador de datos programable

En el nuevo modelo, los **datos de salida** dependen de la combinación de dos factores: los **datos de entrada** y el programa. Con los mismos datos de entrada, usted puede generar distintas salidas si cambia el programa. De manera similar, con el mismo programa, puede generar diferentes salidas si cambia la entrada. Finalmente, si los datos de entrada y el programa permanecen igual, la salida deberá ser la misma. Veamos tres casos.

Mismo programa, diferentes datos de entrada

La figura 1.3 muestra el mismo programa de ordenamiento con datos distintos. Aunque el programa es el mismo, la salida es diferente porque se procesan distintos datos de entrada.

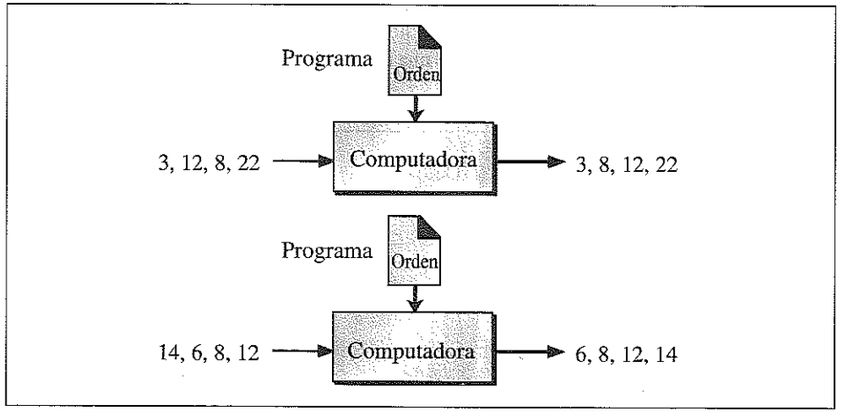


Figura 1.3 Mismo programa, diferentes datos

Mismos datos de entrada, diferentes programas

La figura 1.4 muestra los mismos datos de entrada con diferentes programas. Cada programa hace que la computadora realice distintas operaciones con los mismos datos de entrada. El primer programa ordena los datos, el segundo suma los datos y el tercero encuentra el número más pequeño.

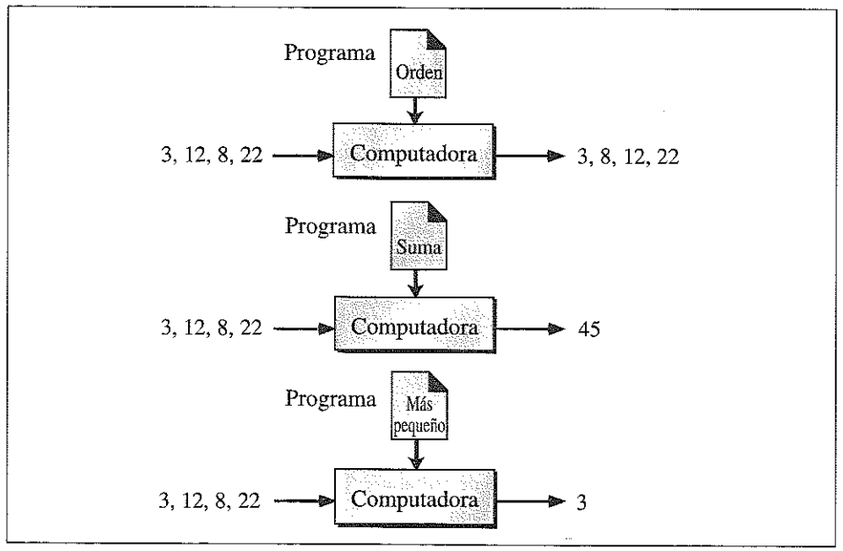


Figura 1.4 Mismos datos, diferentes programas

Mismos datos de entrada, mismo programa

Desde luego, usted espera el mismo resultado cada vez que tanto los datos de entrada como el programa son los mismos. En otras palabras, cuando el mismo programa se ejecuta con la misma entrada, usted espera la misma salida.

1.2 EL MODELO DE VON NEUMANN

En la actualidad cada computadora se basa en el modelo de Von Neumann (que lleva el nombre de John von Neumann). El modelo examina el interior de la computadora (la caja negra) y define cómo se realiza el procesamiento. Se basa en tres ideas.

CUATRO SUBSISTEMAS

El modelo define una computadora como cuatro subsistemas: memoria, unidad lógica aritmética, unidad de control y entrada/salida (figura 1.5).

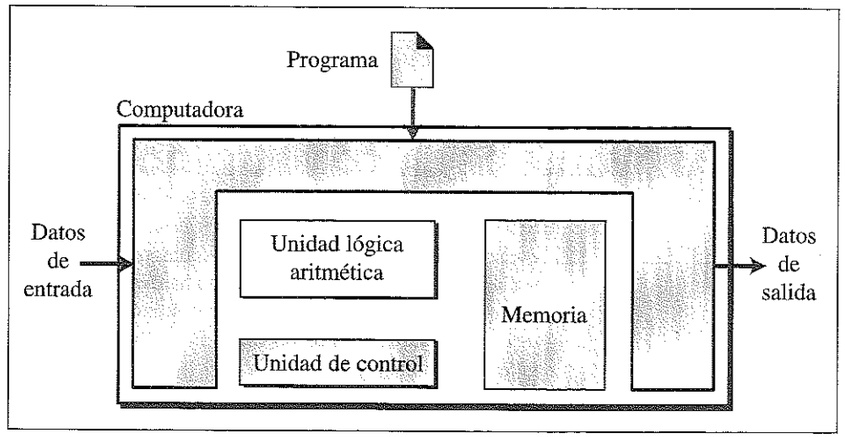


Figura 1.5 Modelo de von Neumann

Memoria

La **memoria** es el área de almacenamiento, donde los programas y los datos se almacenan durante el procesamiento. Más adelante en este capítulo se analiza la razón de ser del almacenamiento de programas y datos.

Unidad lógica aritmética

La **unidad lógica aritmética** (**ALU**: *arithmetic logic unit*) es donde el cálculo aritmético y las operaciones lógicas toman lugar. Si una computadora es un procesador de datos, usted debería poder realizar operaciones aritméticas con los datos (por ejemplo, sumar una lista de números). También debería poder realizar operaciones lógicas con ellos (por ejemplo, encontrar el menor de dos elementos de datos, como en el ejemplo de clasificación previo).

Unidad de control

La **unidad de control** determina las operaciones de la memoria, de la ALU y del subsistema de entrada/salida.

Entrada/Salida

El subsistema de entrada acepta datos de entrada y el programa desde el exterior de la computadora; el subsistema de salida envía el resultado del procesamiento al exterior. La definición del subsistema de entrada/salida es muy amplia; también incluye los dispositivos de almacenamiento secundarios como un disco o cinta que almacena datos y programas para

procesamiento. Un disco se considera un dispositivo de salida cuando almacena los datos que se obtienen como resultado del procesamiento y se considera un dispositivo de entrada cuando usted lee datos del disco.

CONCEPTO DE PROGRAMA ALMACENADO

El modelo de von Neumann establece que el programa debe almacenarse en la memoria. Esto es totalmente diferente de la arquitectura de las primeras computadoras en las cuales sólo se almacenaban los datos en la memoria. El programa para una tarea se implementaba mediante la manipulación de una serie de interruptores o al cambiar el sistema de cableado.

La memoria de las computadoras modernas aloja tanto un programa como sus datos correspondientes. Esto implica que ambos, tanto los datos como el programa, deben tener el mismo formato porque se almacenan en la memoria. De hecho, se guardan como patrones binarios (una secuencia de ceros y unos) en la memoria.

EJECUCIÓN SECUENCIAL DE INSTRUCCIONES

Un programa en el modelo de von Neumann se conforma de un número finito de **instrucciones**. En este modelo, la unidad de control trae una instrucción de la memoria, la interpreta y luego la ejecuta. En otras palabras, las instrucciones se ejecutan una después de otra. Desde luego, una instrucción puede requerir que la unidad de control salte a algunas instrucciones previas o posteriores, pero esto no significa que las instrucciones no se ejecutan de manera secuencial.

1.3 HARDWARE DE LA COMPUTADORA

Sin lugar a dudas, el modelo de von Neumann establece el estándar de los componentes esenciales de una computadora. Una computadora física debe incluir los cuatro componentes, a los que se hace referencia como hardware de la computadora, definidos por von Neumann. Pero usted puede tener diferentes tipos de memoria, diferentes tipos de subsistemas de entrada/salida, y así por el estilo. El hardware de la computadora se analiza con más detalle en el capítulo 5.

1.4 DATOS

Este modelo define claramente a una computadora como una máquina de procesamiento de datos que acepta datos de entrada, los procesa y produce el resultado.

ALMACENAMIENTO DE DATOS

El modelo de von Neumann no define cómo deben almacenarse los datos en una computadora. Si una computadora es un dispositivo electrónico, la mejor manera de almacenar los datos es en forma de señal eléctrica, específicamente su presencia o ausencia. Esto implica que una computadora puede almacenar datos en uno de dos estados.

Evidentemente, los datos que usted usa en la vida diaria no están sólo en uno de dos estados. Por ejemplo, nuestro sistema de numeración emplea dígitos que pueden estar en uno de diez estados (0 a 9). Este tipo de información no se puede (todavía) almacenar en una computadora. Necesita ser cambiada a otro sistema que use sólo dos estados (0 y 1).

Usted necesita además procesar otros tipos de datos (texto, imágenes, audio, video). Éstos tampoco pueden almacenarse en una computadora directamente, sino que deben cambiarse a la forma apropiada (ceros y unos).

En los capítulos 2 y 3, aprenderá cómo almacenar distintos tipos de datos como un patrón binario, una secuencia de ceros y unos. En el capítulo 4, mostraremos la manera en que se manipulan los datos como un patrón binario, dentro de una computadora.

ORGANIZACIÓN DE DATOS

Aun cuando los datos deben almacenarse sólo en una forma (un patrón binario) dentro de una computadora, los datos fuera de una computadora pueden tomar muchas formas. Además, las computadoras (y la noción del procesamiento de datos) han creado un nuevo campo de estudio conocido como **organización de datos**. ¿Puede usted organizar sus datos en diferentes entidades y formatos antes de almacenarlos dentro de una computadora? Hoy día, los datos no se tratan como una secuencia de información, sino que se organizan en unidades pequeñas, las cuales a su vez están organizadas en unidades más grandes, y así sucesivamente. En los capítulos 2 a 4, estudiaremos los datos desde este punto de vista.

1.5 SOFTWARE DE LA COMPUTADORA

La característica principal del modelo de von Neumann es el concepto del programa almacenado. Aunque las primeras computadoras no usaron este modelo, sí usaron el concepto de programas. La *programación* de estas primeras computadoras implicaba cambiar los sistemas de cableado o encender y apagar una serie de interruptores. La programación era una tarea realizada por un operador o ingeniero antes de que comenzara el procesamiento de los datos.

El modelo de von Neumann cambió el significado del término **"programación"**. En este modelo, hay dos aspectos de la programación que deben ser comprendidos.

LOS PROGRAMAS DEBEN ALMACENARSE

En el modelo de von Neumann, los programas se almacenan en la memoria de la computadora. No sólo se necesita memoria para mantener los datos, sino que además se requiere memoria para mantener el programa (figura 1.6).

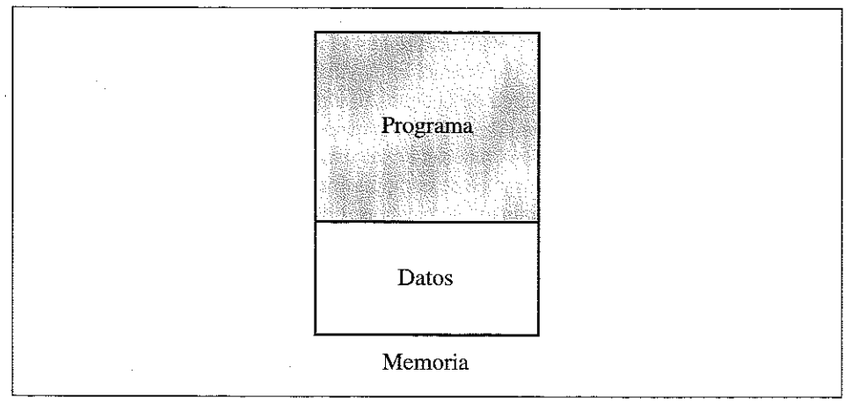


Figura 1.6 Programa y datos en memoria

UNA SECUENCIA DE INSTRUCCIONES

Otro requisito del modelo es que el programa debe ser una secuencia de instrucciones. Cada instrucción opera en una o más piezas de datos. De esta manera cada instrucción puede cambiar el efecto de una instrucción previa. Por ejemplo, la figura 1.7 muestra un programa que introduce dos números, luego los suma e imprime el resultado. Este programa consiste de cuatro instrucciones individuales.

Una persona podría preguntar por qué un programa debe estar hecho de instrucciones. La respuesta es por su capacidad para usarse de nuevo. Actualmente, las computadoras realizan millones de tareas. Si el programa para cada tarea fuera una entidad independiente sin una sección común con otros programas, la programación sería difícil. El modelo de von Neumann facilita la programación mediante una definición cuidadosa de diferentes instruccio-

1. Introduzca la primera pieza de datos en la memoria.
2. Introduzca la segunda pieza de datos en la memoria.
3. Sume las dos piezas y almacene el resultado en la memoria.
4. Imprima* el resultado.

Programa

Figura 1.7 Programa hecho de instrucciones

nes que las computadoras pueden usar. Un programador combina estas instrucciones para hacer cualquier número de programas. Cada programa puede ser una combinación diferente de distintas instrucciones.

ALGORITMOS

El requisito anterior hace posible la programación, pero aporta otra dimensión al uso de la computadora. Un programador no sólo debe aprender la tarea que realiza cada instrucción, sino que además debe saber cómo combinar estas instrucciones para realizar una tarea en particular. Al considerar este problema desde un punto de vista diferente, un programador primero debe resolver el problema paso a paso y luego tratar de encontrar la instrucción (o la serie de instrucciones) apropiada que resuelva el problema. La solución paso a paso se conoce como **algoritmo**. Los algoritmos juegan un papel muy importante en la ciencia de la computación, y se estudian en el capítulo 8.

LENGUAJES

Al principio de la era de las computadoras, no había un lenguaje de computadora. Los programadores escribían instrucciones (usando patrones binarios) para resolver un problema. Sin embargo, a medida que los programas crecieron, escribir programas largos usando estos patrones se volvió tedioso. A los científicos de la computación se les ocurrió la idea de usar símbolos para representar patrones binarios, justo como la gente usa símbolos (palabras) para dar órdenes en la vida diaria. Pero desde luego, los símbolos usados en la vida diaria difieren de aquellos usados en las computadoras. De esta manera nació el concepto de los lenguajes de computadora. Un idioma natural (por ejemplo el inglés) es un lenguaje rico y tiene muchas reglas para combinar las palabras de una manera correcta; por otra parte, un lenguaje de computación tiene un número más limitado de símbolos y también un número limitado de palabras. Los lenguajes de computadora se cubren en el capítulo 9.

INGENIERÍA DE SOFTWARE

Algo que no se definió en el modelo de von Neumann es la **ingeniería de software**, la cual es el diseño y la escritura de programas estructurados. En la actualidad no es aceptable sólo escribir un programa que realiza una tarea; el programa debe seguir principios y reglas estrictos. Estos principios, conocidos en conjunto como ingeniería de software, se estudiarán en el capítulo 10.

SISTEMAS OPERATIVOS

Durante la evolución de las computadoras, los científicos se dieron cuenta de que existía una serie de instrucciones comunes a todos los programas. Por ejemplo, casi todos los programas requieren instrucciones para indicar a una computadora a dónde enviar los datos y dónde recibirlos. Es más eficiente escribir estas instrucciones sólo una vez de manera que las usen todos los programas. Así fue como surgió el concepto de **sistema operativo**. Originalmente un sistema operativo trabajaba como un administrador para facilitarle a un programa el acceso a los componentes de la computadora. Hoy día, los sistemas operativos hacen mucho más. Aprenderá sobre ellos en el capítulo 7.

*N. del T. Bajo este contexto, *imprimir* significa enviar el resultado del procesamiento de los datos a una impresora, un archivo, la pantalla, un modem, un dispositivo de almacenamiento o cualquier otro dispositivo de salida.

1.6 HISTORIA

Antes de cerrar este capítulo, daremos un breve repaso a la historia de la computación y las computadoras, para lo cual dividimos esta historia en tres periodos.

MÁQUINAS MECÁNICAS (ANTES DE 1930)

Durante este periodo, se inventaron varias máquinas computadoras que tienen cierto parecido con el concepto moderno de computadora.

- En el siglo XVII, Blaise Pascal, un matemático y filósofo francés, inventó la Pascalina, una calculadora mecánica para operaciones de suma y resta. En el siglo XX, cuando Niklaus Wirth inventó un lenguaje de programación estructurado, lo llamó Pascal en honor al inventor de la primera calculadora mecánica.
- A finales del siglo XVII, el matemático alemán Gottfried Leibnitz inventó una calculadora mecánica más compleja que podía realizar operaciones de multiplicación y división, así como de suma y resta. Se le llamó la Rueda de Leibnitz.
- La primera máquina que usó la idea de almacenamiento y programación fue el telar de Jacquard, inventado por Joseph-Marie Jacquard a principios del siglo XIX. El telar usaba tarjetas perforadas (como un programa almacenado) para controlar el aumento de hilos en la fabricación de textiles.
- En 1823, Charles Babbage inventó la Máquina Diferencial, la cual podía hacer más que operaciones aritméticas simples; también podía resolver ecuaciones polinomiales. Posteriormente inventó una máquina llamada Máquina Analítica que, en cierta medida, es paralela a la idea de las computadoras modernas. Tenía cuatro componentes: un molino (ALU moderno), un almacén (memoria), un operador (unidad de control) y una salida (entrada/salida).
- En 1890, Herman Hollerit, mientras trabajaba en la Oficina de Censos de Estados Unidos, diseñó y construyó una máquina programadora que podía leer, contar y ordenar automáticamente los datos almacenados en las tarjetas perforadas.

NACIMIENTO DE LAS COMPUTADORAS ELECTRÓNICAS (1930-1950)

Entre 1930 y 1950, algunos científicos, que podrían considerarse los pioneros de la industria de la computación electrónica, inventaron varias computadoras.

Primeras computadoras electrónicas

Las primeras computadoras de este periodo no almacenaban el programa en memoria; todas se programaban externamente. Durante estos años destacaron cinco computadoras:

- La primera computadora para propósito especial que codificaba información de manera eléctrica fue inventada por John V. Atanasoff y su asistente Clifford Berry en 1939. Se le llamó ABC (Atanasoff Berry Computer) y se diseñó específicamente para resolver un sistema de ecuaciones lineales.
- Al mismo tiempo, el matemático alemán Konrad Zuse diseñó una máquina de propósito general llamada Z1.
- En la década de 1930, el ejército estadounidense e IBM patrocinaron un proyecto en la Universidad de Harvard bajo la dirección de Howard Airen para construir una computadora enorme llamada Mark I. Esta computadora usaba componentes eléctricos y mecánicos.
- En Inglaterra, Alan Turing inventó una computadora llamada Colossus diseñada para descifrar el código Enigma alemán.

- La primera computadora de propósito general totalmente electrónica fue fabricada por John Mauchly y J. Presper Eckert y recibió el nombre de ENIAC (*Electronic Numerical Integrator and Calculator:* Calculadora e integrador numérico electrónico). Se terminó en 1946. Utilizaba 18 000 tubos de vacío, medía 100 pies de largo por 10 pies de alto y pesaba 30 toneladas.

Computadoras basadas en el modelo de von Neumann

Las cinco computadoras precedentes utilizaron memoria sólo para almacenar datos. Se programaron externamente usando cables o interruptores. John von Neumann propuso que el programa y los datos deberían almacenarse en la memoria. De esa manera, cada vez que usted utilizara una computadora para realizar una tarea nueva, sólo necesitaría cambiar el programa en lugar de volver a conectar los cables de la máquina o encender y apagar cientos de interruptores.

La primera computadora basada en la idea de von Neumann se construyó en 1950 en la Universidad de Pennsylvania y se llamó EDVAC. Al mismo tiempo, Maurice Wilkes construyó una computadora similar llamada EDSAC en la Universidad de Cambridge en Inglaterra.

GENERACIONES DE COMPUTADORAS (1950-hoy día)

Las computadoras construidas después de 1950 siguieron, más o menos, al modelo de von Neumann. Las computadoras se han vuelto más rápidas, más pequeñas y más baratas, pero el principio es casi el mismo. Los historiadores dividen este periodo en generaciones, con cada generación presenciando un cambio importante en el hardware o el software (pero no en el modelo).

Primera generación

La primera generación (aproximadamente de 1950 a 1959) se caracteriza por la aparición de computadoras comerciales. Durante este periodo, las computadoras eran utilizadas sólo por profesionales. Estaban encerradas en habitaciones con acceso restringido únicamente al operador o a especialistas en computación. Las computadoras eran voluminosas y usaban tubos de vacío como interruptores electrónicos. En esta época las computadoras eran asequibles sólo para las grandes organizaciones.

Segunda generación

Las computadoras de la segunda generación (aproximadamente de 1959 a 1965) utilizaban transistores en lugar de tubos de vacío. Esto redujo su tamaño así como su costo y las puso al alcance de las empresas medianas y pequeñas. Dos lenguajes de programación de alto nivel, FORTRAN y COBOL (véase el Cap. 9), se inventaron y facilitaron la programación. Estos dos lenguajes separaron la tarea de la programación de la tarea de la operación de la computadora. Un ingeniero civil podía escribir un programa en FORTRAN para resolver un problema sin involucrarse en detalles electrónicos de la arquitectura de la computadora.

Tercera generación

La invención del **circuito integrado** (transistores, cableado y otros componentes en un solo chip) redujeron el costo y el tamaño de las computadoras aún más. Las minicomputadoras aparecieron en el mercado. Los programas empaquetados, popularmente conocidos como paquetes de **software**, se volvieron disponibles. Una pequeña empresa podía comprar un paquete que necesitaba (por ejemplo, para contabilidad) en lugar de escribir su propio programa. Una nueva industria nació, la industria del software. La generación duró más o menos de 1965 a 1975.

Cuarta generación

La cuarta generación (aproximadamente de 1975 a 1985) vio nacer las **microcomputadoras**. Las primera calculadora de escritorio (Altair 8800) se volvió disponible en 1975. Los avances en la industria de la electrónica permitieron que subsistemas de computadoras completos cupieran en una sola tarjeta de circuito. Esta generación también vio la aparición de las redes de computadoras (Cap. 6).

Quinta generación

Esta generación de duración indefinida comenzó en 1985. Presenció la aparición de las computadoras laptop y palmtop, mejoras en los medios de almacenamiento secundarios (CD-ROM, DVD, etc.), el uso de la multimedia y el fenómeno de la realidad virtual.

1.7 TÉRMINOS CLAVE

algoritmo
caja negra
ciencia de la computación
circuito integrado
datos de entrada
datos de salida
ingeniería de software
instrucción
lenguaje de computadora
memoria
microcomputadora
modelo de von Neumann
procesador de datos
procesador de datos programable
programa
sistema operativo
software
unidad de control
unidad lógica aritmética (ALU)

1.8 RESUMEN

- Ciencia de la computación, en este texto, se refiere a temas relacionados con una computadora.
- Una computadora es un procesador de datos programable que acepta datos de entrada y programas, y datos de salida.
- Un programa es una serie de instrucciones ejecutadas de manera secuencial que indican a la computadora qué hacer con los datos.
- Actualmente todas las computadoras se basan en el modelo de von Neumann.
- El modelo de von Neumann especifica un subsistema de memoria, un subsistema de unidad lógica aritmética, un subsistema de unidad de control y un subsistema de entrada/salida.
- Los datos y los programas se almacenan en la memoria de la computadora.
- Una solución paso a paso para un problema se llama algoritmo.
- Un programa es escrito en un lenguaje de computadora.
- La ingeniería de software es el diseño y la escritura de programas de forma estructurada.

1.9 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cómo se define a la ciencia de la computación en este libro?
2. ¿Qué modelo es la base para las computadoras de hoy?
3. ¿Por qué no se debe llamar a una computadora un procesador de datos?
4. ¿Qué requiere un procesador de datos programable para producir datos de salida?
5. ¿Cuáles son los subsistemas del modelo de computadora von Neumann?
6. ¿Cuál es la función del subsistema de memoria en el modelo de von Neumann?
7. ¿Cuál es la función del subsistema ALU en el modelo de von Neumann?
8. ¿Cuál es la función del subsistema de unidad de control en el modelo de von Neumann?
9. ¿Cuál es la función del subsistema de entrada/salida en el modelo de von Neumann?
10. Compare y contraste el contenido de la memoria de las primeras computadoras con el contenido de la memoria de una computadora basada en el modelo de von Neumann.
11. ¿De qué manera el modelo de von Neumann cambió el concepto de la programación?

PREGUNTAS DE OPCIÓN MÚLTIPLE

12. El modelo _____ es la base para las computadoras de hoy.
    a. Ron Neumann
    b. von Neumann
    c. Pascal
    d. Charles Babbage

13. En el modelo de von Neumann, el subsistema _____ almacena datos y programas.
    a. ALU
    b. entrada/salida
    c. memoria
    d. unidad de control

14. En el modelo de von Neumann, el subsistema _____ realiza cálculos y operaciones lógicas.
    a. ALU
    b. entrada/salida
    c. memoria
    d. unidad de control

15. En el modelo de von Neumann, el subsistema _____ acepta datos y programas y envía los resultados del procesamiento a dispositivos de salida.
    a. ALU
    b. entrada/salida
    c. memoria
    d. unidad de control

16. En el modelo de von Neumann, el subsistema _____ sirve como un administrador de los otros subsistemas.
    a. ALU
    b. entrada/salida
    c. memoria
    d. unidad de control

17. Según el modelo de von Neumann, _____ se almacenan en la memoria.
    a. sólo los datos
    b. sólo los programas
    c. los datos y los programas
    d. ninguno de los anteriores

18. Una solución paso a paso para un problema se llama _____.
    a. hardware
    b. un sistema operativo
    c. un lenguaje de computadora
    d. un algoritmo

19. FORTRAN y COBOL son ejemplos de _____.
    a. hardware
    b. sistemas operativos
    c. lenguajes de computadora
    d. algoritmos

20. Una máquina computadora del siglo XVII que podía realizar operaciones de suma y resta era la _____.
    a. Pascalina
    b. telar de Jacquard
    c. máquina analítica
    d. máquina de Babbage

21. _____ es una serie de instrucciones en un lenguaje de computación que indica a la computadora qué hacer con los datos.
    a. un sistema operativo hardware
    b. un algoritmo
    c. un procesador de datos
    d. un programa

22. _____ es el diseño y la escritura de un programa de forma estructurada.
    a. ingeniería de software
    b. ingeniería de hardware
    c. desarrollo de algoritmos
    d. arquitectura de instrucciones

23. La primera computadora electrónica para un uso especial se llamó _____.
    a. Pascal
    b. Pascalina
    c. ABC
    d. EDVAC

24. Una de las primeras computadoras basadas en el modelo de von Neumann se llamó _____.
    a. Pascal
    b. Pascalina
    c. ABC
    d. EDVAC

25. La primera máquina computadora en usar la idea de almacenamiento y programación se llamó _____.
    a. la Madeline
    b. EDVAC
    c. la máquina de Babbage
    d. el telar de Jacquard

26. Los _____ separaron la tarea de la programación de las tareas de operación de la computadora.
    a. algoritmos
    b. procesadores de datos
    c. lenguajes de programación de alto nivel
    d. sistemas operativos

EJERCICIOS

27. Utiliza Internet o acude a la biblioteca para averiguar cuándo se inventaron los teclados.

28. Utiliza Internet o acude a la biblioteca para averiguar cuándo se inventaron las impresoras.

29. Utiliza Internet o acude a la biblioteca para averiguar cuándo se inventaron los discos magnéticos.

30. Según el modelo de von Neumann, ¿los discos duros actuales pueden utilizarse como entrada o como salida? Explica.

31. Un lenguaje de programación tiene diez instrucciones diferentes. ¿Cuántos programas de cinco instrucciones pueden escribirse con este lenguaje sin repetir ninguna instrucción? ¿Cuántos programas de siete instrucciones?

32. ¿Actualmente qué es más costoso, el hardware o el software?

33. ¿Qué es más valioso para una organización hoy día, el hardware, el software o los datos?

34. ¿Cómo impone la organización de los datos un programa procesador de palabras?

35. Utiliza Internet o acude a la biblioteca para encontrar más información sobre la Pascalina.

36. Utiliza Internet o acude a la biblioteca para encontrar más información sobre la Rueda de Leibnitz.

37. Utiliza Internet o acude a la biblioteca para encontrar más información sobre el telar de Jacquard y su impacto social.

38. Utiliza Internet o acude a la biblioteca para encontrar más información sobre la Máquina Analítica.

39. Utiliza Internet o acude a la biblioteca para encontrar más información sobre Hollerit y su tabuladora.

Representación de datos

Como se estudió en el capítulo 1, una computadora es una máquina que procesa datos. Pero antes de que podamos hablar sobre el procesamiento de datos, necesita comprender la naturaleza de los mismos. En este capítulo y en el siguiente se analizan los diferentes tipos de datos y cómo se representan dentro de una computadora. En el capítulo 4 se muestra cómo se manipulan los datos en una computadora.

2.1 TIPOS DE DATOS

En la actualidad los datos se presentan de diferentes maneras, por ejemplo números, texto, imágenes, audio y video (figura 2.1). La gente necesita procesar todos estos tipos de datos.

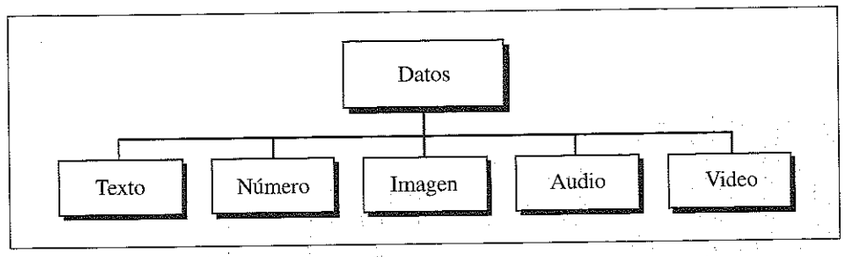


Figura 2.1 Diferentes tipos de datos

- Un programa de ingeniería utiliza una computadora principalmente para procesar números: hacer aritmética, resolver ecuaciones algebraicas o trigonométricas, encontrar las raíces de una ecuación diferencial, y así por el estilo.
- Un programa de procesamiento de palabras, por otra parte, utiliza una computadora más que nada para procesar texto: justificarlo, moverlo, eliminarlo, etcétera.
- Un programa de procesamiento de imágenes usa una computadora para manipular imágenes: crearlas, reducirlas, ampliarlas, rotarlas, etcétera.
- Una computadora también puede manejar datos de audio. Usted puede reproducir música en una computadora e introducir su voz como datos.
- Finalmente, una computadora puede usarse no sólo para mostrar películas, sino también para crear los efectos especiales que se ven en ellas.

La industria de la computación usa el término *multimedia* para definir información que contiene números, texto, imágenes, audio y video.

2.2 DATOS DENTRO DE LA COMPUTADORA

La pregunta es: ¿Cómo se manejan todos estos tipos de datos? ¿Se necesitan otras computadoras para procesar los distintos tipos de datos? Es decir, ¿se tiene una categoría de computadoras que procesan sólo números? ¿Hay una categoría de computadoras que procesan sólo texto?

Esta solución de diferentes computadoras para procesar distintos tipos de datos no es económica ni práctica porque los datos por lo general son una mezcla de tipos. Por ejemplo, aunque un banco procesa principalmente números, también necesita almacenar, como texto, los nombres de sus clientes. Como otro ejemplo, una imagen con frecuencia es una mezcla de gráficos y texto.

La solución más eficaz es usar una representación uniforme de los datos. Todo tipo de datos que entran del exterior a una computadora se transforman en esta representación uniforme cuando se almacenan en una computadora y se vuelven a transformar en su representación original cuando salen de la computadora. Este formato universal se llama *patrón de bits.*

BIT

Antes de continuar con el análisis de los patrones de bits, se debe definir un bit. Un **bit** (*binary digit:* **dígito binario**) es la unidad más pequeña de datos que puede almacenarse en una computadora; puede ser ya sea 0 o 1. Un bit representa el estado de un dispositivo que puede tomar uno de dos estados. Por ejemplo, un **interruptor** puede estar ya sea apagado o encendido. La convención es representar el estado de encendido como 1 y el estado de apagado como 0. Un interruptor electrónico puede representar un bit. En otras palabras, un interruptor puede almacenar un bit de información. Actualmente las computadoras utilizan varios dispositivos binarios de dos estados para almacenar datos.

PATRÓN DE BITS

Un solo bit no puede resolver el problema de la representación de datos, si cada pieza de datos pudiera representarse por un 1 o un 0, entonces sólo se necesitaría un bit. Sin embargo, usted necesita almacenar números más grandes, necesita almacenar texto, gráficos y otros tipos de datos.

Para representar diferentes tipos de datos se utiliza un **patrón de bits**, una secuencia o, como a veces se le llama, una cadena de bits. La figura 2.2 muestra un patrón de bits formado por 16 bits, es una combinación de ceros (0) y unos (1). Esto significa que si usted quiere almacenar un patrón de bits formado por 16 bits, necesita 16 interruptores electrónicos. Si quiere almacenar 1 000 patrones de bits, cada uno de 16 bits, necesita 16 000 bits y así sucesivamente.

```
1000101010111111
```

Figura 2.2 Patrón de bits

Ahora la pregunta es: ¿Cómo sabe la memoria de la computadora qué tipo de datos representa el patrón de bits? No lo sabe. La memoria de la computadora sólo almacena los datos como patrones de bits. Es responsabilidad de los dispositivos de entrada/salida o de los programas interpretar un patrón de bits como un número, texto o algún otro tipo de datos. En otras palabras, los datos se codifican cuando entran a la computadora y se decodifican cuando se presentan al usuario (figura 2.3).

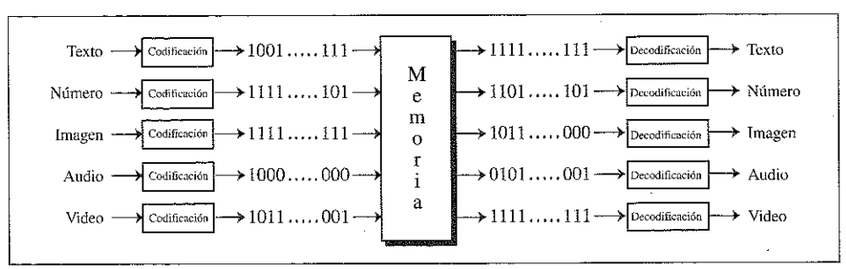


Figura 2.3 Ejemplos de patrones de bits

BYTE

Por tradición, un patrón de bits con una longitud de 8 bits se llama **byte**. Este término también se ha utilizado para medir el tamaño de la memoria o de otros dispositivos de almacenamiento. Por ejemplo, se dice que la memoria de una computadora que puede almacenar 8 millones de bits de información es una memoria de 1 millón de bytes.

2.3 REPRESENTACIÓN DE DATOS

Ahora podemos explicar cómo pueden representarse diferentes tipos de datos usando patrones de bits.

TEXTO

Una pieza de **texto** en cualquier idioma es una secuencia de símbolos usados para representar una idea en ese idioma. Por ejemplo, el idioma inglés utiliza 26 símbolos (A, B, C, . . . , Z) para representar las letras mayúsculas, 26 símbolos (a, b, c, . . . , z) para representar las letras minúsculas, 9 símbolos (0, 1, 2, . . . , 9) para los caracteres numéricos (no números; la diferencia se verá más adelante) y símbolos (., ?, :, ; . . . , !) para representar la puntuación. Otros símbolos como el espacio en blanco, la línea nueva y el tabulador se usan para alineación de texto y legibilidad.

Usted puede representar cada símbolo con un patrón de bits. Dicho de otra forma, texto como la palabra "BYTE", formada por cuatro símbolos, puede representarse como 4 patrones de bits, en los que cada patrón define un solo símbolo (figura 2.4).

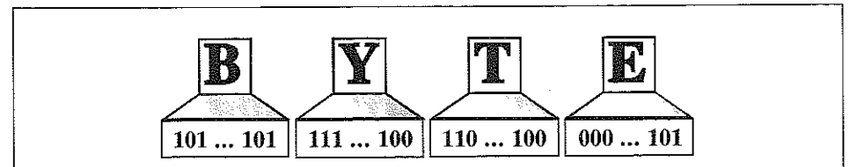


Figura 2.4 Representación de símbolos usando patrones de bits

La pregunta es: ¿Cuántos bits se necesitan en un patrón de bits para representar un símbolo en un idioma? Depende de cuántos símbolos haya en la secuencia. Por ejemplo, si usted crea un idioma imaginario que utilice sólo las letras mayúsculas del idioma inglés, sólo necesita 26 símbolos. Un patrón de bits en este idioma requiere representar al menos 26 símbolos. Para otro idioma, como el chino, pueden necesitarse muchos símbolos más. La longitud del patrón de bits que representa un símbolo en un idioma depende del número de símbolos usados en ese idioma. Más símbolos significan un patrón de bits más grande.

Aunque la longitud del patrón de bits depende del número de símbolos, la relación no es lineal; es logarítmica. Si se requieren dos símbolos, la longitud es 1 bit (el $\log_2 2$ es 1). Si se necesitan cuatro símbolos, la longitud es 2 bits ($\log_2 4$ es 2). La tabla 2.1 muestra esta relación, la cual es fácilmente perceptible. Un patrón de bits de 2 bits puede tomar cuatro formas diferentes: 00, 01, 10 y 11; cada una de las cuales representa un símbolo. Del mismo modo, un patrón de tres bits puede tomar ocho formas diferentes: 000, 001, 010, 011, 100, 101, 110 y 111.

Número de símbolos	Longitud del patrón de bits
2	1
4	2
8	3
16	4
. . .	. . .
128	7
256	8
. . .	. . .
65 536	16

Tabla 2.1 Número de símbolos y longitud de un patrón de bits

Códigos

Se han diseñado diferentes secuencias de patrones de bits para representar símbolos de texto. A cada secuencia se le conoce como **código** y al proceso de representar los símbolos se le llama codificación. En esta sección explicamos los códigos comunes.

ASCII El **Instituto Nacional Norteamericano de Estándares** (**ANSI**: *American Nacional Standards Institute*) desarrolló un código llamado **Código norteamericano de estándares para intercambio de información** (**ASCII**: *American Standard Code for Information Interchange*). Este código utiliza siete bits para cada símbolo. Esto significa que 128 (2^7) símbolos distintos pueden definirse mediante este código. Los patrones de bits completos para el código ASCII están en el apéndice A. La figura 2.5 muestra cómo se representa la palabra "BYTE" en código ASCII.

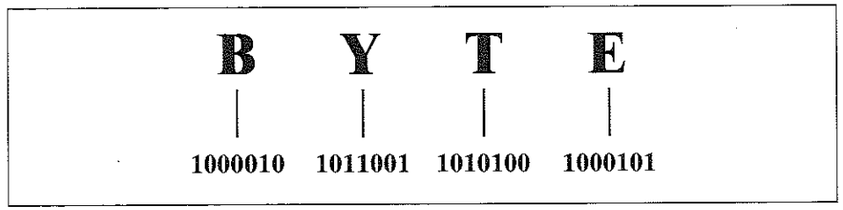


Figura 2.5 Representación de la palabra "BYTE" en código ASCII

La lista siguiente destaca algunas de las características de este código:

- ASCII utiliza un patrón de siete bits que varía de 0000000 a 1111111.
- El primer patrón (0000000) representa el carácter nulo (la ausencia de carácter).
- El último patrón (1111111) representa el carácter de eliminación.
- Hay 31 caracteres de control (no imprimibles).
- Los caracteres numéricos (0 a 9) se codifican antes que las letras.
- Hay varios caracteres de impresión especiales.
- Las letras mayúsculas (A ... Z) están antes que las letras minúsculas (a ... z).
- Los caracteres en mayúsculas y en minúsculas se distinguen sólo por un bit. Por ejemplo, el patrón para A es 1000001; el partón para a es 1100001. La única diferencia es el sexto bit a partir de la derecha.
- Hay seis caracteres especiales entre las letras mayúsculas y minúsculas.

ASCII extendido Para hacer que el tamaño de cada patrón sea de 1 byte (8 bits), a los patrones de bits ASCII se les aumenta un 0 más a la izquierda. Ahora cada patrón puede caber fácilmente en un byte de memoria. En otras palabras, en **ASCII extendido** el primer patrón es 00000000 y el último es 01111111.

Algunos fabricantes han decidido usar el bit de más para crear un sistema de 128 símbolos adicional. Sin embargo, este intento no ha tenido éxito debido a la secuencia no estándar creada por cada fabricante.

EBCDIC A principios de la era de las computadoras, IBM desarrolló un código llamado **Código extendido de intercambio decimal codificado en binario** (**EBCDIC**: *Extended Binary Coded Decimal Interchange Code*). Este código utiliza patrones de ocho bits, de manera que puede representar hasta 256 símbolos. Sin embargo, este código no se utiliza más que en computadoras mainframe de IBM.

Unicode Ninguno de los códigos anteriores representa símbolos que pertenecen a idiomas distintos al inglés. Por eso, se requiere un código con mucha más capacidad. Una coalición de fabricantes de hardware y software ha diseñado un código llamado **Unicode** que utiliza 16 bits y puede representar hasta 65 536 (2^{16}) símbolos. Diferentes secciones del código se asignan a los símbolos de distintos idiomas en el mundo. Algunas partes del código se usan para símbolos gráficos y especiales. El lenguaje Java™ utiliza este código para representar caracteres. Microsoft Windows usa una variación de los primeros 256 caracteres. En el apéndice B hay un pequeño conjunto de símbolos Unicode.

ISO La **Organización Internacional para la Estandarización** (*Internacional Standard Organization*), conocida como ISO, ha diseñado un código que utiliza patrones de 32 bits. Este código representa hasta 4 294 967 296 (2^{32}) símbolos, definitivamente lo suficiente para representar cualquier símbolo en el mundo actual.

NÚMEROS

En una computadora, los números se representan usando el **sistema binario.** En este sistema, un patrón de bits (una secuencia de ceros y unos) representa un número. Sin embargo, un código como el ASCII no se usa para representar datos. La razón para ello y un análisis de la representación de números se presentan en el capítulo 3.

IMÁGENES

Hoy día las **imágenes** se representan en una computadora mediante uno de dos métodos: **gráficos de mapa de bits** o **gráficos de vectores** (figura 2.6).

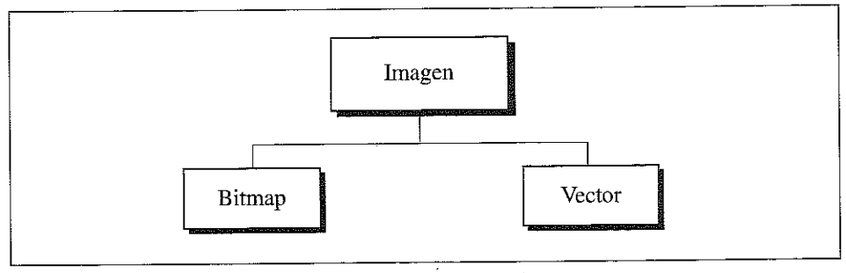


Figura 2.6 Métodos de representación de imágenes

Gráficos de mapa de bits

En este método, una imagen se divide en una matriz de **pixeles** (*picture elements:* **elementos de imagen**), donde cada pixel es un pequeño punto. El tamaño del pixel depende de lo que se conoce como *resolución*. Por ejemplo, una imagen puede dividirse en 1000 pixeles o 10 000 pixeles. En el segundo caso, aunque hay una mejor representación de la imagen (mejor resolución), se necesita más memoria para almacenarla.

Después de dividir una imagen en pixeles, a cada pixel se asigna un patrón de bits. El tamaño y el valor del patrón dependen de la imagen. Para una imagen formada sólo por puntos blancos y negros (por ejemplo, un tablero de ajedrez), un patrón de un bit es suficiente para representar un pixel. Un patrón de 0 representa un pixel negro y uno de 1 representa un pixel blanco. Luego los patrones se registran uno tras otro y se almacenan en la computadora. La figura 2.7 muestra una imagen de este tipo y su representación.

Si una imagen no se forma de pixeles puramente blancos y pixeles puramente negros, usted puede aumentar el tamaño del patrón de bits para representar escalas de grises. Por ejemplo, para mostrar cuatro niveles de la escala de grises, se puede usar un patrón de dos bits. Un pixel negro puede representarse por 00, un gris oscuro por 01, un pixel gris claro por 10 y un pixel blanco por 11.

Para representar imágenes a color, cada pixel coloreado se descompone en tres colores primarios: rojo, verde y azul (RGB). Luego se mide la intensidad de cada color y se le asigna un

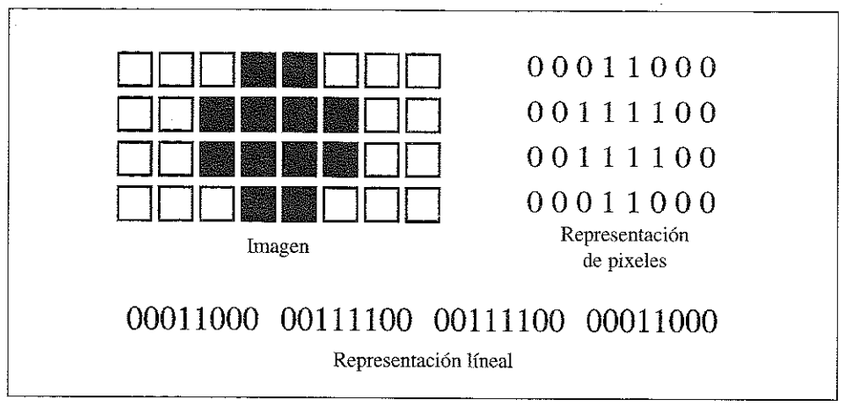


Figura 2.7 Método de gráficos de mapa de bits de una imagen blanca y negra

patrón de bits (por lo general ocho bits). En otras palabras, cada pixel tiene tres patrones de bits: uno para representar la intensidad del color rojo, uno para la intensidad del color verde y uno para la intensidad del color azul. Por ejemplo, la figura 2.8 muestra cuatro patrones de bits para algunos pixeles en una imagen a color.

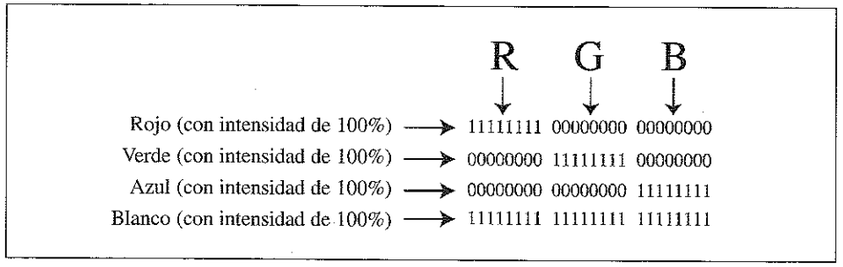


Figura 2.8 Representación de pixeles de color

Gráficos de vectores

El problema con el método de los gráficos de mapa de bits es que los patrones de bits exactos para representar una imagen particular deben guardarse en una computadora. Posteriormente, si usted desea cambiar el tamaño de la imagen debe cambiar el tamaño de los pixeles, lo cual crea una apariencia difusa y granulada. No obstante, el método de gráficos de vector no guarda los patrones de bits. Una imagen se descompone en una combinación de curvas y líneas. Cada curva o línea se representa por medio de una fórmula matemática. Por ejemplo, una línea puede describirse mediante las coordenadas de sus puntos extremos y un círculo puede describirse mediante las coordenadas de su centro y la longitud de su radio. La combinación de estas fórmulas se almacena en una computadora. Cuando la imagen se va a desplegar o imprimir, el tamaño de la imagen se proporciona al sistema como una entrada. El sistema rediseña la imagen con el nuevo tamaño y usa la misma fórmula para dibujar la imagen. En este caso, cada vez que una imagen se dibuja, la fórmula se vuelve a evaluar.

AUDIO

El audio es una representación de sonido o música. Aunque no hay un estándar para almacenar el sonido o la música, la idea es convertir el audio a datos **digitales** y usar patrones de bits. El audio por naturaleza es información **análoga**. Es continuo (análogo), no discreto (digital). La figura 2.9 muestra los pasos a seguir para cambiar los datos de audio a patrones de bits. Estos pasos son los siguientes:

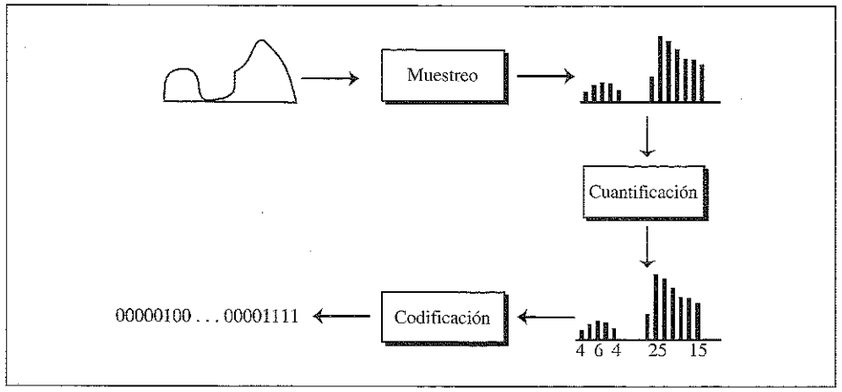


Figura 2.9 Representación de audio

1. La señal análoga se muestrea. El **muestreo** significa medir el valor de la señal a intervalos iguales.
2. Las muestras se cuantifican. La **cuantificación** significa asignar un valor (de un conjunto) a una muestra. Por ejemplo, si el valor de una muestra es 29.2 y el conjunto es el conjunto de enteros entre 0 y 63, se asigna un valor de 29 a la muestra.
3. Los valores cuantificados se cambian a patrones binarios. Por ejemplo, el número 25 se cambia al patrón binario 00011001 (consulte el capítulo 3 para la transformación de números en patrones).
4. Los patrones binarios se almacenan.

VIDEO

El **video** es una representación de imágenes (llamadas cuadros o *frames*) en el tiempo. Una película es una serie de cuadros desplegados uno tras otro para crear la ilusión de movimiento. Así que si usted sabe cómo almacenar una imagen dentro de una computadora, también sabe cómo almacenar un video; cada imagen o cuadro cambia a una serie de patrones de bits y se almacena. La combinación de las imágenes representa el video. Observe que el video actual se comprime normalmente. En el capítulo 15 estudiaremos MPEG, una técnica de compresión de video común.

2.4 NOTACIÓN HEXADECIMAL

El patrón de bits se diseñó para representar datos cuando éstos se almacenan dentro de una computadora. Sin embargo, para la gente es difícil manipular los patrones de bits. Escribir una serie de números 0 y 1 es tedioso y propenso al error. La notación hexadecimal ayuda.

La **notación hexadecimal** se basa en 16 (*hexadec* es la palabra griega para 16). Esto significa que hay 16 símbolos (dígitos hexadecimales): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. La importancia de la notación hexadecimal se hace evidente cuando se convierte un patrón de bits a notación hexadecimal.

Cada dígito hexadecimal puede representar cuatro bits y cuatro bits pueden representarse mediante un dígito hexadecimal. La tabla 2.2 muestra la relación entre un patrón de bits y un dígito hexadecimal.

Un patrón de 4 bits puede representarse mediante un dígito hexadecimal, y viceversa.

Patrón de bits	Dígito hexadecimal	Patrón de bits	Dígito hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Tabla 2.2 Dígitos hexadecimales

CONVERSIÓN

La conversión de un patrón de bits a notación hexadecimal se realiza por medio de la organización del patrón en grupos de cuatro y luego hallar el valor hexadecimal para cada grupo de cuatro bits. Para una conversión de hexadecimal a patrón de bits se convierte cada dígito hexadecimal a su equivalente de cuatro bits (figura 2.10).

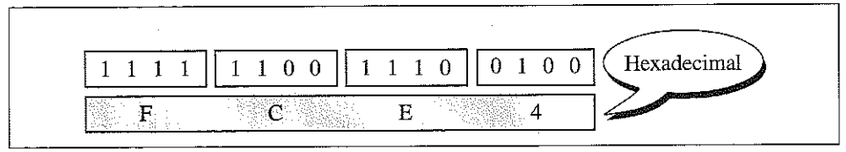


Figura 2.10 Transformación de binario a hexadecimal y de hexadecimal a binario

Observe que la notación hexadecimal se escribe en dos formatos. En el primer formato, usted añade una x minúscula (o mayúscula) antes de los dígitos para mostrar que la representación está en hexadecimal. Por ejemplo, xA34 representa un valor hexadecimal en esta convención. En otro formato, usted indica la base del número (16) como el subíndice después de cada notación. Por ejemplo, $A34_{16}$ muestra el mismo valor en la segunda convención. En este libro se usan ambas convenciones.

EJEMPLO 1

Determine el hexadecimal equivalente del patrón de bits 110011100010.

SOLUCIÓN

Cada grupo de cuatro bits se traduce a un dígito hexadecimal. El equivalente es xCE2. ■

EJEMPLO 2

Determine el hexadecimal equivalente del patrón de bits 0011100010.

SOLUCIÓN

El patrón de bits se divide en grupos de cuatro bits (a partir de la derecha). En este caso, se añaden dos 0 más a la izquierda para hacer el número total de bits divisible entre cuatro. Así que usted tiene 000011100010, lo cual se traduce a x0E2. ■

EJEMPLO 3

¿Cuál es el patrón de bits para x24C?

SOLUCIÓN

Cada dígito hexadecimal se escribe como su patrón de bits equivalente y se obtiene 001001001100. ■

2.5 NOTACIÓN OCTAL

Otra notación usada para agrupar patrones de bits es la notación octal. La **notación octal** se basa en 8 (*oct* es la palabra griega para ocho). Esto significa que existen ocho símbolos (dígitos octales): 0, 1, 2, 3, 4, 5, 6, 7. La importancia de la notación octal se hace evidente a medida que usted aprende a convertir un patrón de bits en notación octal.

Cada dígito octal representa tres bits y tres bits pueden representarse mediante un dígito octal. La tabla 2.3 muestra la relación entre un patrón de bits y un dígito octal.

Un patrón de tres bits puede representarse por medio de un dígito octal y viceversa.

Patrón de bits	Dígito octal	Patrón de bits	Dígito octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Tabla 2.3 Dígitos octales

CONVERSIÓN

La conversión de un patrón de bits a notación octal se realiza mediante la organización del patrón en grupos de tres y la determinación del valor octal de cada grupo de tres bits. Para la conversión de octal a patrón de bits, se convierte cada dígito octal a su equivalente de tres bits (figura 2.11).

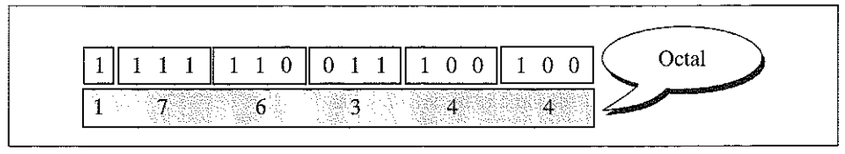


Figura 2.11 Transformación de binario a octal y de octal a binario

Observe que la notación octal también se escribe en dos formatos. En el primer formato, usted añade 0 (cero) antes de los dígitos para mostrar que la representación está en notación octal (a veces se utiliza una o minúscula). Por ejemplo, 0634 representa un valor octal en esta convención. En el otro formato, usted indica la base del número (8) como el subíndice después de la notación. Por ejemplo, 634_8 muestra el mismo valor en la segunda convención. En este libro se usan ambas convenciones.

EJEMPLO 4

Muestre el equivalente octal del patrón de bits 101110010.

SOLUCIÓN

Cada grupo de tres bits se traduce a un dígito octal. El equivalente es 0562, o562 o 562_8. ■

EJEMPLO 5

Muestre el equivalente octal del patrón de bits 1100010.

SOLUCIÓN

El patrón de bits se divide en grupos de tres bits (a partir de la derecha). En este caso, se añaden dos 0 más a la izquierda para hacer el número total de bits divisible entre 3. Así que usted tiene 001100010, lo cual se traduce a 0142, o142 o 142_8. ■

EJEMPLO 6

¿Cuál es el patrón de bits para 24_8?

SOLUCIÓN

Cada dígito octal se escribe como su patrón de bits equivalente para obtener 010100. ■

2.6 TÉRMINOS CLAVE

análogo
ASCII extendido
bit
byte
código
Código extendido de intercambio decimal codificado en binario (EBCDIC)
Código norteamericano de estándares para intercambio de información (ASCII)
cuantificación
digital
dígito binario
elemento de imagen
gráfico de mapa de bits
gráfico de vectores
imagen
Instituto Nacional Norteamericano de Estándares (ANSI)
interruptor
muestreo
notación hexadecimal
notación octal
Organización Internacional para la Estandarización (ISO)
patrón de bits
pixel
sistema binario
texto
Unicode
video

2.7 RESUMEN

- Los números, el texto, las imágenes, el audio y el video, todos son formas de datos. Las computadoras necesitan procesar todo tipo de datos.
- Todos los tipos de datos se transforman en una representación uniforme llamada patrón de bits para su procesamiento por la computadora.
- Un bit es la unidad más pequeña de datos que puede almacenarse en una computadora.
- Un interruptor, con sus dos estados de encendido y apagado, puede representar un bit.
- Un patrón de bits es una secuencia de bits que pueden representar un símbolo.
- Un byte son ocho bits.
- La codificación es el proceso de transformar datos en un patrón de bits.
- ASCII es un código popular para los símbolos.
- EBCDIC es un código utilizado en los mainframes de IBM.
- Unicode es un código de 16 bits y la ISO ha desarrollado un código de 32 bits. Ambos códigos permiten un mayor número de símbolos.
- Las imágenes utilizan el método de gráficos de mapa de bits o gráficos de vectores para representación de datos. La imagen se divide en pixeles a los que luego pueden asignarse patrones de bits.
- Los datos de audio se transforman a patrones de bits a través del muestreo, la cuantificación y la codificación.
- Los datos de video son una serie de imágenes en secuencia.

2.8 PRÁCTICA

PREGUNTAS DE REPASO

1. Nombre cinco tipos de datos que puede procesar una computadora.
2. ¿Cómo maneja una computadora todos los tipos de datos que debe procesar?
3. ¿Qué es un patrón de bits?
4. ¿Cuál es la diferencia entre ASCII y ASCII extendido?
5. ¿Qué es EBCDIC?
6. ¿Cómo se relaciona la longitud de un patrón de bits con el número de símbolos que puede representar?
7. ¿Cómo representa el método de gráficos de mapa de bits una imagen como un patrón de bits?
8. ¿Cuál es la ventaja del método de gráficos de vector sobre el método de gráficos de mapa de bits?
9. ¿Qué pasos deben seguirse para convertir datos de audio en patrones de bits?
10. ¿Cuál es la relación entre los datos de imagen y los datos de video?

PREGUNTAS DE OPCIÓN MÚLTIPLE

11. De lo siguiente, ¿qué puede clasificarse como datos?
    a. números
    b. video
    c. audio
    d. todos los anteriores

12. Para almacenar un byte, usted necesita _____ interruptores electrónicos.
    a. 1
    b. 2
    c. 4
    d. 8

13. Un byte consiste de _____ bits.
    a. 2
    b. 4
    c. 8
    d. 16

14. En un conjunto de 64 símbolos, cada símbolo requiere una longitud de patrón de bits de _____ bits.
    a. 4
    b. 5
    c. 6
    d. 7

15. ¿Cuántos símbolos pueden representarse mediante un patrón de bits con 10 bits?
    a. 128
    b. 256
    c. 512
    d. 1024

16. En ASCII extendido cada símbolo es _____ bits.
    a. 7
    b. 8
    c. 9
    d. 10

17. Si el código ASCII para E es 1000101, entonces el código ASCII para e es _____.
    a. 1000110
    b. 1000111
    c. 0000110
    d. 1100101

18. En código ASCII extendido, un _____ del patrón de bits para el código ASCII regular.
    a. bit 0 se añade a la izquierda
    b. bit 0 se añade a la derecha
    c. bit 1 se añade a la izquierda
    d. bit 1 se añade a la derecha

19. _____ es un código usado en los mainframes de IBM.
    a. ASCII
    b. ASCII extendido
    c. EBCDIC
    d. Unicode

20. _____ es un código de 16 bits que puede representar símbolos en idiomas distintos al inglés.
    a. ASCII
    b. ASCII extendido
    c. EBCDIC
    d. Unicode

21. _____ es un código usado por el lenguaje Java para representar caracteres.
    a. ASCII
    b. ASCII extendido
    c. EBCDIC
    d. Unicode

22. Un código de 32 bits se desarrolló por _____ para representar símbolos en todos los idiomas.
    a. ANSI
    b. ISO
    c. EBCDIC
    d. Hamming

23. Una imagen puede representarse en una computadora usando el método de _____.
- **a.** gráficos de mapa de bits
- **b.** gráficos de vectores
- **c.** gráficos de matrices
- **d.** a o b

24. El método de gráficos de mapa de bits y el método de gráficos de vectores se usan para representar _____ en una computadora.
- **a.** audio
- **b.** video
- **c.** imágenes
- **d.** números

25. En el método de gráficos _____ para la representación de una imagen en una computadora, a cada pixel se asigna uno o más patrones de bits.
- **a.** de mapa de bits
- **b.** de vectores
- **c.** cuantificado
- **d.** binario

26. En el método de gráficos _____ para la representación de una imagen en una computadora, la imagen se descompone en una combinación de curvas y líneas.
- **a.** de mapa de bits
- **b.** de vectores
- **c.** cuantificado
- **d.** binario

27. En el método de gráficos _____ para la representación de una imagen en una computadora, al cambiar el tamaño de la imagen se crea una imagen difusa y granulada.
- **a.** de mapa de bits
- **b.** de vectores
- **c.** cuantificado
- **d.** binario

28. Cuando usted quiere descargar música a una computadora, la señal de audio debe ser _____.
- **a.** muestreada
- **b.** cuantificada
- **c.** codificada
- **d.** todos los anteriores

EJERCICIOS

29. Dados cinco bits, ¿cuántos patrones de cinco bits distintos puede tener?

30. En cierto país las placas de la licencia vehicular tienen dos dígitos decimales (0 a 9). ¿Cuántas placas distintas puede tener? Si el dígito 0 no está permitido en las placas, ¿cuántas placas distintas puede tener?

31. Vuelva a hacer el ejercicio 30 para un placa que tiene dos dígitos seguidos por tres letras mayúsculas (A a Z).

32. Una máquina tiene ocho ciclos diferentes. ¿Cuántos bits se necesitan para representar cada ciclo?

33. La calificación de un estudiante en un curso puede ser A, B, C, D, F, R (retirado) o I (incompleto). ¿Cuántos bits se requieren para representar la calificación?

34. Una compañía ha decidido asignar un patrón de bits único a cada empleado. Si la compañía tiene 900 empleados, ¿cuál es el número mínimo de bits necesarios para crear este sistema de representación? ¿Cuántos patrones no son asignados? Si la compañía contrata otros 300 empleados, ¿debe aumentar el número de bits? Explique su respuesta.

35. Si utiliza un patrón de cuatro bits para representar los dígitos 0 a 9, ¿cuántos patrones de bits se usan?

36. Una imagen en escala de grises se digitaliza usando cuatro niveles de gris distintos. Si la imagen se compone de 100×100 pixeles, ¿cuántos bits se necesitan para representar la imagen?

37. Una señal de audio se muestrea 8000 veces por segundo. Cada muestra se representa mediante 256 niveles distintos. ¿Cuántos bits por segundo se necesitan para representar esta señal?

38. Cambie los siguientes patrones de bits a notación hexadecimal:
- **a.** 100011110000
- **b.** 1000001101
- **c.** 10001
- **d.** 11111111

39. Cambie los siguientes a patrones de bits:
- **a.** x120
- **b.** x2A34
- **c.** x00
- **d.** xFF

40. Cambie los siguientes patrones de bits a notación octal:
- **a.** 100011110000
- **b.** 1000001101
- **c.** 10001
- **d.** 11111111

41. Cambie los siguientes a patrones de bits:
- **a.** o12
- **b.** o27
- **c.** o45
- **d.** o20

42. ¿Cuántos dígitos hexadecimales se necesitan para convertir un patrón de 19 bits?

43. ¿Cuántos dígitos octales se requieren para convertir un patrón de 19 bits?

44. ¿Cuántos dígitos hexadecimales se necesitan para convertir un patrón de 6 bytes?

Representación de números

En el capítulo 2 mostramos cómo el texto, el audio, las imágenes y el video pueden representarse en una computadora mediante patrones de bits. Pospusimos el análisis de la representación de los números porque ésta es muy diferente de la representación de los datos no numéricos. Algunas de las razones de esta diferencia son las siguientes:

- Un código de caracteres como el ASCII no es eficiente para representar números. ASCII puede representar 128 símbolos, pero el sistema decimal necesita sólo 10. (Observe que si se consideran otros símbolos como +, − y el punto decimal, se necesitan aún más símbolos, pero todavía menos que 128.) Por ejemplo, si usted quiere almacenar el número 65 535 usando ASCII, necesita cinco bytes (un byte para cada dígito). Pero si el número se representa como un entero sin signo (usted verá esta representación posteriormente en este capítulo), sólo necesita dos bytes.
- Las operaciones con los números (por ejemplo, la suma y la resta) son muy complicadas si los dígitos de un número se representan en un código de caracteres.
- La representación de la precisión de un número (por ejemplo, el número de lugares después del punto decimal) requiere muchos bytes. Por ejemplo, para almacenar 23454.00001 se requieren 11 bytes, pero si el mismo número se representa en un punto flotante (esta representación se verá más adelante en este capítulo), necesita sólo unos cuantos bytes.

3.1 DECIMAL Y BINARIO

Dos sistemas de numeración predominan actualmente en el mundo de la computación: decimal y binario. Analizaremos estos dos tipos distintos de sistemas antes de presentar cómo se representan los números mediante una computadora.

SISTEMA DECIMAL

Hoy día, el mundo utiliza el **sistema decimal** para los números desarrollado por matemáticos árabes en el siglo VIII. Los primeros en usar un sistema numérico decimal fueron los antiguos egipcios. Los babilonios mejoraron el sistema egipcio al dar un significado a las posiciones del sistema numérico. Todos comprendemos fácilmente el sistema numérico decimal.

De hecho lo hemos usado tanto que es básicamente intuitivo. Pero, ¿realmente entendemos por qué la segunda posición en el sistema decimal representa las decenas y la tercera, las centenas? La respuesta yace en las potencias de la base del sistema, que es 10 en el sistema decimal. De esta manera la primera posición es 10 elevado a la potencia 0, la segunda posición es 10 elevado a la potencia 1 y la tercera posición es 10 elevado a la potencia 2. La figura 3.1 muestra la relación entre las potencias y el número 243.

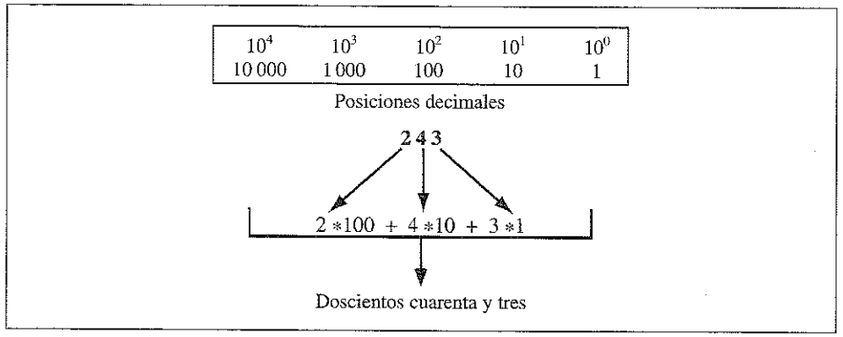


Figura 3.1 Sistema decimal

SISTEMA BINARIO

Mientras que el sistema decimal se basa en 10, el **sistema binario** se basa en 2. Sólo hay dos dígitos en el sistema binario, 0 y 1. La figura 3.2 muestra los valores posicionales para un sistema binario y el dígito 243 en binario. En la tabla de posiciones, cada posición es el doble de la posición anterior. De nuevo, esto se debe a que la base del sistema es 2. Las potencias binarias deben memorizarse cuando menos hasta 2^{10}.

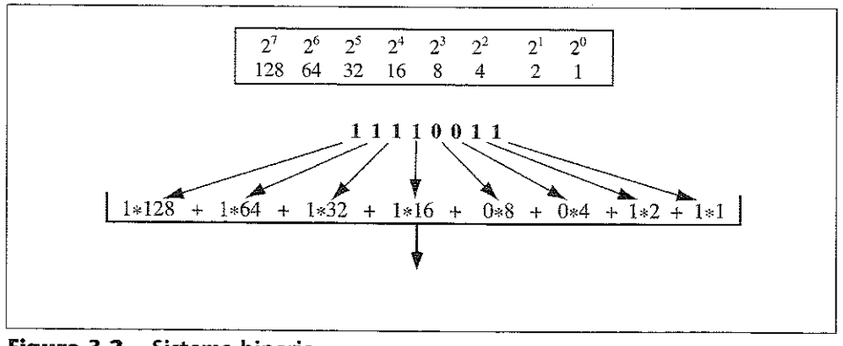


Figura 3.2 Sistema binario

3.2 CONVERSIÓN

Antes de estudiar cómo los números en forma de patrones de bits se almacenan dentro de una computadora, debe comprender cómo convertir manualmente un número del sistema decimal al sistema binario y viceversa.

CONVERSIÓN DE BINARIO A DECIMAL

Comenzaremos por convertir un número del sistema binario al sistema decimal. Tome el número binario y multiplique cada dígito binario por el valor de su posición. Como cada bit binario puede ser sólo 0 o 1, el resultado será ya sea 0 o el valor posicional. Después de multiplicar todos los dígitos sume los resultados. La **conversión de binario a decimal** se muestra en la figura 3.3.

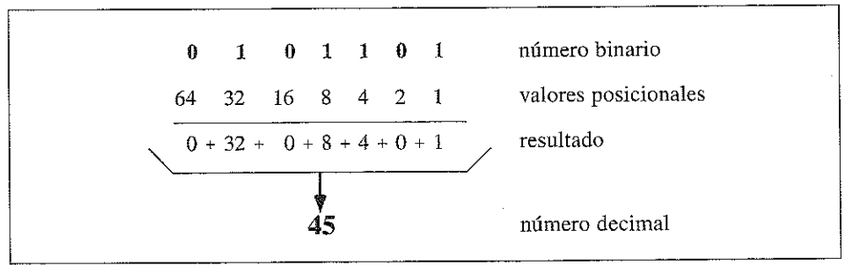


Figura 3.3 Conversión de binario a decimal

EJEMPLO 1

Convierta el número binario 10011 a decimal.

SOLUCIÓN

Se escriben los bits y sus valores posicionales. Se multiplica el bit por su valor correspondiente y luego se anota el resultado. Al final, los resultados se suman para obtener el número decimal.

Binario	**1**		**0**		**0**		**1**		**1**
Valores posicionales	16		8		4		2		1
	16	+	0	+	0	+	2	+	1
Decimal									**19**

■

CONVERSIÓN DE DECIMAL A BINARIO

Para convertir del sistema decimal al binario utilice la división repetitiva. El número original, 45 en el ejemplo, se divide por 2. El residuo (1) se vuelve el primer dígito binario y el segundo dígito se obtiene dividiendo el cociente (22) por 2 para determinar la siguiente posición. Este proceso continúa hasta que el cociente es 0. La **conversión de decimal a binario** se muestra en la figura 3.4.

EJEMPLO 2

Convierta el número decimal 35 a binario.

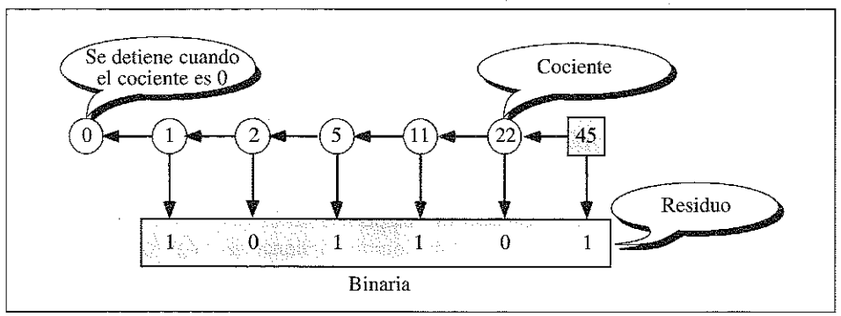


Figura 3.4 Conversión decimal a binaria

SOLUCIÓN

Se escribe el número en la esquina derecha. Se divide el número repetidamente por 2 y se anota el cociente y el residuo. Los cocientes se mueven a la izquierda y el residuo se anota bajo cada operación. Este proceso se suspende cuando el cociente es 0.

0	←	**1**	←	**2**	←	**4**	←	**8**	←	**17**	←	**35**	**(Decimal)**
		↓		↓		↓		↓		↓		↓	
Binario		**1**		**0**		**0**		**0**		**1**		**1**	■

3.3 REPRESENTACIÓN DE ENTEROS

Ahora que sabe cómo convertir del sistema decimal al sistema binario, veamos cómo almacenar enteros dentro de una computadora. Los **enteros** son **números íntegros** (es decir, números sin una fracción). Por ejemplo, 134 es un entero, pero 134.23 no lo es. Como otro ejemplo, −134 es un entero, pero −134.567 no lo es.

Un entero puede ser positivo o negativo. Un **entero negativo** varía del infinito negativo a 0; un **entero positivo** varía de 0 al infinito positivo (figura 3.5). Sin embargo, ninguna computadora puede almacenar todos los enteros en este intervalo. Para hacerlo, requeriría un número infinito de bits, lo cual significa una computadora con una capacidad de almacenamiento infinita.

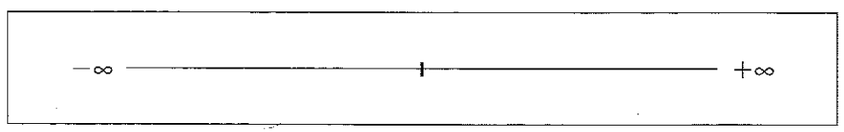


Figura 3.5 Intervalo de enteros

Para usar la memoria de una computadora de manera más eficiente, se han desarrollado dos amplias categorías de representación de enteros: enteros sin signo y enteros con signo. Los enteros con signo también pueden representarse de tres maneras distintas (figura 3.6).

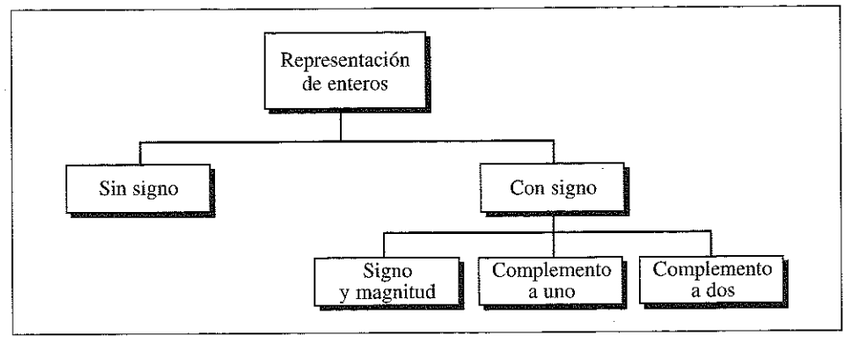


Figura 3.6 Taxonomía de enteros

En la actualidad la representación de uso más común es el complemento a dos. Sin embargo, primero estudiamos las otras representaciones debido a que son más simples y sirven como una buena base para el complemento a dos.

FORMATO DE ENTEROS SIN SIGNO

Un **entero sin signo** es un entero que no tiene intervalo, su rango está entre 0 y el infinito positivo. No obstante, como no hay manera de que una computadora represente a todos los enteros en este intervalo, la mayoría de las computadoras define una constante llamada el entero máximo sin signo. Un entero sin signo varía entre 0 y esta constante. El entero máximo sin signo depende del número de bits que la computadora asigna para almacenar un entero sin signo. A continuación se define el intervalo de los enteros sin signo en una computadora, donde N es el número de bits asignado para representar un entero sin signo:

Intervalo: $0 \ldots (2^N - 1)$

La tabla 3.1 muestra dos intervalos comunes para las computadoras de hoy.

Número de bits	Intervalo
8	0 . . . 255
16	0 . . . 65 535

Tabla 3.1 Intervalo de enteros sin signo

Representación

El almacenamiento de los enteros sin signo es un proceso sencillo según se esboza en los pasos siguientes:

1. El número cambia a binario.
2. Si el número de bits es menor que N, se añaden 0 a la izquierda del número binario de manera que haya un total de N bits.

EJEMPLO 3

Almacene 7 en una localidad de memoria de ocho bits.

SOLUCIÓN

Primero se cambia el número a binario: 111. Se añaden cinco 0 para hacer un total de N (8) bits: 00000111. El número se almacena en la memoria. ■

EJEMPLO 4

Almacene 258 en una localidad de la memoria de 16 bits. ■

SOLUCIÓN

Primero se cambia el número a binario: 100000010. Se añaden siete 0 para hacer un total de N (16) bits: 0000000100000010. El número se almacena en la localidad de la memoria.

La tabla 3.2 muestra cómo se almacenan los enteros no asignados en dos computadoras diferentes: una usa localidades de ocho bits y la otra usa localidades de 16 bits. Observe que los números decimales 258 y 24760 no pueden almacenarse en una computadora que usa localidades de ocho bits para un entero sin signo. El número decimal 1 245 678 no puede almacenarse en ninguna de estas dos computadoras; a esta condición se le llama desbordamiento (se estudia en el capítulo 4).

Decimal	Localidad de 8 bits	Localidad de 16 bits
7	00000111	0000000000000111
234	11101010	0000000011101010
258	Desbordamiento	0000000100000010
24 760	Desbordamiento	0110000010111000
1 245 678	Desbordamiento	Desbordamiento

Tabla 3.2 Almacenamiento de enteros sin signo en dos computadoras diferentes

Interpretación

¿Cómo se interpreta una representación binaria sin signo en decimal? El proceso es simple. Cambie los N bits del sistema binario al sistema decimal como se mostró al principio del capítulo.

EJEMPLO 5

Interprete 00101011 en decimal si el número se almacenó como un entero sin signo.

SOLUCIÓN

Usando el procedimiento mostrado en la figura 3.3, el número decimal es 43.

Desbordamiento

Si usted intenta almacenar un entero sin signo como 256 en una localidad de memoria de ocho bits, obtiene una condición llamada desbordamiento (*overflow*).

Aplicaciones

La representación de enteros sin signo puede mejorar la eficiencia del almacenamiento debido a que usted no necesita almacenar el signo de un entero. Esto significa que la localidad de bits del entero puede utilizarse para almacenar el número. La representación de enteros sin signo puede usarse siempre que no se necesiten los enteros negativos. En seguida se listan algunos casos:

- Conteo. Cuando se cuenta, no se necesitan los números negativos. Usted comienza contando a partir de 1 (a veces de 0) y continúa.
- Direccionamiento. Algunos lenguajes de computación almacenan la dirección de una localidad de memoria dentro de otra localidad de memoria. Las direcciones son números

positivos que comienzan a partir de 0 (el primer byte de memoria) y continúan hasta un número que representa la capacidad de memoria total en bytes. De nuevo, usted no necesita números negativos. Los enteros sin signo pueden hacer el trabajo fácilmente.

FORMATO DE SIGNO Y MAGNITUD

El almacenamiento de un entero en el formato de signo y magnitud requiere 1 bit para representar el signo (0 para positivo, 1 para negativo). Esto significa que en una asignación de ocho bits, usted sólo puede usar siete bits para representar el valor absoluto del número (número sin el signo). Por consiguiente, el máximo valor positivo es la mitad del valor sin signo. Lo siguiente define el intervalo de enteros de signo y magnitud en una computadora, donde N es el número de bits asignados para representar a un entero de signo y magnitud:

$$\text{Intervalo:} \quad -(2^{N-1}-1) \quad \ldots \quad +(2^{N-1}-1)$$

La tabla 3.3 muestra los intervalos comunes para las computadoras actuales. Observe que en este sistema hay dos 0: +0 y −0.

En la representación de signo y magnitud hay dos 0: positivo y negativo. En una asignación de ocho bits:

$$+0 \longrightarrow 00000000$$
$$-0 \longrightarrow 10000000$$

Número de bits	Rango
8	−127 ... −0 +0 ... +127
16	−32 767 ... −0 +0 ... +32 767
32	−2 147 483 647 ... −0 +0 ... +2 147 483 647

Tabla 3.3 Intervalo de enteros de signo y magnitud

Representación

Almacenar enteros de signo y magnitud es un proceso sencillo:

1. El número se cambia a binario; el signo se ignora.
2. Si el número de bits es menor que $N - 1$, los 0 se añaden a la izquierda del número de manera que haya un total de $N - 1$ bits.
3. Si el número es positivo, un 0 se añade a la izquierda (para volverlo de N bits). Si el número es negativo, se añade un 1 a la izquierda (para hacerlo de N bits).

En la representación de signo y magnitud, el bit en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

EJEMPLO 6

Almacene +7 en una localidad de memoria de ocho bits usando una representación de signo y magnitud.

SOLUCIÓN

Primero se cambia el número a binario: 111. Se añaden cuatro 0 para hacer un total de $N - 1$ (7) bits: 0000111. Luego se añade un cero más, que aquí se muestra en negritas, ya que el número es positivo. El resultado es 0000111. ■

EJEMPLO 7

Almacene −258 en una localidad de memoria de 16 bits usando una representación de signo y magnitud.

SOLUCIÓN

Primero se cambia el número a binario: 100000010. Se añaden seis 0 para hacer un total de $N - 1$ (15) bits: 000000100000010. Se añade un 1 más, que aquí se muestra en negritas, puesto que el número es negativo. El resultado es **1**000000100000010. ■

La tabla 3.4 muestra cómo se almacenan los números de signo y magnitud en dos computadoras diferentes; una que usa localidad de ocho bits y una que usa localidad de 16 bits.

Decimal	Localidad de 8 bits	Localidad de 16 bits
+7	**0**0000111	**0**000000000000111
−124	**1**1111100	**1**000000001111100
+258	Desbordamiento	**0**000000100000010
−24 760	Desbordamiento	**1**110000010111000

Tabla 3.4 Almacenamiento de enteros de signo y magnitud en dos computadoras diferentes

Interpretación

¿Cómo se interpreta una representación binaria de signo y magnitud en decimal? El proceso es simple:

1. Ignore el primer bit (el que está en el extremo izquierdo).
2. Cambie los $N - 1$ bits de binario a decimal como se explicó al principio del capítulo.
3. Agregue un signo + o − al número con base en el bit que está en el extremo izquierdo.

EJEMPLO 8

Interprete 10111011 en decimal si el número se almacenó como un entero de signo y magnitud.

SOLUCIÓN

Al ignorar el bit que está en el extremo izquierdo, los bits restantes son 0111011. Este número en decimal es 59. El bit en el extremo izquierdo es 1, así que el número es −59. ■

Aplicaciones

Actualmente, la representación de signo y magnitud no se usa para que las computadoras actuales almacenen números con signo. Hay cuando menos dos razones para ello. Primero, las operaciones como la suma y la resta no son sencillas para esta representación. Segundo, hay dos 0 en esta representación que vuelven las cosas difíciles para los programadores. Sin embargo, la representación de signo y magnitud tiene una ventaja: la transformación de decimal a binario y viceversa es muy fácil. Esto hace que esta representación sea conveniente para aplicaciones que no necesitan operaciones con números. Un ejemplo es el cambio de señales analógicas a señales digitales. Usted muestrea la señal analógica, asigna un número positivo o negativo a la muestra y lo cambia a binario para enviarlo por canales de comunicación de datos.

EL FORMATO DE COMPLEMENTO DE UNO

Tal vez haya notado que la representación de un número en el sistema binario es una cuestión de convención. En la representación de signo y magnitud adoptamos la convención de que el bit que está en el extremo izquierdo representa el signo; este bit no es parte del valor.

Los diseñadores de la **representación del complemento a uno** adoptaron una convención diferente: para representar un número positivo, usan la convención adoptada para un entero sin signo. Y para representar un número negativo, complementan el número positivo. En otras palabras, +7 se representa justo como un número sin signo, mientras que −7 se representa como el complemento de +7. En el complemento a uno, el complemento de un número se obtiene al cambiar todos los 0 a 1 y todos los 1 a 0.

A continuación se define el intervalo de los enteros complemento a uno en una computadora, donde N es el número de bits asignados para representar un entero complemento a uno:

$$\text{Intervalo:} \quad -(2^{N-1} - 1) \quad \ldots \quad +(2^{N-1} - 1)$$

Existen dos 0 en la representación del complemento de uno: positivo y negativo. En una asignación de 8 bits:

+0 ⟶ 00000000
−0 ⟶ 11111111

La tabla 3.5 muestra los intervalos comunes actuales para las computadoras. Observe que en este sistema hay al menos dos 0: un +0 y un −0.

Número de bits	Intervalo
8	−127 ... −0 +0 ... +127
16	−32 767 ... −0 +0 ... +32 767
32	−2 147 483 647 ... −0 +0 ... +2 147 483 647

Tabla 3.5 Rango de los enteros complemento de uno

Representación

Para almacenar los enteros complemento de uno se siguen estos pasos:

1. Cambie el número a binario; el signo es ignorado.
2. Añada uno o varios 0 a la izquierda del número para hacer un total de N bits.
3. Si el signo es positivo, no se necesita ninguna otra acción. Si el signo es negativo, complemente cada bit (cambie 0 por 1 y 1 por 0).

En la representación del complemento de uno, el bit que está en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

EJEMPLO 9

Almacene +7 en una localidad de memoria de ocho bits usando la representación de complemento a uno.

SOLUCIÓN

Primero se cambia el número a binario: 111. Se añaden cinco 0 de modo que haya un total de $N(8)$ bits: 00000111. El signo es positivo así que no se requiere realizar otra acción. ■

EJEMPLO 10

Almacene −256 en una localidad de memoria de 16 bits usando la representación del complemento de uno.

SOLUCIÓN

Primero se cambia el número a binario: 100000010. Se añaden siete 0 con el fin de que haya un total de $N(16)$ bits: 0000000100000010. El signo es negativo, de manera que cada bit complementa. El resultado es 1111111011111101. ■

La tabla 3.6 muestra cómo se almacenan los números complemento a uno en dos computadoras distintas: una que usa localidades de ocho bits y otra que usa localidades de 16 bits.

Decimal	Localidades de 8 bits	Localidades de 16 bits
+7	00000111	0000000000000111
−7	11111100	1111111111111000
+124	01111100	0000000001111100
−124	10000011	1111111110000011
+24 760	Desbordamiento	0110000010111000
−24 760	Desbordamiento	1001111101000111

Tabla 3.6 Almacenamiento de enteros complemento a uno en dos computadoras distintas

Interpretación

¿Cómo se interpreta una representación binaria del complemento a uno en decimal? El proceso implica estos pasos:

1. Si el bit que está en el extremo izquierdo es 0 (número positivo),
   a. se cambia el número entero de binario a decimal.
   b. se pone un signo más (+) enfrente del número.
2. Si el bit que está en el extremo izquierdo es 1 (bit negativo),
   a. se complementa el número entero (cambiando todos los 0 a 1, y viceversa).
   b. se cambia el número entero de binario a decimal.
   c. se pone un signo negativo (−) enfrente del número.

EJEMPLO 11

Interprete 11110110 en decimal si el número se almacenó como un entero complemento a uno.

SOLUCIÓN

El bit en el extremo izquierdo es 1, de modo que el número es negativo. Primero se complementa. El resultado es 00001001. El complemento en decimal es 9, de manera que el número original era −9. Observe que el complemento de un complemento es el número original. ■

La operación complemento a uno significa invertir todos los bits. Si se aplica la operación complemento a uno a un número positivo, se obtiene el número negativo correspondiente. Si se aplica la operación complemento a uno a un número negativo, se obtiene el número positivo correspondiente. Si un número se complementa dos veces, se obtiene el número original.

Desbordamiento

Si se intenta almacenar un entero complemento a uno como +256 en una localidad de memoria de ocho bits, se obtiene una condición llamada desbordamiento.

Aplicaciones

Actualmente la representación del complemento a uno no se usa para almacenar números en computadoras. Hay al menos dos razones para ello. Primero, las operaciones como la suma y la resta no son sencillas para esta representación. Segundo, hay dos 0 en esta representación, lo cual vuelve las cosas difíciles para los programadores. Sin embrago, esta representación tiene cierta relevancia. Primero, es la base para la siguiente representación (el complemento a dos). Segundo, tiene propiedades que la vuelven interesante para aplicaciones de comunicación de datos tales como la detección y corrección de errores.

FORMATO DEL COMPLEMENTO A DOS

Como se mencionó previamente, la representación del complemento a uno tiene dos 0 (+0 y −0). Esto puede crear un poco de confusión en los cálculos. Además, en el siguiente capítulo se verá que si se suma un número y su complemento (+4 y −4) en esta representación, se obtiene −0 negativo en lugar de +0 positivo. La **representación del complemento a dos** resuelve todos estos problemas.

El complemento a dos es la representación de enteros más común, más importante y de más amplio uso en la actualidad.

A continuación se define el intervalo de los enteros complemento a dos en una computadora, donde N es un número de bits asignados a un entero complemento a dos:

$$\text{Intervalo:} \quad -(2^{N-1}) \quad \ldots \quad +(2^{N-1}-1)$$

La tabla 3.7 muestra los intervalos comunes actuales para las computadoras. Observe que en este sistema hay sólo un 0 y que el principio del intervalo es 1 menos que aquel para el complemento a uno.

Número de bits	Intervalo
8	−128 ... −0 +0 ... +127
16	−32 768 ... −0 +0 ... +32 767
32	−2 147 483 648 ... −0 +0 ... +2 147 483 647

Tabla 3.7 Intervalo de números complemento a dos

Representación

Para almacenar el complemento a dos se deben seguir estos pasos:

1. El número se cambia a binario; el signo se ignora.
2. Si el número de bits es menor que N se añaden 0 a la izquierda del número de manera que haya un total de N bits.
3. Si el signo es positivo, no se necesita una acción posterior. Si el signo es negativo, todos los 0 en el extremo derecho y el primer 1 permanecen sin cambios. El resto de los bits se complementa.

En la representación del complemento a dos, el bit en el extremo izquierdo define el signo del número. Si éste es 0, el número es positivo. Si es 1, el número es negativo.

EJEMPLO 12

Almacene +7 en una localidad de memoria de ocho bits usando la representación del complemento a dos.

SOLUCIÓN

Primero se cambia el número a binario: 111. Se añaden cinco 0 con el fin de que haya un total de $N(8)$ bits: 00000111. El signo es positivo, así que no se requiere realizar ninguna otra acción. ■

EJEMPLO 13

Almacene −40 en una localidad de memoria de 16 bits usando la representación del complemento a dos.

SOLUCIÓN

Primero se cambia el número a binario: 101000. Se añaden diez 0 de modo que haya un total de $N(16)$ bits: 0000000000101000. El signo es negativo, así que los 0 en el extremo derecho permanecen sin cambio hasta el primer 1 (inclusive) y el resto se complementa. El resultado es 1111111111101**1000**. ■

La tabla 3.8 muestra cómo se almacenan los números complemento a dos en dos computadoras distintas, una que usa localidad de ocho bits y otra que usa localidad de 16 bits.

Decimal	Localidad de 8 bits	Localidad de 16 bits
+7	00000111	0000000000000111
−7	11111001	1111111111111001
+124	01111100	0000000001111100
−124	10000100	1111111110000100
+24 760	Desbordamiento	0110000010111000
−24 760	Desbordamiento	1001111101001000

Tabla 3.8 Ejemplo de representaciones de complementos a dos en dos computadoras

Sólo hay un 0 en el complemento a dos: En una asignación de ocho bits:

0 ⟶ 00000000

Interpretación

¿Cómo se interpreta una representación binaria del complemento a dos en decimal? El proceso involucra los pasos siguientes:

1. Si el bit en el extremo izquierdo es 0 (número positivo),
   a. se cambia todo el número de binario a decimal.
   b. se pone un signo más (+) enfrente del número.
2. Si el bit en el extremo izquierdo es 1 (número negativo),
   a. se dejan los bits en el extremo derecho hasta el primer 1 (inclusive) como están.
   b. se cambia todo el número de binario a decimal.
   c. se pone un signo negativo (−) enfrente del número.

EJEMPLO 14

Interprete 11110110 en decimal si el número se almacenó como un entero complemento a dos.

SOLUCIÓN

El bit en el extremo izquierdo es 1. El número es negativo. El 10 a la derecha se deja tal como está y el resto se complementa. El resultado es 00001010. El número complemento a dos es 10, así que el número original era −10.

La operación complemento a dos puede lograrse al invertir todos los bits, excepto los bits que están en el extremo derecho hasta el primer 1 (inclusive). Si se aplica la operación complemento a dos a un número positivo, se obtiene el número negativo correspondiente. Si se aplica la operación complemento a dos a un número negativo, se obtiene el número positivo correspondiente. Si un número se complementa dos veces, se obtiene el número original.

Aplicaciones

La representación del complemento a dos es la representación estándar actual para almacenar enteros en las computadoras. En el capítulo siguiente, usted verá por qué ocurre esto cuando se considera la simplicidad de las operaciones que usan el complemento a dos.

RESUMEN DE LA REPRESENTACIÓN DE ENTEROS

Para darse una idea general de los métodos de representación de números, examine la tabla 3.9. En esta tabla, suponga que N es 4. La localidad de memoria puede almacenar sólo cuatro bits. Si usted mira el contenido de la localidad de memoria, puede interpretar el número en una de las cuatro representaciones. Aun cuando la interpretación es la misma para los enteros positivos, es diferente para los enteros negativos.

Contenido de la memoria	Sin signo	Signo y magnitud	Complemento a uno	Complemento a dos
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	−0	−7	−8
1001	9	−1	−6	−7
1010	10	−2	−5	−6
1011	11	−3	−4	−5
1100	12	−4	−3	−4
1101	13	−5	−2	−3
1110	14	−6	−1	−2
1111	15	−7	−0	−1

Tabla 3.9 Resumen de la representación de enteros

3.4 SISTEMA EXCESS

Otra representación que permite almacenar tanto números positivos como negativos en una computadora es el **sistema Excess**. En este sistema, es fácil transformar un número de decimal a binario, y viceversa. Sin embargo, las operaciones con los números son muy complicadas. Su única aplicación en uso actualmente es en el almacenamiento del valor exponencial de una fracción. Esto se estudia en la sección siguiente.

En una conversión Excess, un número positivo conocido como el número mágico se utiliza en el proceso de conversión. El número mágico normalmente es (2^{N-1}) o $(2^{N-1} - 1)$ donde N es la asignación de bits. Por ejemplo, si N es 8, el número mágico es ya sea 128 o 127. En el primer caso, llamamos a la representación Excess_128 y en el segundo, Excess_127.

Representación

Para representar un número en Excess, utilice el procedimiento siguiente:

1. Sume el número mágico al entero.
2. Cambie el resultado a binario y añada uno o varios 0 de modo que haya un total de N bits.

EJEMPLO 15

Represente −125 en Excess_127 usando localidades de ocho bits.

SOLUCIÓN

Primero se suma 127 a −25 y se obtiene 102. Este número en binario es 1100110. Añada un bit para obtener una longitud de ocho bits. La representación es 01100110. ■

Interpretación

Para interpretar un número en Excess, utilice el siguiente procedimiento:

1. Cambie el número a decimal.
2. Reste el número mágico del entero.

EJEMPLO 16

Interprete 11111110 si la representación está en Excess_127.

SOLUCIÓN

Primero cambie el número a decimal: 254. Luego reste 127 del número. El resultado es 127 en decimal. ■

3.5 REPRESENTACIÓN DE PUNTO FLOTANTE

Para representar un **número de punto flotante** (un número que contiene un entero y una **fracción**), el número se divide en dos partes: el entero y la fracción. Por ejemplo, el número de punto flotante 14.234 tiene un entero de 14 y una fracción de 0.234.

CONVERSIÓN A BINARIO

Para convertir un número de punto flotante a binario, utilice el procedimiento siguiente:

1. Convierta la parte entera a binario.
2. Convierta la fracción a binario.
3. Ponga un punto decimal entre las dos partes.

Conversión de la parte entera

Este procedimiento es el mismo que se presentó en el capítulo 2.

Conversión de la parte fraccionaria

Para convertir una fracción a binario, use la multiplicación repetitiva. Por ejemplo, para convertir 0.125 a binario, multiplique la fracción por 2; el resultado es 0.250. La parte entera del resultado (0) se extrae y se vuelve el dígito binario en el extremo izquierdo. Ahora multiplique por 2 la parte fraccionaria (0.250) del resultado para obtener 0.50. De nuevo, se extrae la parte entera del resultado y se vuelve el siguiente dígito binario. Este proceso continúa hasta que la parte fraccionaria se vuelve 0 o cuando se llega al límite del número de bits que se pueden usar (figura 3.7).

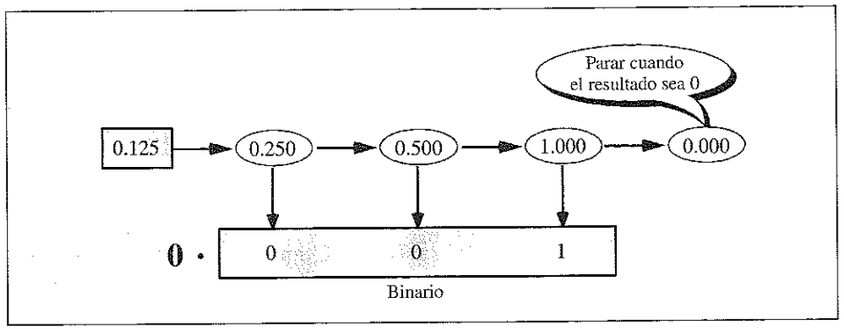


Figura 3.7 Cambio de fracciones a binario

EJEMPLO 17

Transforme la fracción 0.875 a binario.

SOLUCIÓN

La fracción se escribe en la esquina izquierda. El número se multiplica continuamente por 2 y se extrae la parte entera como dígito binario. El proceso se detiene cuando el número es 0.0.

Fracción	0.875	→	1.750	→	1.50	→	1.0	→	0.0
			↓		↓		↓		
Binario:	**0 .**		**1**		**1**		**1**		

■

EJEMPLO 18

Transforme la fracción 0.4 a un binario de 6 bits.

SOLUCIÓN

La fracción se escribe en la esquina izquierda. El número se multiplica continuamente por 2 y se extrae la parte entera como dígito binario. En este caso, usted no puede obtener la representación binaria exacta debido a que reaparece la fracción original. Sin embargo, puede continuar con el proceso hasta obtener seis bits.

Fracción	0.4	→	0.8	→	1.6	→	1.2	→	0.4	→	0.8	→	1.6
			↓		↓		↓		↓		↓		↓
Binario:	**0 .**		**0**		**1**		**1**		**0**		**0**		**1**

■

NORMALIZACIÓN

Para representar el número 71.3125 (+1000111.0101), en la memoria se almacena el signo, todos los bits y la posición del punto decimal. Aunque esto es posible, dificulta las operaciones con números. Se necesita una representación estándar para números de punto flotante. La solución es la **normalización**, es decir, el desplazamiento del punto decimal para que haya sólo un 1 a la izquierda del punto decimal.

1 . xxxxxxxxxxxxxxxxxx

Para indicar el valor original del número, éste se multiplica por 2^e, donde e es el número de bits en que se desplazó el punto decimal: positivo para el desplazamiento izquierdo y negativo para el desplazamiento derecho. Un signo positivo o negativo se añade luego dependiendo del signo del número original. La tabla 3.10 muestra ejemplos de la normalización.

Número original	Desplazamiento	Normalizado
+1010001.11001	⟵ 6	$+2^6 \times$ **1.01000111001**
−111.000011	⟵ 2	$-2^2 \times$ **1.11000011**
+0.00000111001	6 ⟶	$+2^{-6} \times$ **1.11001**
−0.001110011	3 ⟶	$-2^{-3} \times$ **1.110011**

Tabla 3.10 Ejemplos de la normalización

SIGNO, EXPONENTE Y MANTISA

Después de que se normaliza un número, se almacenan sólo tres piezas de información del mismo: signo, exponente y mantisa (los bits a la derecha del punto decimal). Por ejemplo, +1000111.0101 se convierte en:

+	2^6	×	1.0001110101
Signo: +	**Exponente: 6**		**Mantisa: 0001110101**

Observe que el 1 a la izquierda del punto decimal no se almacena; esto es comprensible.

Signo

El signo de un número puede almacenarse usando 1 bit (0 o 1).

Exponente

El exponente (potencia de 2) define el movimiento del punto decimal. Observe que la potencia puede ser negativa o positiva. El método que se utiliza para almacenar el exponente es la representación de Excess. El número de bits asignado N define el intervalo de números que una computadora puede almacenar.

Mantisa

La **mantisa** es el número binario a la derecha del punto decimal. Define la precisión del número. La mantisa se almacena como un entero sin signo.

ESTÁNDARES IEEE

El Instituto de Ingenieros Electrónicos y Eléctricos (IEEE: Institute of Electrical and Electronics Engineers) ha definido tres estándares para almacenar números de punto flotante; dos de ellos se utilizan para almacenar números en la memoria (precisión simple y precisión doble). Estos formatos se muestran en la figura 3.8. Observe que el número dentro de los cuadros es el número de bits para cada campo. Analizamos el formato de precisión simple y dejamos el **formato de precisión doble** como un ejercicio.

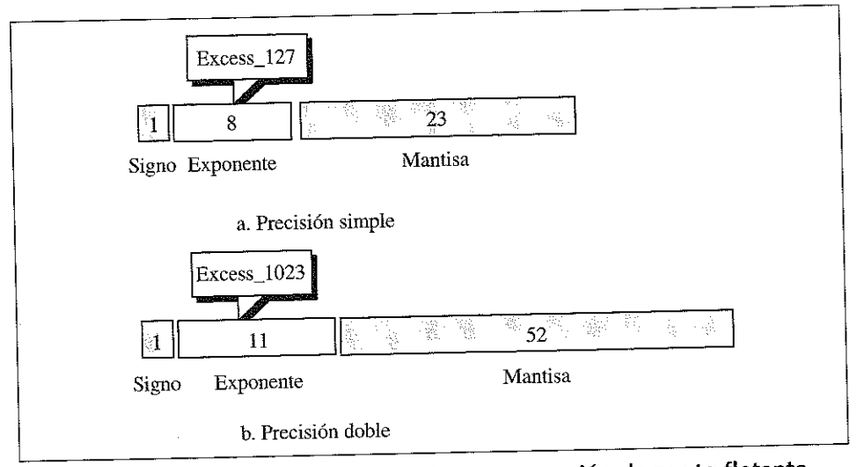


Figura 3.8 Estándares del IEEE para la representación de punto flotante

Representación de precisión simple

El procedimiento para almacenar un número de punto flotante en la memoria mediante el uso del **formato de precisión simple** es el siguiente:

1. Almacene el signo como 0 (positivo) o 1 (negativo).
2. Almacene el exponente (potencia de 2) como Excess_127.
3. Almacene la mantisa como un entero sin signo.

EJEMPLO 19

Muestre la representación del número normalizado

+ 2^6 × **1.01000111001**

SOLUCIÓN

El signo es positivo y se representa como 0. El exponente es 6. En la representación Excess_127, se le suma 127 y se obtiene 133. En binario es 10000101. La mantisa es 01000111001. Cuando se aumenta la longitud a 23, se obtiene 01000111001000000000000. Advierta que no se puede ignorar el 0 a la izquierda porque es una fracción, sin considerar que el 0 es lo mismo que multiplicar el número por 2. Observe que los 0 se añadieron a la derecha (no a la izquierda) debido a que se trata de una fracción. La adición de 0 a la derecha de una fracción no modifica a la fracción, pero añadir 0 a la izquierda significa dividir el número entre una potencia de 2. El número en la memoria es un número de 32 bits como se muestra en seguida:

Signo	exponente	mantisa
0	**10000101**	01000111001000000000000

■

La tabla 3.11 exhibe más ejemplos de representación de punto flotante.

Número	Signo	Exponente	Mantisa
$-2^2 \times$ 1.11000011	1	10000001	11000011000000000000000
$+2^{-6} \times$ 1.11001	0	01111001	11001000000000000000000
$-2^{-3} \times$ 1.110011	1	01111100	11001100000000000000000

Tabla 3.11 Ejemplos de representación de punto flotante

Interpretación de punto flotante para precisión simple

El procedimiento siguiente interpreta un número de punto flotante de 32 bits almacenado en la memoria.

1. Use el bit del extremo izquierdo como el signo.
2. Cambie los siguientes ocho bits a formato decimal y reste 127 del resultado. Éste es el exponente.
3. Añada un 1 y un punto decimal a los siguientes 32 bits. Usted puede ignorar cualquier 0 adicional a la derecha.
4. Mueva el punto decimal a la posición correcta usando el valor del exponente.
5. Cambie la parte entera a decimal.
6. Cambie la parte fraccionaria a decimal.
7. Combine las partes entera y fraccionaria.

EJEMPLO 20

Interprete el siguiente número de punto flotante de 32 bits:

1 01111100 11001100000000000000000

SOLUCIÓN

El bit en el extremo izquierdo es el signo (−). Los siguientes ocho bits son 01111100. Éste es 124 en decimal. Si se le resta 127, se obtiene el exponente −3. Los 23 bits siguientes son la mantisa. Si se ignoran los 0 adicionales, se obtiene 110011. Después de que se añade un 1 a la izquierda del punto decimal, el número normalizado en binario es:

$$-2^{-3} \times 1.110011$$ ■

3.6 NOTACIÓN HEXADECIMAL

Como se estudió en el capítulo 2, los números pueden representarse en notación hexadecimal. Por ejemplo, el número 48 puede representarse en un formato sin signo de ocho bits como 00110000 o como el hexadecimal x30. De manera similar, un número como 81.5625 puede representarse en el estándar IEEE como 0**10000101**01000111001000000000000 o en notación hexadecimal como x42A39000.

3.7 TÉRMINOS CLAVE

conversión de binario a decimal
conversión de decimal a binario
entero
entero negativo
entero positivo
entero sin signo
formato de precisión doble
formato de precisión simple
fracción
mantisa
normalización
número de punto flotante
número entero
representación de signo y magnitud
representación del complemento a uno
representación del complemento a dos
sistema binario
sistema decimal
sistema Excess

3.8 RESUMEN

- El sistema decimal tiene diez dígitos y se basa en potencias de 10.
- El sistema binario, utilizado por computadoras para almacenar números, tiene dos dígitos, 0 y 1, y se basa en potencias de 2.
- La asignación de bits es el número de bits utilizado para representar a un entero.
- Los enteros pueden representarse como números enteros con signo y sin signo.
- Existen tres métodos principales de representación de números con signo: signo y magnitud, complemento a uno y complemento a dos.
- Los números sin signo se usan comúnmente para conteo y direccionamiento.
- En el método de signo y magnitud de la representación de enteros, un bit representa el signo de número; los bits restantes representan la magnitud.
- En el método de representación de enteros del complemento a uno, un número negativo se representa por medio de la operación complemento del número positivo correspondiente.
- Complementar un número significa convertir cada 1 a 0 y cada 0 a 1.
- En el método de representación de enteros del complemento a uno, un número negativo se representa al dejar todos los 0 en el extremo derecho y el primer 1 sin cambio y luego complementar los bits restantes.
- La mayoría de las computadoras actuales utiliza el método de representación de enteros del complemento a dos.
- Tanto el método de signo y magnitud como el de complemento a uno tienen dos representaciones para el valor 0; el complemento a dos tiene sólo una representación para el valor 0.
- Un número de punto flotante es un número entero y una fracción. La conversión de la fracción a binario requiere que el denominador de la fracción se exprese como una potencia de 2. El sistema Excess_X se utiliza para almacenar esta potencia de 2.
- Una fracción se normaliza con la finalidad de que las operaciones sean más simples.
- Para almacenar una fracción en la memoria, se necesita su signo, exponente y mantisa.

3.9 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cómo se convierte un número decimal a binario?
2. ¿Cómo se convierte un número binario a decimal?
3. En un sistema binario, ¿cada posición es una potencia de qué número?
4. En un sistema decimal, ¿cada posición es una potencia de qué número?
5. ¿Cuáles son los tres métodos para representar enteros con signo?
6. ¿Cuál es el significado del término *máximo entero sin signo*?
7. ¿Cuál es el significado del término *asignación de bits*?
8. ¿Por qué no se puede almacenar el número decimal 256 en una localidad de memoria de ocho bits?
9. ¿Cuáles son los usos de los enteros sin signo? Mencione dos.
10. ¿Qué sucede cuando se intenta almacenar el decimal 130 utilizando la representación de signo y magnitud con una asignación de ocho bits?
11. Compare y contraste la representación de los enteros positivos en los métodos de signo y magnitud, complemento a uno y complemento a dos.
12. Compare y contraste la representación de los enteros negativos en los métodos de signo y magnitud, complemento a uno y complemento a dos.
13. Compare y contraste la representación del 0 en los métodos de signo y magnitud, complemento a uno y complemento a dos.
14. Compare y contraste el rango de números que pueden representarse en los métodos de signo y magnitud, complemento a uno y complemento a dos.
15. Comente el papel que juega el bit en el extremo izquierdo en los métodos de signo y magnitud, complemento a uno y complemento a dos.
16. ¿Cuál es el uso principal del sistema Excess_X?
17. ¿Por qué es necesaria la normalización?
18. ¿Qué es la mantisa?
19. Después de que se normaliza un número, ¿qué tipo de información almacena una computadora en la memoria?

PREGUNTAS DE OPCIÓN MÚLTIPLE

20. Los únicos dígitos usados en el sistema numérico _____ son 0 y 1.
- **a.** decimal
- **b.** octal
- **c.** binario
- **d.** hexadecimal

21. Cuando se convierte un número decimal a binario, se divide repetidamente en _____.
- **a.** 2
- **b.** 8
- **c.** 10
- **d.** 16

22. ¿Cuál de los siguientes métodos de representación de enteros maneja tanto números positivos como negativos?
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** todos los anteriores

23. En los enteros sin signo, una asignación de cuatro bits permite _____ números negativos.
- **a.** 7
- **b.** 8
- **c.** 15
- **d.** 16

24. En todas las representaciones de enteros sin signo, una asignación de cuatro bits permite _____ números no negativos.
- **a.** 7
- **b.** 8
- **c.** 15
- **d.** 16

25. En la representación de números (de) _____, 1111 en la memoria representa −0.
- **a.** enteros sin signo
- **b.** signo y magnitud
- **c.** complemento a uno
- **d.** complemento a dos

26. En la representación de números (de) _____, 1111 en la memoria representa −1.
- **a.** enteros sin signo
- **b.** signo y magnitud
- **c.** complemento a uno
- **d.** complemento a dos

27. En la representación de números (de) _____, hay dos representaciones para 0.
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** a y b

28. En la representación de números (de) _____, hay una representación para 0.
- **a.** enteros sin signo
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** a y c

29. Si el bit en el extremo izquierdo es 0 en la representación de números de _____, entonces el número decimal es positivo.
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** todas las anteriores

30. Si el bit en el extremo izquierdo es 1 en la representación de números de _____, entonces el número decimal es positivo.
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** ninguna de las anteriores

31. ¿Cuál es el método de representación de números de uso más difundido hoy día para almacenar números en una computadora?
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** enteros sin signo

32. ¿Cuál método de representación de números se usa con frecuencia para convertir señales analógicas en señales digitales?
- **a.** enteros sin signo
- **b.** signo y magnitud
- **c.** complemento a uno
- **d.** b y c

33. Los enteros sin signo pueden utilizarse para _____.
- **a.** conteo
- **b.** direccionamiento
- **c.** procesamiento de señales
- **d.** a y b

34. ¿Cuál método de representación de números se usa con frecuencia para almacenar el valor exponencial de una fracción?
- **a.** enteros sin signo
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** Excess_X

35. En una conversión Excess_X, usted _____ el número mágico X al número que se va a convertir.
- **a.** suma
- **b.** resta
- **c.** multiplica
- **d.** divide

36. En la representación de números Excess_X, ¿generalmente cuál es la relación entre X y *N*, la asignación de bits?
- **a.** $X = 2^N - 1$
- **b.** $X = 2^N + 1$
- **c.** $X = 2^{N-1} - 1$
- **d.** a o c

37. Cuando una computadora almacena una fracción, el _____ por lo general se expresa como una potencia de 2.
- **a.** numerador
- **b.** denominador
- **c.** número entero
- **d.** a o b

38. Si el denominador de una fracción es 1 024, entonces el numerador tiene _____ de longitud.
- **a.** 2
- **b.** 8
- **c.** 10
- **d.** 16

39. Si el numerador de una fracción tiene tres bits de longitud, entonces el denominador es _____.
- **a.** 2
- **b.** 8
- **c.** 10
- **d.** 16

40. ¿Cuál es la representación de 5 en Excess_128?
- **a.** 00000101
- **b.** 10000100
- **c.** 10000101
- **d.** 10000001

41. Cuando una fracción se normaliza, hay un _____ a la izquierda del punto decimal.
- **a.** bit 0
- **b.** bit 1
- **c.** secuencia de bits aleatoria
- **d.** a o b

42. Un número normalizado se multiplica por _____ donde *e* es el número de bits que se desplazó el punto decimal.
- **a.** $2e$
- **b.** $e/2$
- **c.** e^2
- **d.** 2^e

43. Cuando una fracción se normaliza, la computadora almacena el (la) _____.
- **a.** signo
- **b.** exponente
- **c.** mantisa
- **d.** todos los anteriores

44. La precisión del número fraccionario almacenado en una computadora se define por el (la) _____.
- **a.** signo
- **b.** exponente
- **c.** mantisa
- **d.** ninguno de los anteriores

45. ¿Cómo se almacena la mantisa en una computadora?
- **a.** en el complemento a uno
- **b.** en el complemento a dos
- **c.** como un entero sin signo
- **d.** en signo y magnitud

46. Un dígito octal convertido a binario se compone por _____ bits.
- **a.** 2
- **b.** 3
- **c.** 4
- **d.** 8

EJERCICIOS

47. Cambie los siguientes números decimales a enteros de ocho bits sin signo si es posible.
- **a.** 23
- **b.** 121
- **c.** 34
- **d.** 342

48. Cambie los siguientes números decimales a enteros sin signo de 16 bits.
- **a.** 41
- **b.** 411
- **c.** 1234
- **d.** 342

49. Cambie los siguientes números decimales a enteros de signo y magnitud de ocho bits.
- **a.** 32
- **b.** −101
- **c.** 56
- **d.** 129

50. Cambie los siguientes números decimales a enteros de signo y magnitud de 16 bits.
- **a.** 142
- **b.** −180
- **c.** 560
- **d.** 2456

51. Cambie los siguientes números decimales a enteros complemento a uno de 16 bits.
- **a.** 162
- **b.** −110
- **c.** 2560
- **d.** 12 123

52. Cambie los siguientes números decimales a enteros complemento a dos de ocho bits.
- **a.** −12
- **b.** −101
- **c.** 56
- **d.** 142

53. Cambie los siguientes números decimales a enteros complemento a dos de 16 bits.
- **a.** 102
- **b.** −179
- **c.** 534
- **d.** 62 056

54. Cambie los siguientes números sin signo de ocho bits a decimal.
- **a.** 01101011
- **b.** 10010100
- **c.** 00000110
- **d.** 01010000

55. Cambie los siguientes números signo y magnitud de ocho bits a decimal.
- **a.** 01111011
- **b.** 10110100
- **c.** 01100011
- **d.** 11010000

56. Cambie los siguientes números complemento a uno de ocho bits a decimal.
- **a.** 01100011
- **b.** 10000100
- **c.** 01110011
- **d.** 11000000

57. Cambie los siguientes números complemento a dos de ocho bits a decimal.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110100
- **d.** 11001110

58. Los siguientes números son números binarios de signo y magnitud. Muestre cómo cambiar el signo del número.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110111
- **d.** 11001110

59. Los números siguientes son números binarios complemento a uno. Muestre cómo cambiar el signo del número.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110111
- **d.** 11001110

60. Los números siguientes son números binarios complemento a dos. Muestre cómo cambiar el signo del número.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110111
- **d.** 11001110

61. En este capítulo mostramos cómo aplicar la operación del complemento a dos de un número al saltar algunos bits y complementar el resto (cambiar los 0 a 1 y los 1 a 0). Otro método es primero aplicar la operación complemento a uno y luego sumar 1 al resultado. Pruebe ambos métodos con los números siguientes. Compare y contraste los resultados.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110100
- **d.** 11001110

62. Si usted aplica la operación complemento a uno a un número dos veces, obtiene el número original. Aplique dos veces la operación complemento a uno a cada uno de los números siguientes y vea si puede obtener el número original.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110100
- **d.** 11001110

63. Si usted aplica la operación complemento a dos a un número dos veces, obtiene el número original. Aplique dos veces la operación complemento a dos a cada uno de los números siguientes y vea si puede obtener el número original.
- **a.** 01110111
- **b.** 11111100
- **c.** 01110100
- **d.** 11001110

64. El equivalente de un número complemento a uno en decimal se llama complemento de nueve (10 −1 es 9). De esta manera, el complemento de nueve de un número se obtiene al restar cada dígito de 9. Encuentre el complemento a nueve para cada uno de los siguientes números decimales. Suponga que está utilizando sólo tres dígitos.
- **a.** +234
- **b.** +112
- **c.** −125
- **d.** −111

65. Si usted usa tres dígitos, ¿cuál es el intervalo de números que puede representar usando el complemento de nueve? En este sistema, ¿cómo puede determinar el signo de un número? ¿Se tienen dos ceros en este sistema? ¿Cuál es la representación para +0? ¿Cuál es la representación para −0?

66. Normalice los siguientes números binarios de punto flotante. Muestre de manera explícita el valor del exponente después de la normalización.
- **a.** 1.10001
- **b.** $2^3 \times 111.1111$
- **c.** $2^{-2} \times 101.110011$
- **d.** $2^{-5} \times 101101.00000110011000$

67. Muestre los siguientes números en formato IEEE de 32 bits.
- **a.** $-2^0 \times 1.10001$
- **b.** $+2^3 \times 1.111111$
- **c.** $+2^{-4} \times 1.01110011$
- **d.** $-2^{-5} \times 1.01101000$

68. Muestre los siguientes números en formato IEEE de 64 bits.
- **a.** $-2^0 \times 1.10001$
- **b.** $+2^3 \times 1.111111$
- **c.** $+2^{-4} \times 1.01110011$
- **d.** $-2^5 \times 1.01101000$

69. Cambie las siguientes fracciones decimales a fracciones binarias.
- **a.** 3 5/8
- **b.** 12 3/32
- **c.** 4 13/64
- **d.** 12 5/128

70. La tabla 3.12 muestra otra forma de escribir una fracción de manera que el denominador sea una potencia de 2 (1, 4, 8, 16, etc.). Sin embargo, a veces se necesita una combinación de entradas para encontrar la fracción apropiada. Por ejemplo, 0.635 no está en la tabla, pero usted sabe que 0.625 es 0.5 + 0.125. Esto significa que 0.625 puede escribirse como 1/2 + 1/8, o 5/8, lo cual es la forma adecuada. Cambie las fracciones decimales siguientes a una fracción con una potencia de 2.

Original	Nueva	Original	Nueva
0.5	$^1/_2$	0.25	$^1/_4$
0.125	$^1/_8$	0.0625	$^1/_{16}$
0.03125	$^1/_{32}$	0.015625	$^1/_{64}$

Tabla 3.12 Ejercicio 70

- **a.** 0.1875
- **b.** 0.640625
- **c.** 0.40625
- **d.** 0.375

71. Usando los resultados del problema previo, cambie los siguientes números decimales a números binarios.
- **a.** 7.1875
- **b.** 12.640625
- **c.** 11.40625
- **d.** 0.375

72. Usando el resultado del problema anterior, muestre los siguientes números en el formato IEEE de 32 bits.
- **a.** +7.1875
- **b.** +12.640625
- **c.** −11.40625
- **d.** −0.375

73. Muestre el resultado de las operaciones siguientes usando el formato IEEE.
- **a.** x012A00 + x12AAFF
- **b.** x0000011 + x820000
- **c.** x9111111 + x211111
- **d.** xE111111 + x777777

Operaciones con bits

En los capítulos 2 y 3 mostramos la manera de almacenar diferentes tipos de datos en una computadora. En este capítulo, mostramos cómo hacer operaciones con bits. Las operaciones con bits pueden dividirse en dos amplias categorías: operaciones aritméticas y operaciones lógicas (figura 4.1).

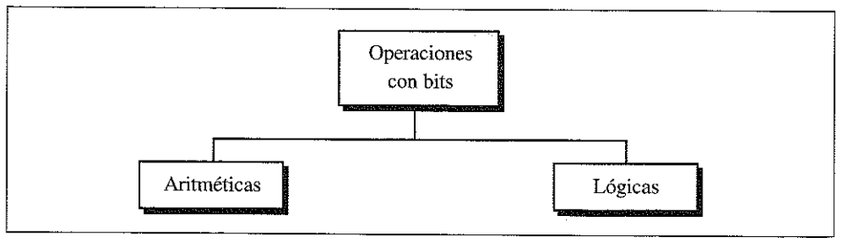


Figura 4.1 Operaciones con bits

4.1 OPERACIONES ARITMÉTICAS

Las **operaciones aritméticas** involucran la suma, la resta, la multiplicación, la división y así por el estilo. Usted puede aplicar estas operaciones a los enteros y a los números de punto flotante. Primero nos concentraremos en los enteros.

OPERACIONES ARITMÉTICAS CON ENTEROS

Todas las operaciones aritméticas como la suma, la resta, la multiplicación y la división pueden aplicarse a los enteros. Sin embrago, sólo nos concentraremos en la suma y la resta. La operación de multiplicación puede implementarse en software usando sumas sucesivas, o en hardware usando otras técnicas. La operación de división también puede implementarse en software usando restas sucesivas o en hardware usando otras técnicas. No obstante, un análisis profundo de la multiplicación y la división está más allá del ámbito de este libro y lo dejamos como un tema para los libros sobre arquitectura de computadoras.

Usted puede aplicar la suma y la resta a todas las representaciones de enteros. Sin embargo, presentamos la representación del **complemento a dos** debido a que éste es el único método utilizado actualmente para almacenar enteros en las computadoras.

Suma en el complemento a dos

La suma de números en el complemento a dos es como la suma de números en el sistema decimal; usted suma columna por columna y si existe un acarreo, éste se suma a la siguiente columna. Sin embargo, debe recordar que está tratando con dígitos binarios, no con dígitos decimales. Cuando se suman dos bits el resultado es 0 o 1. Asimismo, se debe **acarrear** un 1 que se transmite a la siguiente columna. Por consiguiente, necesita ser cuidadoso al sumar dos o tres bits. La tabla 4.1 es una guía para sumar bits.

Número de 1	Resultado	Acarreo
Ninguno	0	
Uno	1	
Dos	2	1
Tres	3	1

Tabla 4.1 Suma con bits

Ahora podemos definir la regla para sumar dos enteros en el complemento a dos:

Regla para sumar enteros en el complemento a dos

Sume dos bits y propague el número de acarreo a la siguiente columna. Si hay un acarreo final después de la suma de la columna del extremo izquierdo, descártelo.

EJEMPLO 1

Sume dos números en la representación del complemento a dos:

$$(+17) + (+22) \longrightarrow (+39)$$

SOLUCIÓN

Estos números en el complemento a dos se representan como 00010001 y 00010110 para una localidad de memoria de ocho bits. El resultado es similar para cualquier tamaño de asignación.

```
Acarreo        1
               0 0 0 1 0 0 0 1 +
               0 0 0 1 0 1 1 0
               -----------------
Resultado      0 0 1 0 0 1 1 1
```

El resultado es 39 en decimal.

EJEMPLO 2

Sume 24 y −17. Ambos números están en formato de complemento a dos.

$$(+24) \quad + \quad (-17) \quad \longrightarrow \quad (+7)$$

SOLUCIÓN

Los números en el complemento a dos pueden representarse como sigue:

```
Acarreo        1 1 1 1 1
                 0 0 0 1 1 0 0 0 +
                 1 1 1 0 1 1 1 1
                 ---------------
Resultado        0 0 0 0 0 1 1 1
```

Observe que el resultado es +7 y que el último acarreo (a partir de la columna en el extremo izquierdo) se descarta.

EJEMPLO 3

Sume −35 y 20. Ambos números están en formato de complemento a dos.

$$(-35) \quad + \quad (+20) \quad \longrightarrow \quad (-15)$$

SOLUCIÓN

Los números en el complemento a dos pueden representarse como sigue:

```
Acarreo          1 1 1
                 1 1 0 1 1 1 0 1 +
                 0 0 0 1 0 1 0 0
                 ---------------
Resultado        1 1 1 1 0 0 0 1
```

Observe que el resultado es −15 (si se aplica la operación complemento a dos al resultado, se obtiene 15).

EJEMPLO 4

Sume 127 y 3. Ambos números están en formato del complemento a dos.

$$(+127) \quad + \quad (+3) \quad \longrightarrow \quad (+130)$$

SOLUCIÓN

Los números en el complemento a dos pueden representarse como sigue:

```
Acarreo        1 1 1 1 1 1 1
               0 1 1 1 1 1 1 1 +
               0 0 0 0 0 0 1 1
               ---------------
Resultado      1 0 0 0 0 0 1 0
```


Un error resalta de inmediato aquí. El bit en el extremo izquierdo del resultado es 1, lo cual significa que el número es negativo (esperamos un número positivo). ¿Cuál es este número? El complemento a dos es 126. Esto significa que el número es −126 en lugar de 130. En la siguiente sección sobre desbordamiento tratamos este problema. ■

Desbordamiento

El **desbordamiento** es un error que ocurre cuando usted intenta almacenar un número que no está dentro del intervalo definido por la asignación. Cuando sume números en el complemento a dos usando N bits, asegúrese de que cada número y el resultado estén en el intervalo definido para la representación del complemento a dos. En el capítulo 3, mencionamos que el intervalo de números que pueden representarse en el complemento a dos está entre -2^{N-1} y $2^{N-1}-1$.

Intervalo de números en la representación del complemento a dos:
$-(2^{N-1})$ —— o —— $+(2^{N-1}-1)$

Para el ejemplo 4, el intervalo es -2^{8-1} a $+2^{8-1}-1$, lo cual es −128 a 127. El resultado de la suma (130) no está en el intervalo. Pero otra pregunta sigue sin respuesta: ¿Por qué la respuesta es −126? La respuesta puede encontrarse si usted visualiza los números en el complemento a dos como puntos de un círculo (Fig. 4.2).

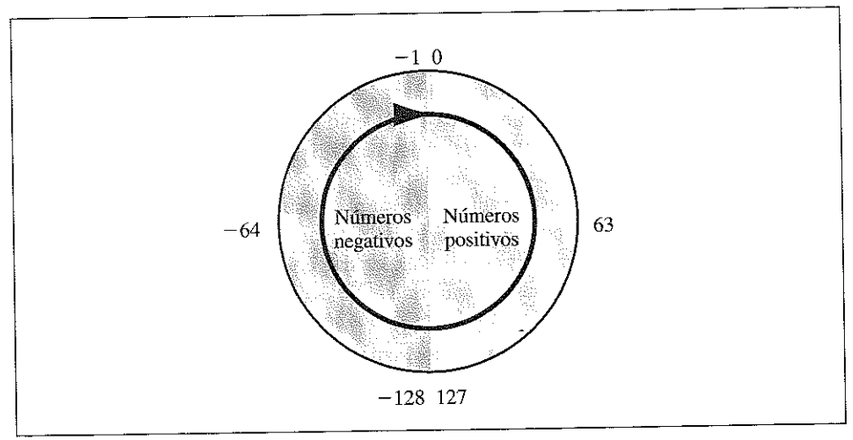


Figura 4.2 Visualización de números en el complemento a dos

Comience desde 0 y sume 1 sin interrupción hasta que obtenga 127. Si suma un 1 más no obtiene 128, sino −128. Si suma otro 1, obtiene −127. Sume un 1 más y obtendrá −126. Esto fue lo que ocurrió en el ejemplo 4.

Cuando realice operaciones aritméticas con números en una computadora, recuerde que cada número y el resultado deben estar en el intervalo definido por la asignación con bits.

Resta en el complemento a dos

Una de las ventajas de la representación del complemento a dos es que no hay diferencia entre la suma y la resta. Para restar, se invierte (complemento a dos) el segundo número y se suma. En otras palabras, las dos expresiones siguientes son lo mismo:

$$\text{Número 1} - \text{Número 2} \longleftrightarrow \text{Número 1} + (-\text{Número 2})$$

EJEMPLO 5

Reste 62 de 101 en formato del complemento a dos.

$$(+101) - (+62) \longleftrightarrow (+101) + (-62) \longrightarrow (+39)$$

SOLUCIÓN

Los números en el complemento a dos pueden representarse como sigue: ■

Acarreo	1 1
	0 1 1 0 0 1 0 1 +
	1 1 0 0 0 0 1 0
Resultado	0 0 1 0 0 1 1 1

El resultado es +39. Observe que el acarreo en el extremo izquierdo se descarta. ■

OPERACIONES ARITMÉTICAS EN NÚMEROS DE PUNTO FLOTANTE

Todas las operaciones aritméticas como la suma, la resta, la multiplicación y la división pueden aplicarse a los números de punto flotante (flotantes). Sin embargo, sólo nos concentramos en la suma y la resta porque la multiplicación es simplemente una suma sucesiva y la división, una resta sucesiva.

La suma y la resta en los flotantes, cuando se almacenan en formato IEEE, son muy complicadas y detalladas. No podemos cubrir todos los detalles y casos especiales aquí, así que sólo damos el concepto general.

Suma y resta

La suma y la resta para los números de punto flotante son un proceso. Los pasos son como sigue:

- Revise los signos.
  a. Si los signos son iguales, sume los números y asigne el signo al resultado.
  b. Si los signos son diferentes, compare los valores absolutos, reste el menor del mayor y asigne el signo del mayor al resultado.
- Desplace los puntos decimales para igualar los exponentes. Esto significa que si los exponentes no son iguales, el punto decimal del número con el exponente menor se mueve a la izquierda para igualar los exponentes.
- Sume o reste las **mantisas** (incluyendo la parte entera y la parte fraccionaria).
- Normalice el resultado antes de almacenarlo en la memoria.
- Verifique si hay desbordamiento.

EJEMPLO 6

Sume dos flotantes:

0 10000100 10110000000000000000000
0 10000010 01100000000000000000000

SOLUCIÓN

El exponente del primer número es 132−127, o 5. El exponente del segundo número es 130−127, o 3. Por consiguiente, los números son:

$+2^5 \times$ **1.1011**
$+2^3 \times$ **1.011**

Iguale los exponentes:

$+2^5 \times$ **1.1011**
$+2^5 \times$ **0.01011**
$+2^5 \times$ **10.00001**

Ahora normalice el resultado: ■

$+2^6 \times 1.000001$

El flotante almacenado en la computadora es: ■

0 10000101 00000100000000000000000

4.2 OPERACIONES LÓGICAS

Un solo bit puede ser ya sea 0 o 1. Usted puede interpretar el 0 como el valor lógico *falso* y el 1 como el valor lógico *verdadero*. De esta manera, un bit almacenado en la memoria de una computadora puede representar un valor lógico que es ya sea falso o verdadero.

Si un bit se interpreta como un valor lógico, entonces pueden aplicarse las operaciones lógicas a este bit. Una **operación lógica** acepta uno o dos bits para crear sólo un bit. Si la operación se aplica sólo a una entrada, es una **operación unaria**. Si ésta se aplica a dos bits es una **operación binaria** (figura 4.3).

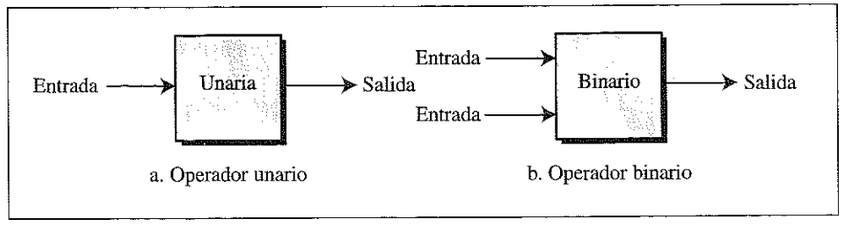


Figura 4.3 Operaciones unarias y binarias

En esta sección se analiza un operador unario y tres operadores binarios, los cuales se muestran en la figura 4.4.

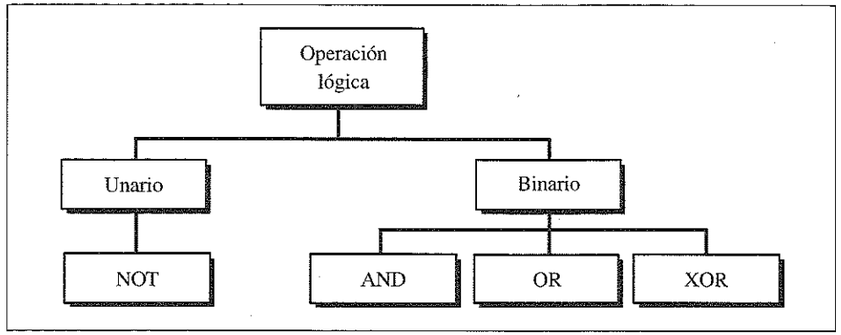
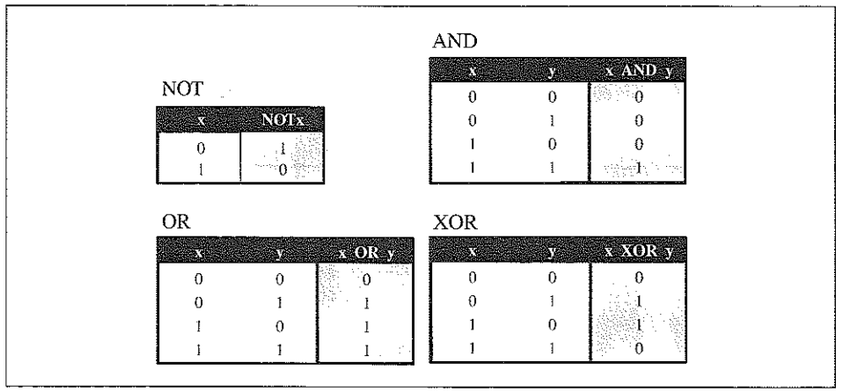


Figura 4.4 Operaciones lógicas

TABLAS DE VERDAD

Una manera de mostrar el resultado de una operación lógica es con una tabla de verdad. Una **tabla de verdad** lista todas las combinaciones de entrada posibles con su salida correspondiente. En el caso de un operador unario como NOT, hay dos posibilidades de salida. En el caso de un operador binario hay cuatro posibilidades de salida. La figura 4.5 muestra las tablas de verdad para NOT, AND, OR y XOR.



NOT

x	NOTx
0	1
1	0

AND

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Figura 4.5 Tablas de verdad

OPERADOR UNARIO

El único **operador unario** que analizamos es el operador NOT.

Operador NOT

El operador NOT tiene una entrada (un patrón con bits). Este operador invierte los bits, es decir, cambia el 0 a 1 y el 1 a 0. La figura 4.6 muestra el resultado de aplicar el operador NOT a un patrón con bits. En ella, aparece el símbolo NOT convencional que convierte cada bit y un cuadro NOT que es una operación aplicada a todo el patrón. Observe que una tabla de verdad se aplica a cada bit individual.

EJEMPLO 7

Use el operador NOT en el patrón con bits 10011000.

SOLUCIÓN

La solución es:

Objetivo	**10011000**	**NOT**
Resultado	**01100111**	

■

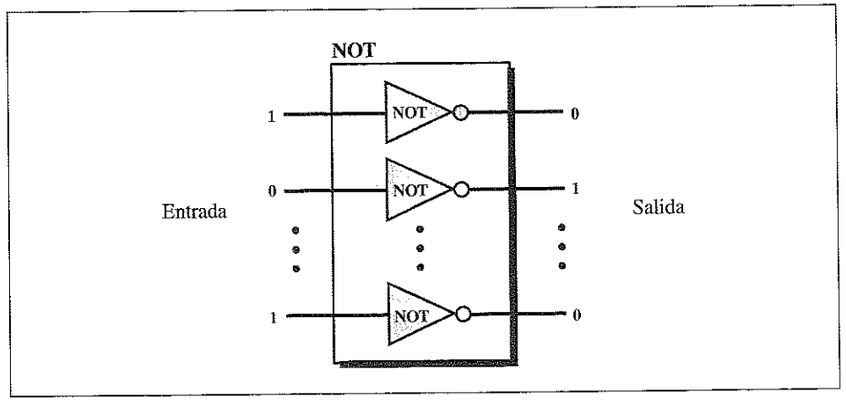


Figura 4.6 Operador NOT

OPERADORES BINARIOS

En esta sección se analizan los **operadores binarios**: AND, OR y XOR.

Operador AND

El **operador AND** es un operador binario. Este operador toma dos entradas (dos patrones con bits) y crea una salida (patrón con bits). AND aplica la tabla de verdad a un par con bits, uno de cada entrada y crea la salida correspondiente (figura 4.7). Se utilizó el símbolo AND que aplica disyunción a cada par con bits y un cuadro AND que es una operación que se aplica a cada par de patrones. Para cada par con bits de entrada, el resultado es 1 si y sólo si ambos bits son 1; de lo contrario, es 0.

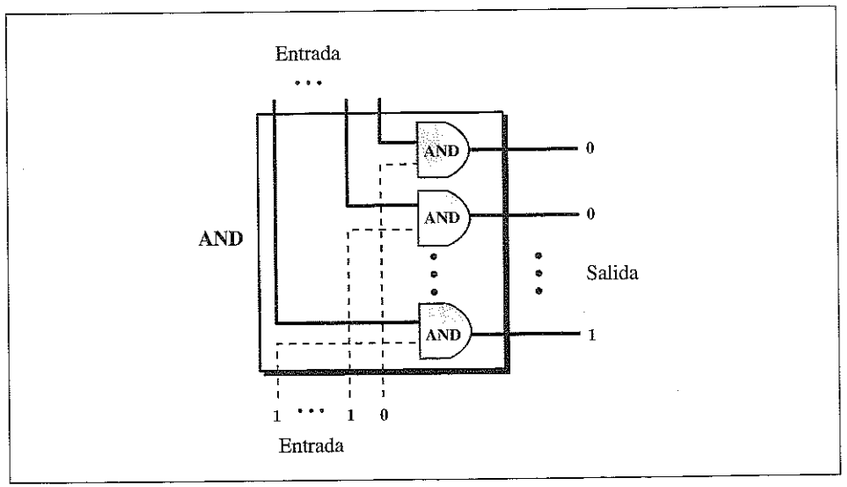


Figura 4.7 Operador AND

EJEMPLO 8

Use el operador AND en el patrón con bits 10011000 y 00110101.

SOLUCIÓN

La solución es:

Patrón 1	**10011000**	**AND**
Patrón 2	**00110101**	
Resultado	**00010000**	

■

Regla inherente del operador AND Un aspecto interesante sobre el operador AND es que si un bit en una entrada es 0, usted no tiene que revisar el bit correspondiente en la otra entrada; puede concluir rápidamente que el resultado es 0. Este hecho se define como la regla inherente del operador AND (figura 4.8).

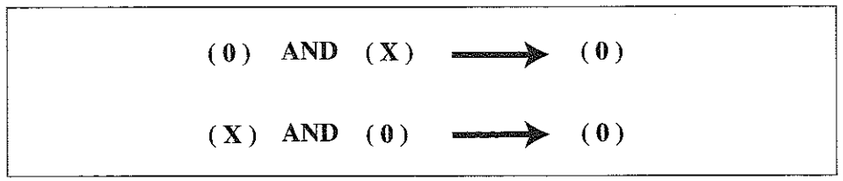


Figura 4.8 Regla inherente del operador AND

Operador OR

El **operador OR** es un operador binario. Este operador toma dos entradas (dos patrones con bits) y crea una salida (patrón de bis). OR aplica la tabla de verdad a un par con bits, uno de cada entrada, y crea la salida correspondiente (figura 4.9). En la figura se utilizó el símbolo OR convencional que incluye cada par con bits y un cuadro OR que es una operación aplicada al par de patrones. Para cada par con bits de entrada, el resultado es 0 si y sólo si ambos bits son 0; de lo contrario, es 1.

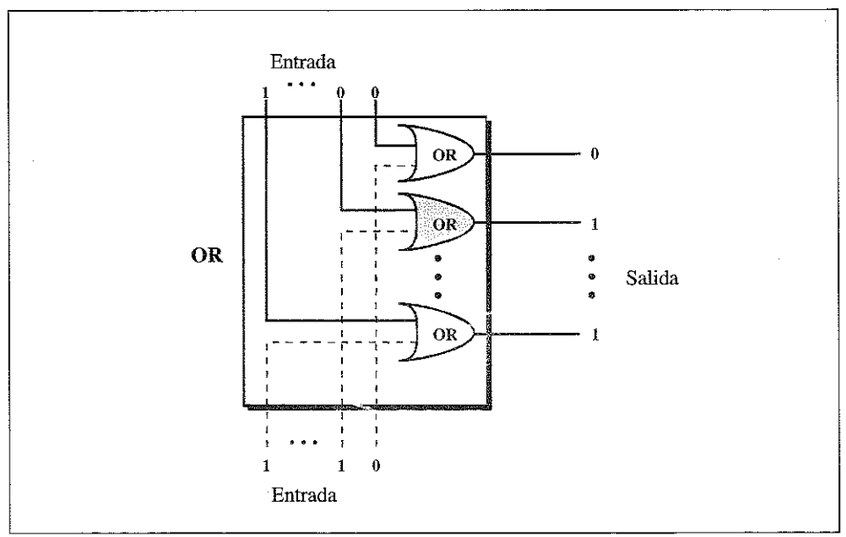


Figura 4.9 Operador OR

EJEMPLO 9

Utilice el operador OR en los patrones con bits 10011000 y 00110101.

SOLUCIÓN

La solución es:

Patrón 1	**10011000**	**OR**
Patrón 2	**00110101**	
Resultado	**10111101**	

■

Regla inherente del operador OR Un aspecto interesante sobre el operador OR es que si un bit en una entrada es 1, usted no tiene que revisar el bit correspondiente en la otra entrada; puede concluir rápidamente que el resultado es 1. Definimos este hecho como la regla inherente del operador OR (figura 4.10).

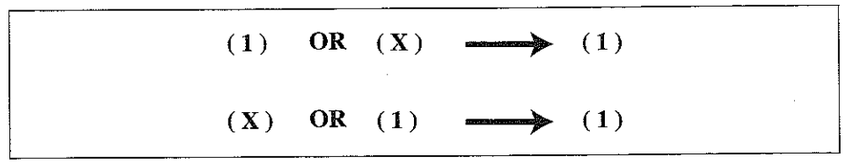


Figura 4.10 Regla inherente del operador OR

Operador XOR

El **operador XOR** es un operador binario, que toma dos entradas (dos patrones con bits) y crea una salida (un patrón con bits). XOR aplica la tabla de verdad a un par con bits, uno de cada entrada, y crea la salida correspondiente (figura 4.11). En la figura se utilizó el símbolo XOR convencional que aplica exclusión a cada par con bits y un cuadro XOR que es una operación aplicada al par de patrones. Para cada par con bits de entrada el resultado es 0 si y sólo si ambos bits son iguales; de lo contrario es 1.

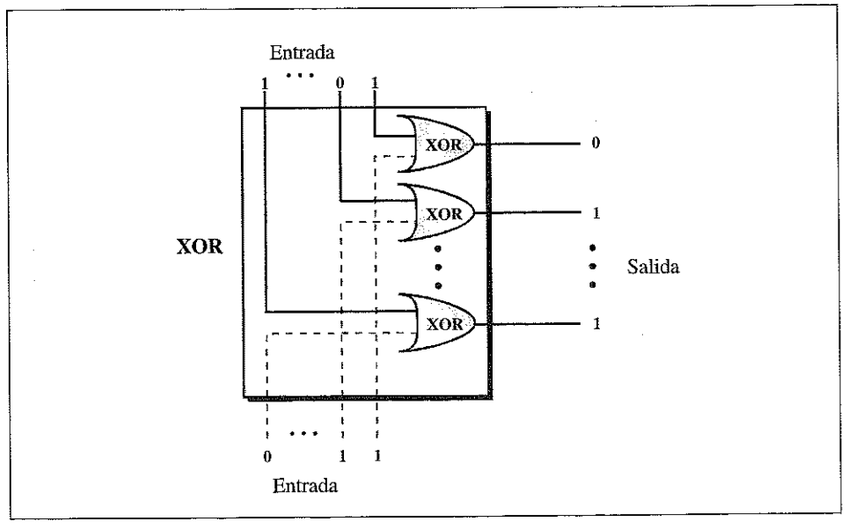


Figura 4.11 Operador XOR

EJEMPLO 10

Use el operador XOR en los patrones con bits 10011000 y 00110101.

SOLUCIÓN

La solución es la siguiente:

Patrón 1	10011000	**XOR**
Patrón 2	00110101	
Resultado	**10101101**	

■

Regla inherente del operador XOR Un aspecto interesante sobre el operador XOR es que si un bit en una entrada es 1, el resultado es el inverso del bit correspondiente en la otra entrada. Definimos este hecho como la regla inherente del operador XOR (figura 4.12).

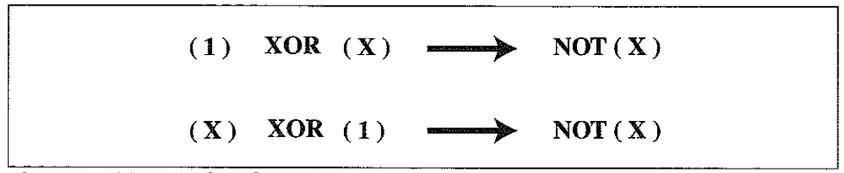


Figura 4.12 Regla inherente del operador XOR

APLICACIONES

Las tres operaciones binarias lógicas pueden utilizarse para modificar un patrón con bits. Pueden hacer disyunción, conjunción o invertir bits específicos. El patrón con bits que se va a modificar se aplica disyunción (AND), conjunción (OR) o exclusión (XOR) con un segundo patrón con bits, el cual se conoce como **máscara**. La máscara se usa para modificar otro patrón con bits (figura 4.18).

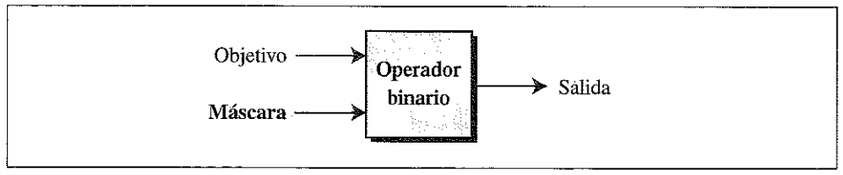


Figura 4.13 Máscara

Apagar bits específicos

Una de las aplicaciones del operador AND es **apagar (forzar a 0)** bits específicos en un patrón con bits. Para hacerlo, use una máscara de apagado con la misma longitud con bits. Las reglas para construir una máscara de indeterminación pueden resumirse como sigue:

1. Para apagar un bit en el patrón con bits objetivo, utilice 0 para el valor correspondiente en la máscara.
2. Para dejar un bit sin cambiar en el patrón con bits objetivo, use 1 para el bit correspondiente en la máscara.

Para comprender por qué operan estas reglas, remítase a la regla inherente del operador AND en la figura 4.8.

Por ejemplo, suponga que quiere apagar los cinco bits en el extremo izquierdo de un patrón de ocho bits. La máscara debe tener cinco 0 a la izquierda, seguidos por tres 1 (figura 4.14).

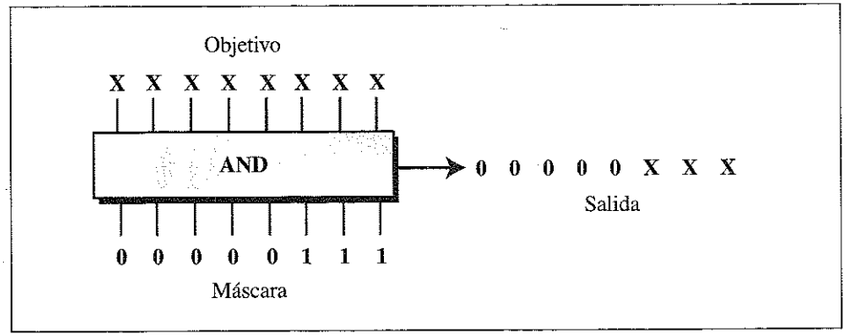


Figura 4.14 Ejemplo de apagado con bits específicos

EJEMPLO 11

Use una máscara para apagar (**borrar**) los cinco bits en el extremo izquierdo de un patrón. Pruebe la máscara con el patrón 10100110.

SOLUCIÓN

La máscara es 00000111. El resultado de aplicar la máscara es:

Objetivo	10100110	**AND**
Máscara	00000111	
Resultado	**00000**110	

■

EJEMPLO 12

Imagine una planta de energía que bombea agua a una ciudad usando ocho bombas. Los estados de las bombas (encendido o apagado) pueden representarse mediante un patrón de ocho bits. Por ejemplo, el patrón 11000111 muestra que las bombas 1 a 3 (a partir de la derecha), 7 y 8 están encendidas, mientras que las bombas 4, 5 y 6 están apagadas. Ahora suponga que la bomba 7 se apaga. ¿Cómo puede mostrar esta situación una máscara?

SOLUCIÓN

Se utiliza la máscara 10111111 operada AND con el patrón objetivo. El único bit 0 (bit 7) en la máscara apaga el séptimo bit en el objetivo.

Situación anterior	**11000111**	**AND**
Máscara	**10111111**	
Nueva situación	**10000111**	

■

Encender bits específicos

Una de las aplicaciones del operador OR es **encender (forzar a 1)** bits específicos en un patrón con bits. Para hacerlo, utilice una máscara de determinación con la misma longitud con bits. Las reglas para construir una máscara de encendido pueden resumirse como sigue:

1. Para encender un bit en el patrón con bits objetivo, use un 1 para el bit correspondiente en la máscara.
2. Para dejar un bit en el patrón con bits objetivo sin cambiar, utilice un 0 para el bit correspondiente en la máscara.

Para comprender por qué estas reglas operan, remítase a la regla inherente del operador OR en la figura 4.10.

Por ejemplo, suponga que quiere encender los cinco bits en el extremo izquierdo de un patrón de ocho bits. La máscara debe tener cinco 1 a la izquierda seguidos por tres 0 (figura 4.15).

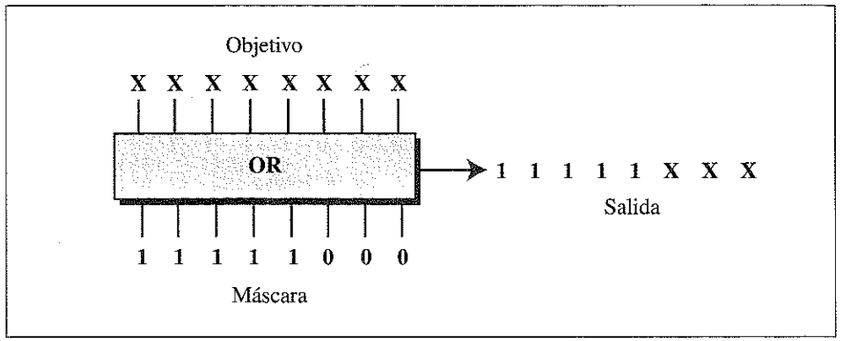


Figura 4.15 Ejemplo de determinación con bits específicos

EJEMPLO 13

Use una máscara para encender los cinco bits en el extremo izquierdo de un patrón. Pruebe la máscara con el patrón 10100110.

SOLUCIÓN

La máscara es 11111000. El resultado de aplicar la máscara es:

Objetivo	10100110	**OR**
Máscara	11111000	
Resultado	11111**1**10	

■

EJEMPLO 14

Tomando el ejemplo de la planta de energía, ¿cómo se podría utilizar una máscara para mostrar que la bomba 6 ahora está encendida?

SOLUCIÓN

Se utiliza la máscara 00100000 y el operador OR con el patrón objetivo. El único bit 1 (bit 6) en la máscara enciende el sexto bit en el objetivo.

Situación previa	10000111	**OR**
Máscara	00100000	
Nueva situación	10100111	

■

Complementar bits específicos

Una de las aplicaciones de XOR es **complementar** bits, lo cual significa cambiar el valor con bits específicos de 0 a 1 y viceversa. Las reglas para construir una máscara XOR pueden resumirse como sigue:

1. Para complementar un bit en el patrón con bits objetivo, utilice un 1 para el bit correspondiente en la máscara.
2. Para dejar un bit en el patrón con bits objetivo sin cambiar, utilice un 0 para el bit correspondiente en la máscara.

Para comprender por qué operan estas reglas, remítase a la regla inherente del operador XOR en la figura 4.12.

Por ejemplo, para complementar los cinco bits en el extremo izquierdo de un patrón de ocho bits, haga una máscara que comience con cinco 1 a la izquierda seguidos por tres 0 (figura 4.16).

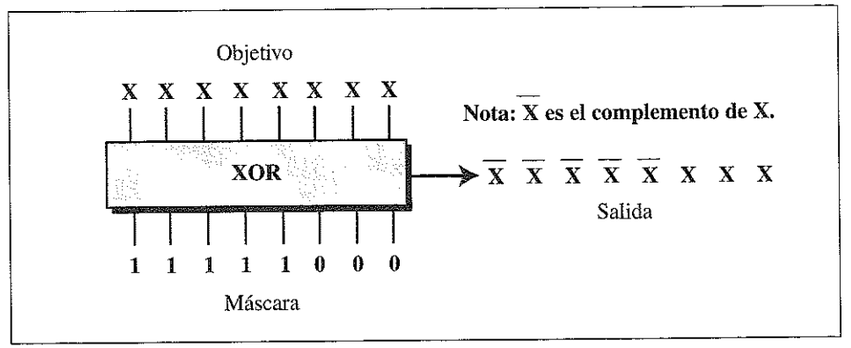


Figura 4.16 Ejemplo de reversión con bits específicos

EJEMPLO 15

Utilice una máscara para revertir los cinco bits en el extremo izquierdo de un patrón. Pruebe la máscara con el patrón 10100110.

SOLUCIÓN

La máscara es 11111000. El resultado de aplicar la máscara es:

Objetivo	**10100110**	**XOR**
Máscara	**11111000**	
Resultado	**01011110**	

■

4.3 OPERACIONES DE DESPLAZAMIENTO

Otra operación común en patrones con bits es la operación de desplazamiento. Un patrón con bits puede ser desplazado a la derecha o a la izquierda. La operación de desplazamiento a la derecha descarta el bit en el extremo derecho, desplaza cada bit hacia la derecha e inserta 0 como el bit en el extremo izquierdo. La operación de desplazamiento a la izquierda descarta el bit en el extremo izquierdo, desplaza cada bit hacia la izquierda e inserta 0 como el bit en el extremo derecho. La figura 4.17 muestra esas dos operaciones en patrones de ocho bits donde a, b, . . ., h representan bits individuales.

Se debe tener cuidado con las operaciones de desplazamiento si el patrón representa un número con signo. Una operación de desplazamiento puede cambiar el bit en el extremo izquierdo del patrón, el cual en un número con signo representa el signo del número. Las operaciones de desplazamiento sólo pueden utilizarse cuando un patrón representa un número sin signo.

EJEMPLO 16

Muestre cómo se puede dividir o multiplicar un número por 2 usando operaciones de desplazamiento.

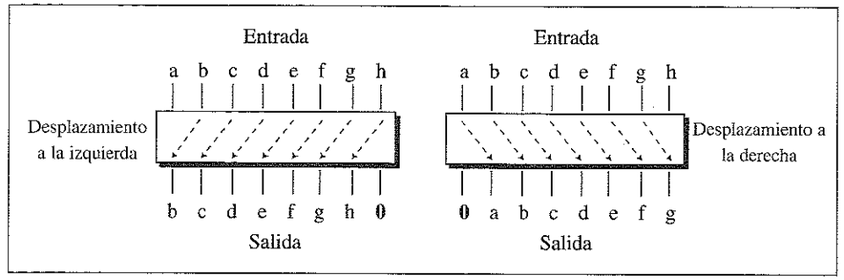


Figura 4.17 Operaciones de desplazamiento

SOLUCIÓN

Si un patrón con bits representa un número sin signo, una operación de desplazamiento a la derecha divide el número por 2 (división de enteros). El patrón 00111011 representa 59. Cuando el número se desplaza a la derecha, se obtiene 00011101, el cual es 29. Si el número original (59) se desplaza a la izquierda, se obtiene 01110110, el cual es 118. ■

EJEMPLO 17

Utilice una combinación de operaciones lógicas y de desplazamiento para encontrar el valor (0 o 1) del cuarto bit (a partir de la derecha) en un patrón.

SOLUCIÓN

Se utiliza la máscara 00001000 operada AND con el objetivo para mantener el cuarto bit y borrar el resto de los bits.

```
a b c d e f g h   AND
0 0 0 0 1 0 0 0
---------------
0 0 0 0 e 0 0 0
```

Para tener acceso al valor del cuarto bit (e), se desplaza el nuevo patrón tres posiciones a la derecha de modo que el bit específico se coloque en la posición del extremo derecho.

0000e000 ⟶ 00000e00 ⟶ 000000e0 ⟶ 0000000e

Ahora es muy fácil probar el valor del nuevo patrón como un entero sin signo. Si el valor es 1, el bit original fue 1; si el valor es 0, el bit original fue 0. ■

4.4 TÉRMINOS CLAVE

acarreo
apagar
borrar
complementar
complemento a dos
desbordamiento
encender
forzar a 0
forzar a 1
mantisa
máscara
número de punto flotante
operador AND
operación aritmética
operación binaria
operación lógica
operación unaria
operador binario
operador NOT
operador OR
operador unario
operador XOR
tabla de verdad

4.5 RESUMEN

- Usted puede realizar operaciones aritméticas o lógicas sobre bits.
- La mayoría de las computadoras utiliza el método de representación de enteros del complemento a dos.
- Si hay un acarreo después de la suma de los dígitos en el extremo izquierdo, el acarreo se descarta.
- Para restar en el complemento a dos sólo se complementa el número a ser restado y se suma.
- Los números a ser sumados deben estar dentro del intervalo definido por la asignación con bits.
- El término *desbordamiento* describe una condición en la cual un número no está dentro del intervalo definido por la asignación con bits.
- La operación lógica sobre bits puede ser unaria (una entrada) o binaria (dos entradas).
- El operador NOT unario invierte su entrada.
- El resultado de la operación binaria AND es verdadero sólo si ambas entradas son verdaderas.
- El resultado de la operación binaria OR es falso sólo si ambas entradas son falsas.
- El resultado de la operación binaria XOR es falso si ambas entradas son iguales.
- Una máscara es un patrón con bits que se aplica a un patrón con bits objetivo para lograr un resultado específico.
- Para apagar (borrar) un bit en un patrón con bits objetivo, se pone el bit de la máscara correspondiente en 0 y se utiliza el operador AND.
- Para encender un bit en un patrón con bits objetivo, se pone el bit de la máscara correspondiente en 1 y se utiliza el operador OR.
- Para complementar un bit en un patrón con bits objetivo, se pone el bit de la máscara correspondiente en 1 y se utiliza el operador XOR.

4.6 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuál es la diferencia entre una operación aritmética y una operación lógica?
2. ¿Cómo se relaciona la multiplicación con la suma? Dé un ejemplo.
3. ¿Qué sucede con un acarreo de la columna en el extremo izquierdo en la suma final?
4. ¿Puede *N*, la asignación con bits, ser igual a 1? ¿Por qué sí o por qué no?
5. ¿Cuál es la definición del término *desbordamiento*?
6. En la suma de números de punto flotante, ¿cómo se ajusta la representación de números con diferentes exponentes?
7. ¿Cuál es la diferencia entre una operación unaria y una operación binaria?
8. Menciona cuáles son las operaciones binarias lógicas.
9. ¿Qué es una tabla de verdad?
10. ¿Qué hace el operador NOT?
11. ¿Cuándo es verdadero el resultado de una operación AND?
12. ¿Cuándo es verdadero el resultado de una operación OR?
13. ¿Cuándo es verdadero el resultado de una operación XOR?
14. ¿Cuál es la regla inherente del operador AND?
15. ¿Cuál es la regla inherente del operador OR?
16. ¿Cuál es la regla inherente del operador XOR?
17. ¿Qué operación binaria puede usarse para encender bits? ¿Qué patrón con bits debe tener la máscara?
18. ¿Qué operación binaria puede usarse para apagar bits? ¿Qué patrón con bits debe tener la máscara?
19. ¿Qué operación binaria puede usarse para revertir bits? ¿Qué patrón con bits debe tener la máscara?

PREGUNTAS DE OPCIÓN MÚLTIPLE

20. _____ es una operación aritmética con bits.
    a. El OR exclusivo
    b. El NOT unario
    c. La resta
    d. Todos los anteriores
21. _____ es un operador lógico con bits.
    a. El OR exclusivo
    b. El NOT unario
    c. El AND binario
    d. Todos los anteriores

22. El método de representación de enteros de _____ es el método más común para almacenar enteros en la memoria de la computadora.
- **a.** signo y magnitud
- **b.** complemento a uno
- **c.** complemento a dos
- **d.** enteros no asignados

23. En la suma del complemento a dos, si hay un acarreo final después de la suma de la columna en el extremo izquierdo, _____.
- **a.** éste se suma a la columna en el extremo derecho
- **b.** éste se suma a la columna en el extremo izquierdo
- **c.** éste se descarta
- **d.** aumenta la longitud con bits

24. Para una asignación de ocho bits, el número decimal más pequeño que puede representarse en la forma de complemento a dos es _____.
- **a.** −8
- **b.** −127
- **c.** −128
- **d.** −256

25. Para una asignación de ocho bits, el número decimal más grande que puede representarse en la forma de complemento a dos es _____.
- **a.** 8
- **b.** 127
- **c.** 128
- **d.** 256

26. En la representación del complemento a dos con una asignación de cuatro bits, usted obtiene _____ cuando le suma 1 a 7.
- **a.** 8
- **b.** 1
- **c.** −7
- **d.** −8

27. En la representación del complemento a dos con una asignación de cuatro bits, usted obtiene _____ cuando le suma 5 a 5.
- **a.** −5
- **b.** −6
- **c.** −7
- **d.** 10

28. Si el exponente en Excess_127 es el binario 10000101, el exponente en decimal es _____.
- **a.** 6
- **b.** 7
- **c.** 8
- **d.** 9

29. Si usted está sumando dos números, uno de los cuales tiene un valor de exponente de 7 y el otro un valor exponente de 9, necesita desplazar el punto decimal del número menor _____.
- **a.** un lugar a la izquierda
- **b.** un lugar a la derecha
- **c.** dos lugares a la izquierda
- **d.** dos lugares a la derecha

30. El operador binario _____ toma dos entradas para producir una salida.
- **a.** AND
- **b.** OR
- **c.** XOR
- **d.** todos los anteriores

31. El operador unario _____ invierte su única entrada.
- **a.** AND
- **b.** OR
- **c.** NOT
- **d.** XOR

32. Para el operador binario _____, si la entrada es dos 0 la salida es un 0.
- **a.** AND
- **b.** OR
- **c.** XOR
- **d.** todos los anteriores

33. Para el operador binario _____, si la entrada es dos 1 la salida es 0.
- **a.** AND
- **b.** OR
- **c.** XOR
- **d.** todos los anteriores

34. Para la operación binaria AND, sólo una entrada de _____ da una salida de 1.
- **a.** dos 0
- **b.** dos 1
- **c.** un 0 y un 1
- **d.** cualquiera de los anteriores

35. Para la operación binaria OR, sólo una entrada de _____ da una salida de 0.
- **a.** dos 0
- **b.** dos 1
- **c.** un 0 y un 1
- **d.** cualquiera de los anteriores

36. Usted utiliza un patrón con bits llamado _____ para modificar otro patrón con bits.
- **a.** máscara
- **b.** acarreo
- **c.** flotante
- **d.** byte

37. Para complementar todos los bits de un patrón con bits, hace una máscara de todos los bits en 1 y luego _____ el patrón con bits y la máscara.
- **a.** opera AND
- **b.** opera OR
- **c.** opera XOR
- **d.** opera NOT

38. Para apagar (forzar a 0) todos los bits de un patrón con bits, hace una máscara de todos los los bits en 0 y luego _____ el patrón con bits y la máscara.
- **a.** opera AND
- **b.** opera OR
- **c.** opera XOR
- **d.** opera NOT

39. Para encender (forzar a 1) todos los bits de un patrón con bits, hace una máscara de todos los bits en 1 y luego _____ el patrón con bits y la máscara.
- **a.** opera AND
- **b.** opera OR
- **c.** opera XOR
- **d.** opera NOT

EJERCICIOS

40. Usando una asignación de ocho bits, primero convierta cada uno de los siguientes números a complemento a dos, realice la operación y luego convierta el resultado a decimal.
- **a.** 19 + 23
- **b.** 19 − 23
- **c.** −19 + 23
- **d.** −19 − 23

41. Usando una asignación de 16 bits, primero convierta cada uno de los siguientes números a complemento a dos, realice la operación y luego convierta el resultado a decimal.
- **a.** 161 + 1023
- **b.** 161 − 1023
- **c.** −161 + 1023
- **d.** −161 − 1023

42. ¿Cuál de las siguientes situaciones crea desbordamiento si los números y el resultado se representan en la notación de ocho bits usando complemento a dos?
- **a.** 11000010 + 00111111
- **b.** 00000010 + 00111111
- **c.** 11000010 + 11111111
- **d.** 00000010 + 11111111

43. Sin hacer realmente el problema, ¿puede decir cuál de las siguientes opciones crea desbordamiento si los números y el resultado están en notación de complemento a dos de ocho bits?
- **a.** 32 + 105
- **b.** 32 − 105
- **c.** −32 + 105
- **d.** −32 − 105

44. Muestre el resultado de las operaciones siguientes suponiendo que los números se almacenan en la representación de complemento a dos en 16 bits. Muestre el resultado en notación hexadecimal.
- **a.** x012A + x0E27
- **b.** x712A + x9E00
- **c.** x8011 + x0001
- **d.** xE12A + x9E27

45. Muestre el resultado de las siguientes operaciones de punto flotante. Primero convierta cada número a notación binaria y realice la operación; luego convierta el resultado de nuevo a notación decimal.
- **a.** 34.075 + 23.12
- **b.** −12.00067 + 451.00
- **c.** 33.677 − 0.00056
- **d.** −344.23 − 123.8902

46. Muestre el resultado de las siguientes operaciones de punto flotante. Primero convierta cada número a notación binaria y realice la operación; luego convierta el resultado de regreso a notación decimal.
- **a.** 23.125 + 12.45
- **b.** 0.234 − 7.192
- **c.** −0.345 + 45.123
- **d.** −0.234 + 5.345

47. ¿En cuál de las siguientes situaciones nunca ocurre un desbordamiento? Justifique su respuesta.
- **a.** la suma a dos enteros positivos
- **b.** la suma de un entero positivo y un entero negativo
- **c.** la resta de un entero positivo y un entero negativo
- **d.** la resta de dos enteros negativos

48. Muestre el resultado de las operaciones siguientes:
- **a.** NOT x99
- **b.** NOT xFF
- **c.** NOT x00
- **d.** NOT x01

49. Muestre el resultado de las operaciones siguientes:
- **a.** x99 AND x99
- **b.** x99 AND x00
- **c.** x99 AND xFF
- **d.** xFF AND xFF

50. Muestre el resultado de las operaciones siguientes:
- **a.** x99 OR x99
- **b.** x99 OR x00
- **c.** x99 OR xFF
- **d.** xFF OR xFF

51. Muestre el resultado de las operaciones siguientes:
- **a.** x99 XOR x99
- **b.** x99 XOR x00
- **c.** x99 XOR xFF
- **d.** xFF XOR xFF

52. Muestre el resultado de las operaciones siguientes:
- **a.** NOT (x99 OR x99)
- **b.** x99 OR (NOT x00)
- **c.** (x99 AND x33) OR (x00 AND xFF)
- **d.** (x99 OR x33) AND (x00 OR xFF)

53. Necesita apagar (forzar a 0) los cuatro bits en el extremo izquierdo de un patrón. Muestre la máscara y la operación.

54. Necesita encender (forzar a 1) los cuatro bits en el extremo derecho de un patrón. Muestre la máscara y la operación.

55. Necesita complementar los tres bits en el extremo derecho y los dos bits en el extremo izquierdo de un patrón. Muestre la máscara y la operación.

56. Necesita apagar los tres bits en el extremo izquierdo y encender los dos bits en el extremo derecho de un patrón. Muestre las máscaras y las operaciones.

57. Use la operación de desplazamiento para dividir un número sin signo por 4.

58. Use la operación de desplazamiento para multiplicar un número sin signo por 8.

59. Utilice una combinación de operaciones lógicas y de desplazamiento para extraer el cuarto y el quinto bits de un número sin signo.

II Hardware de computadora

Organización de la computadora

En este capítulo se estudia la organización de una computadora independiente. En el capítulo siguiente mostraremos cómo conectar computadoras independientes para formar una red y cómo conectar redes para formar una red de redes (o internet).

Las partes que forman una computadora se pueden dividir en tres categorías o subsistemas generales: el CPU (unidad central de procesamiento), la memoria principal y los subsistemas de entrada/salida (E/S). Las siguientes tres secciones tratan sobre estos subsistemas y la manera como se conectan para formar una computadora independiente. La figura 5.1 muestra los tres subsistemas de una computadora independiente.

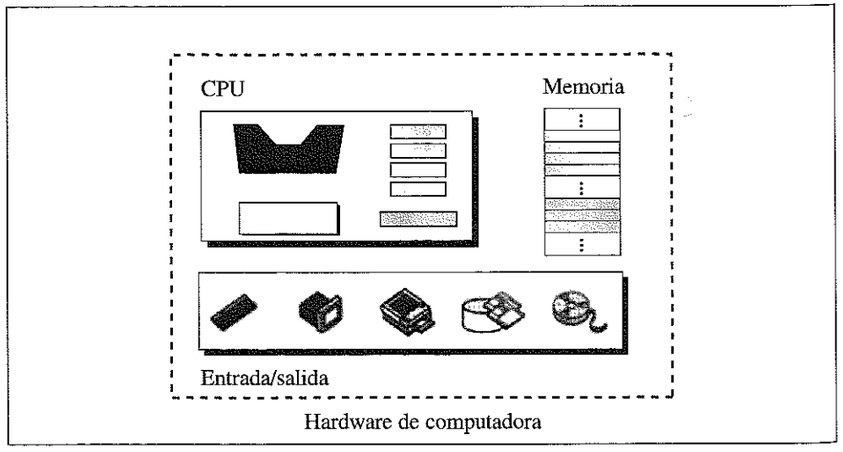


Figura 5.1 Hardware de computadora

5.1 UNIDAD CENTRAL DE PROCESAMIENTO (CPU)

La **unidad central de procesamiento** (**CPU**: *central process unit*) realiza operaciones con los datos. Tiene tres partes: una unidad lógica aritmética (ALU: *arithmetic logic unit*), una unidad de control y una serie de registros (figura 5.2).

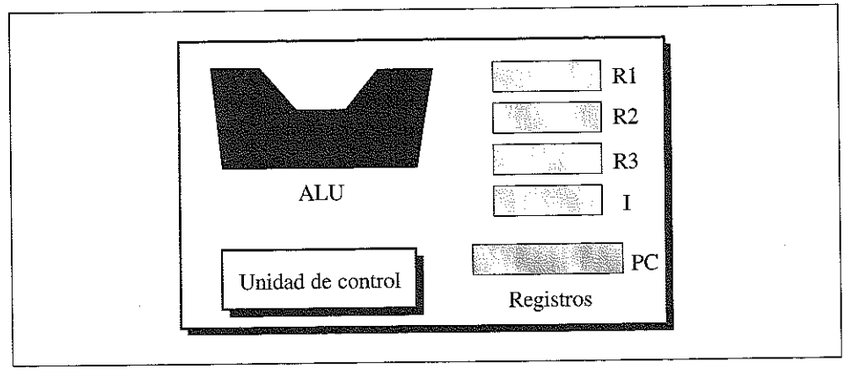


Figura 5.2 CPU

UNIDAD LÓGICA ARITMÉTICA (ALU)

La **unidad lógica aritmética** (**ALU**) realiza operaciones aritméticas y lógicas.

Operación aritmética

Las operaciones unarias más simples son el incremento (sumar 1) y el decremento (restar 1). Las operaciones binarias más simples son la suma, la resta, la multiplicación y la división. La unidad de control, como se verá en breve, es responsable de la selección de una de estas operaciones.

Operación lógica

La operación unaria lógica más simple es la operación NOT. Las operaciones binarias lógicas más simples son AND, OR y XOR. Estudiamos estas operaciones en el capítulo 4. La unidad de control es responsable de la selección de una de estas operaciones.

REGISTROS

Los **registros** son localidades de almacenamiento independientes que alojan los datos temporalmente. Se necesitan varios registros para facilitar la operación del CPU. Algunos de estos registros están en la figura 5.2.

Registros de datos

En el pasado, las computadoras sólo tenían un registro para alojar por turnos uno de los datos de entrada (el otro dato de entrada venía directamente de la memoria) o el resultado. Actualmente las computadoras utilizan docenas de registros dentro del CPU para acelerar las operaciones debido a que cada vez más las operaciones complejas se realizan usando hardware (en vez de usar software) y se requieren varios registros para mantener los resultados intermedios. Por simplicidad, sólo mostramos tres registros generales: dos para entrada de datos y uno para salida de datos (registros R1, R2 y R3) en la figura 5.2.

Registro de instrucción

Hoy día, las computadoras almacenan en la memoria no sólo datos sino también el programa correspondiente. El CPU es responsable de buscar las instrucciones, una por una, desde la memoria, almacenarlas en el **registro de instrucciones** (registro I en la figura 5.2), interpretarlas y ejecutarlas. Este tema se estudiará en una sección posterior.

Contador de programa

Otro registro común en el CPU es el **contador de programa** (registro PC en la figura 5.2). El contador de programa hace un seguimiento de la instrucción que se ejecuta actualmente. Después de la ejecución de la instrucción, el contador se incrementa para apuntar a la dirección de la siguiente instrucción en la memoria.

UNIDAD DE CONTROL

El tercer componente de cualquier CPU es la unidad de control. La **unidad de control** es como la parte del cerebro humano que controla la operación de cada parte del cuerpo. El control se logra a través de líneas de control que pueden estar activas o inactivas. Por ejemplo, una ALU simple necesita realizar tal vez diez operaciones diferentes. Para especificar estas operaciones se necesitan cuatro líneas de control desde la unidad de control al ALU. Cuatro líneas de control pueden definir 16 situaciones diferentes (2^4), diez de las cuales pueden usarse para operaciones aritméticas y lógicas. El resto puede utilizarse para otros propósitos. Puede designar una línea de control inactiva como 0 y una línea de control activa como 1; los estados de las líneas de control pueden designarse como 0000, 0001, 0010... 1111. Se puede definir 0000 (todas las líneas de control inactivas) para denotar ninguna operación, 0001 para denotar incremento, 0010 para denotar decremento y así por el estilo.

5.2 MEMORIA PRINCIPAL

La **memoria principal** es otro subsistema en una computadora (figura 5.3). Es una colección de localidades de almacenamiento, cada una con un identificador único conocido como dirección. Los datos se transfieren hacia y desde la memoria en grupos de bits llamados palabras. Una palabra puede ser un grupo de 8 bits, 16 bits, 32 bits o en ocasiones 64 bits. Si la palabra es de ocho bits, se hace referencia a ella como un byte. El término *byte* es tan común en las ciencias de la computación, que a veces se hace referencia a una palabra de 16 bits como una palabra de 2 bytes, o a una palabra de 32 bits como una palabra de 4 bytes.

ESPACIO DE DIRECCIONAMIENTO

Para tener acceso a una palabra en la memoria se requiere un identificador. Aunque los programadores utilizan un nombre para identificar una palabra (o una colección de palabras), en el nivel del hardware cada palabra se identifica por una dirección. El número total de localidades únicas identificables en la memoria se llama **espacio de direccionamiento**. Por ejemplo, una memoria con 64 kilobytes y un tamaño de palabra de un byte tienen un espacio de direccionamiento que varía de 0 a 65 535.

La tabla 5.1 muestra las unidades usadas para referirse a la memoria. Observe que la terminología es engañosa; aproxima el número de bytes en potencias de 10, pero el número de bytes real está en potencias de 2. Las unidades en potencias de 2 facilitan el direccionamiento.

Unidad	Número exacto de bytes	Aproximación
kilobyte	2^{10} (1 024) bytes	10^3 bytes
megabyte	2^{20} (1 048 576) bytes	10^6 bytes
gigabyte	2^{30} (1 073 741 824) bytes	10^9 bytes
terabyte	2^{40} (1 024) bytes	10^{12} bytes
petabyte	2^{50} (1 024) bytes	10^{15} bytes
exabyte	2^{60} (1 024) bytes	10^{18} bytes

Tabla 5.1 Unidades de memoria

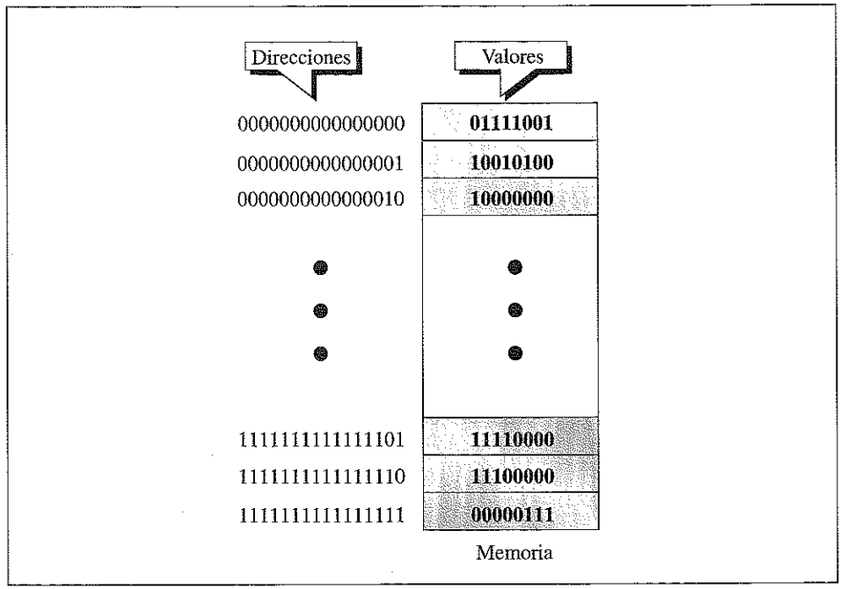


Figura 5.3 Memoria principal

Direcciones como patrones de bits

Puesto que las computadoras operan mediante el almacenamiento de números como **patrones de bits**, la dirección en sí misma también se representa como un patrón de bits. Así que si una computadora tiene 64 kilobytes (2^{16}) de memoria con un tamaño de palabra de 1 byte, entonces para definir una dirección, se necesita un patrón de 16 bits. Recuerde del capítulo 3 que las direcciones pueden representarse como enteros sin signo (usted no tiene direcciones negativas). En otras palabras, la primera localidad se refiere como una dirección 0000000000000000 (dirección 0) y la última locación se refiere como una dirección 1111111111111111 (dirección 65535). En general, si una computadora tiene N palabras de memoria, se necesita un entero sin signo con un tamaño de $\log_2 N$ bits para referirse a cada localidad de memoria.

Las direcciones de memoria se definen usando enteros binarios sin signo.

EJEMPLO 1

Una computadora tiene 32 MB (megabytes) de memoria. ¿Cuántos bits se necesitan para asignar una dirección a cualquier byte individual en la memoria?

SOLUCIÓN

El espacio de direccionamiento de memoria es 32 MB o 2^{25} ($2^5 \times 2^{20}$). Esto significa que se necesitan $\log_2 2^{25}$ o 25 bits para asignar una dirección a cada byte. ■

EJEMPLO 2

Una computadora tiene 128 MB de memoria. Cada palabra en esta computadora tiene ocho bytes. ¿Cuántos bits se necesitan para asignar una dirección a cualquier palabra individual en la memoria?

SOLUCIÓN

El espacio de direccionamiento de memoria es 128 MB, lo cual significa 2^{27}. Sin embargo, cada palabra es de ocho (2^3) bytes, lo cual significa que usted tiene 2^{24} palabras. Esto significa que se necesitan $\log_2 2^{24}$, o 24 bits, para asignar una dirección a cada palabra. ■

TIPOS DE MEMORIA

Hay dos tipos de memoria disponibles: RAM y ROM.

RAM

La **memoria de acceso aleatorio** (**RAM**: *random access memory*) constituye la mayor parte de la memoria principal en una computadora. El término es confuso debido a que también se puede tener acceso a la ROM en forma aleatoria. Lo que distingue a la RAM de la ROM es que el usuario puede leer de y escribir en la RAM. El usuario puede escribir algo en la RAM y posteriormente borrarlo simplemente al sobrescribirlo. Otra característica de la RAM es que es volátil; la información (programa o datos) se borra si el sistema se apaga. En otras palabras, toda la información en la RAM se borra si usted apaga la computadora o si hay un apagón. La tecnología de la RAM se divide en dos categorías generales: SRAM y DRAM.

SRAM La tecnología de **RAM estática** (**SRAM**: *static RAM*) utiliza compuertas flip-flop (compuertas con dos estados, 0 y 1) para almacenar datos. Las compuertas alojan su estado (0 o 1), lo cual significa que los datos se almacenan mientras haya suministro de corriente; no hay necesidad de refrescar. La SRAM es rápida pero costosa.

DRAM La tecnología de **RAM dinámica** (**DRAM**: *dynamic RAM*) utiliza capacitores. Si el capacitor está cargado, el estado es 1; si está descargado el estado es 0. Como un capacitor pierde un poco de su carga con el tiempo, las celdas de la memoria necesitan refrescarse periódicamente. Las DRAM son lentas pero económicas.

ROM

El contenido de la **memoria de sólo lectura** (**ROM**: *read-only memory*) es escrito por el fabricante; el usuario puede leer la ROM pero no escribir en ella. Su ventaja es que no es volátil; su contenido no se borra si usted apaga la computadora. Normalmente la utilizan programas o datos que no deben ser borrados o cambiados aun cuando usted apague la computadora. Por ejemplo, algunas computadoras vienen con una ROM que aloja el programa de arranque que se ejecuta cuando usted enciende la computadora.

PROM Una variación de la ROM es la **memoria de sólo lectura programable** (**PROM**: *programmable read-only memory*). Este tipo de memoria está en blanco cuando la computadora se empaca. El usuario de la computadora, con algún equipo especial, puede almacenar programas en ella. Cuando esto sucede, esta memoria se comporta como la ROM y no puede sobrescribirse. Esto permite que el usuario de la computadora almacene programas específicos en la PROM.

EPROM Una variación de la PROM es la **memoria de sólo lectura programable y borrable** (**EPROM**: *erasable programmable read-only memory*). El usuario puede programar esta memoria, pero es posible borrarla con un dispositivo especial que utiliza luz ultravioleta. Para borrar la memoria EPROM se requiere quitarla físicamente y volver a instalarla.

EEPROM Una variación de la EPROM es la **memoria de sólo lectura programable y borrable electrónicamente** (**EEPROM**: *electronically erasable programmable read-only memory*). Esta memoria puede programarse y borrarse usando impulsos electrónicos sin quitarla físicamente de la computadora.

JERARQUÍA DE LA MEMORIA

Los usuarios necesitan computadoras con mucha memoria, especialmente con memoria que sea muy rápida y muy barata. No siempre es posible satisfacer esta demanda. La memoria muy rápida por lo general no es barata. Se requiere hacer un compromiso. La solución son niveles jerárquicos de memoria (figura 5.4). La jerarquía se basa en lo siguiente:

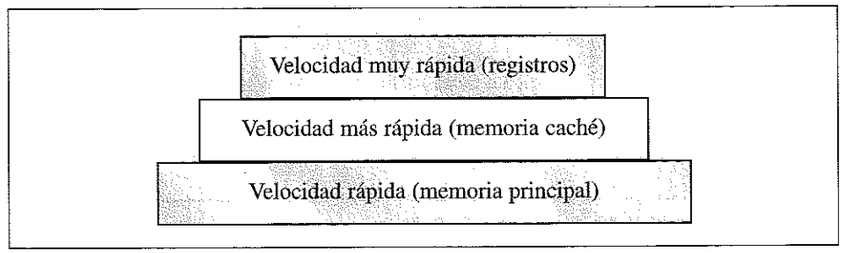


Figura 5.4 Jerarquía de la memoria

- Utilice una cantidad muy pequeña de memoria de alta velocidad cuando la velocidad sea crucial. Los registros dentro del CPU son de este tipo.
- Utilice una cantidad moderada de memoria de velocidad media para almacenar datos a los cuales se accede con frecuencia. La memoria caché, que se analiza enseguida, es de este tipo.
- Utilice una gran cantidad de memoria de baja velocidad para datos a los cuales no se accede con frecuencia. La memoria principal es de este tipo de memoria.

MEMORIA CACHÉ

La **memoria caché** es más rápida que la memoria principal pero más lenta que el CPU y los registros dentro del CPU. La memoria caché, la cual por lo general es pequeña en tamaño, se coloca entre el CPU y la memoria principal (figura 5.5).

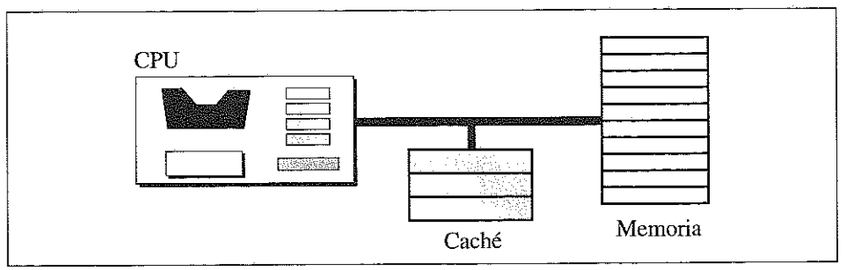


Figura 5.5 Caché

La memoria caché contiene en cualquier momento una copia de una porción de la memoria principal. Cuando el CPU necesita acceder a una palabra en la memoria principal, sigue este procedimiento:

1. El CPU revisa la caché.
2. Si la palabra está ahí, copia la palabra; si no, el CPU accede a la memoria principal y copia un bloque de memoria comenzando con la palabra deseada. El bloque reemplaza el contenido previo de la memoria caché.
3. El CPU accede a la caché y copia la palabra.

Este procedimiento puede acelerar las operaciones; si la palabra está en la caché, se tiene acceso a ella de inmediato. Si la palabra no está en la caché, la palabra y un bloque entero se copian a la caché. Puesto que es muy probable que el CPU, en su ciclo siguiente, necesite obtener acceso a las palabras que siguen a la primera palabra, la existencia de la caché acelera el procesamiento.

Tal vez el lector se pregunte por qué la memoria caché es tan eficiente a pesar de su tamaño pequeño. La respuesta está en la regla 80-20. Se ha observado que la mayoría de las computadoras por lo general invierte el 80 por ciento del tiempo en obtener acceso sólo al 20 por ciento de los datos. En otras palabras, se tiene acceso a los mismos datos una y otra vez. La memoria caché, con su gran velocidad, puede alojar este 20 por ciento para hacer que el acceso sea más rápido al menos el 80 por ciento del tiempo.

5.3 ENTRADA/SALIDA

El tercer subsistema en una computadora es la colección de dispositivos conocidos como el **subsistema de entrada/salida (E/S)**. Este subsistema permite a una computadora comunicarse con el mundo exterior y almacenar programas y datos aun cuando no esté encendida. Los dispositivos de entrada/salida pueden dividirse en dos categorías generales: dispositivos de almacenamiento y dispositivos que no son de almacenamiento.

DISPOSITIVOS QUE NO SON DE ALMACENAMIENTO

Los **dispositivos que no son de almacenamiento** permiten al CPU o a la memoria comunicarse con el mundo exterior, pero no pueden almacenar información.

Teclado y monitor

El teclado y el monitor son dos de los dispositivos de entrada/salida que no son de almacenamiento. El **teclado** proporciona la entrada; el **monitor** despliega la salida y al mismo tiempo repite la entrada que se introduce en el teclado. Los programas, comandos y datos son entrada o salida que utiliza cadenas de caracteres. Los caracteres se codifican usando un código como ASCII (véase el apéndice A).

Impresora

Una **impresora** es un dispositivo de salida que crea un registro permanente; se trata de un dispositivo que no es de almacenamiento porque el material impreso no puede introducirse otra vez directamente en una computadora a menos que alguien lo teclee o lo digitalice en un escáner.

DISPOSITIVOS DE ALMACENAMIENTO

Los **dispositivos de almacenamiento**, aun cuando se clasifican como dispositivos de entrada/salida, pueden almacenar grandes cantidades de información que se recuperará[1] en un momento posterior. Son más económicos que la memoria principal y su contenido no es volátil (no se borra cuando se apaga la computadora). A veces se les llama dispositivos de almacenamiento auxiliares. Los clasificamos como magnéticos u ópticos.

Dispositivos de almacenamiento magnéticos

Este tipo de dispositivos utiliza la magnetización para almacenar bits de datos. Si un punto se magnetiza, representa un 1; si no se magnetiza, representa un 0.

Disco magnético Un **disco magnético** es uno o más discos apilados uno encima de otro. Los discos se cubren con una película magnética delgada. La información se almacena y se recupera de la superficie del disco usando una **cabeza de lectura/escritura** para cada superficie magnetizada del disco. La figura 5.6 muestra el diagrama de un disco magnético.

[1] *N. del T.* Bajo este contexto, recuperar significa tener acceso al dispositivo o a la localidad de almacenamiento para obtener la información.

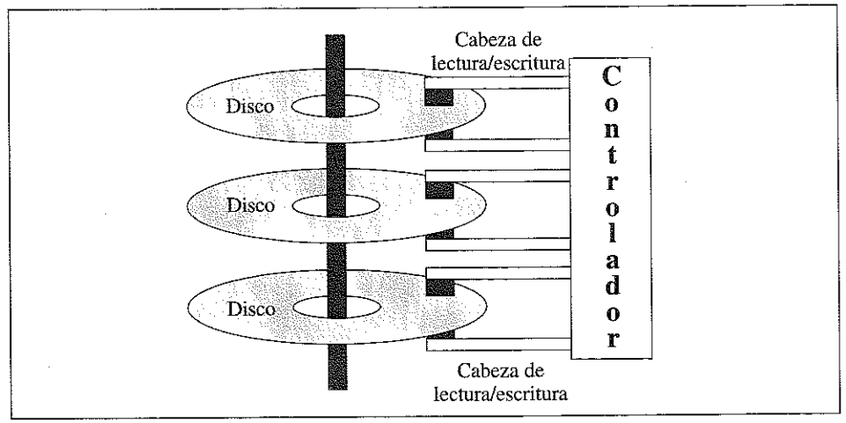


Figura 5.6 Distribución física de un disco magnético

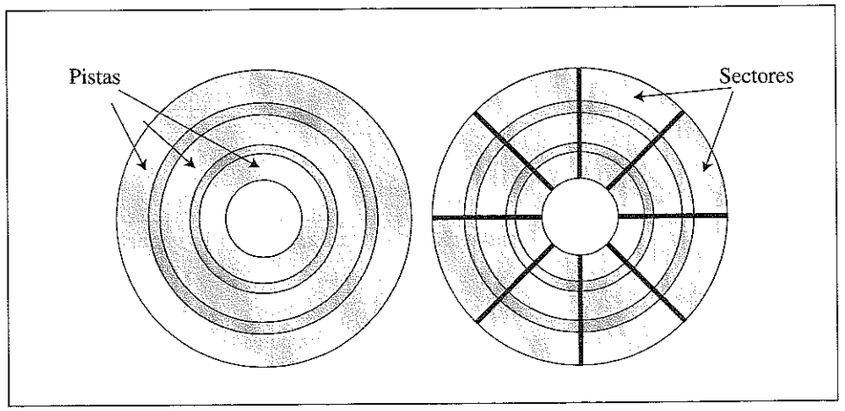


Figura 5.7 Organización de la superficie de un disco

- **Organización de la superficie.** Para organizar los datos almacenados en el disco, cada superficie se divide en **pistas**, las cuales a su vez se dividen en sectores (figura 5.7). Las pistas están separadas por un **espacio entre pistas** y los sectores están separados por un **espacio entre sectores**.

- **Acceso a datos.** Un disco magnético se considera un dispositivo de acceso aleatorio. No obstante, un sector es el área de almacenamiento más pequeña a la que puede tener acceso a la vez. Un bloque de datos puede almacenarse en uno o más sectores y recuperarse sin necesidad de recuperar el resto de la información en el disco.

- **Rendimiento.** El rendimiento de un disco depende de varios factores; los más importantes son la velocidad de rotación, el tiempo de búsqueda y el tiempo de transferencia. La **velocidad de rotación** define qué tan rápido gira el disco. El **tiempo de búsqueda** define el tiempo para mover la cabeza de lectura/escritura a la pista deseada donde se almacenan los datos. El **tiempo de transferencia** define el tiempo para mover los datos del disco al CPU o a la memoria.

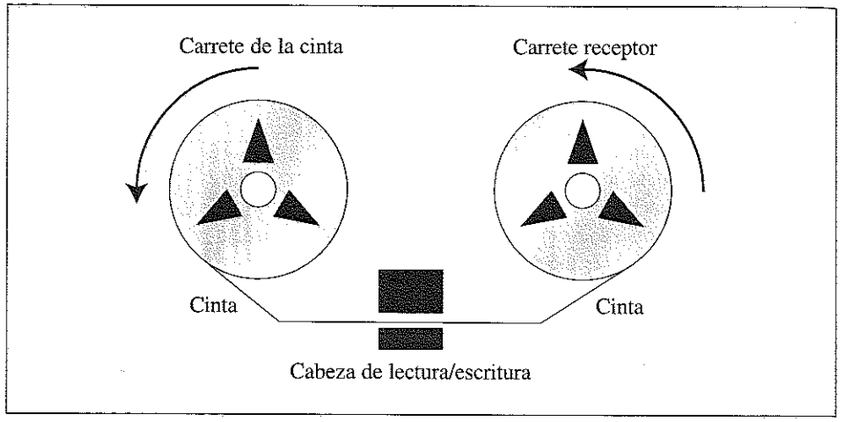


Figura 5.8 Configuración mecánica de una cinta

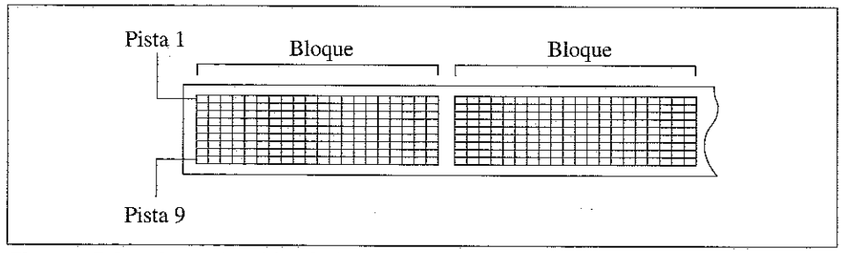


Figura 5.9 Organización de la superficie de una cinta

Cinta magnética Una **cinta magnética** viene en varios tamaños. Un tipo común es una cinta de plástico de media pulgada cubierta con una película magnética gruesa. La cinta se monta en dos carretes y utiliza una cabeza de lectura escritura que lee o escribe la información cuando la cinta pasa por ella. La figura 5.8 muestra la configuración mecánica de una cinta magnética.

- **Organización de la superficie.** El ancho de la cinta se divide en nueve pistas. Cada punto de una pista puede almacenar un bit de información. Nueve puntos verticales pueden almacenar 8 bits de información referente a un byte, más un bit para detección de error (figura 5.9).
- **Acceso a datos.** Una cinta magnética se considera un dispositivo de acceso secuencial. Aunque la superficie puede dividirse en bloques, no hay un mecanismo de direccionamiento para acceder a cada bloque. Para recuperar un bloque específico en una cinta, se necesita pasar por todos los bloques previos.
- **Rendimiento.** Aunque una cinta magnética es más lenta que un disco magnético, es más económica. Actualmente, la gente usa cintas magnéticas para respaldar grandes cantidades de información.

Dispositivos de almacenamiento óptico

Los **dispositivos de almacenamiento óptico** son una nueva tecnología que utiliza luz (láser) para almacenar y recuperar datos. El uso de la tecnología de almacenamiento óptico sucedió a la invención del CD (disco compacto), utilizado para almacenar información de audio. Hoy día, la misma tecnología (ligeramente mejorada) se utiliza para almacenar información en una computadora. Los dispositivos que usan esta tecnología incluyen CD-ROM, CD-R, CD-RW y DVD.

CD-ROM El **disco compacto de memoria de sólo lectura (CD-ROM**: *compact disc read-only memory*) utiliza la misma tecnología que el CD (disco compacto), originalmente desarrollado por Phillips y Sony para grabar música. La única diferencia entre estas dos tecnologías es la mejora; una unidad de CD-ROM es más robusta y busca errores. La figura 5.10 muestra los pasos en la creación y el uso de un CD-ROM.

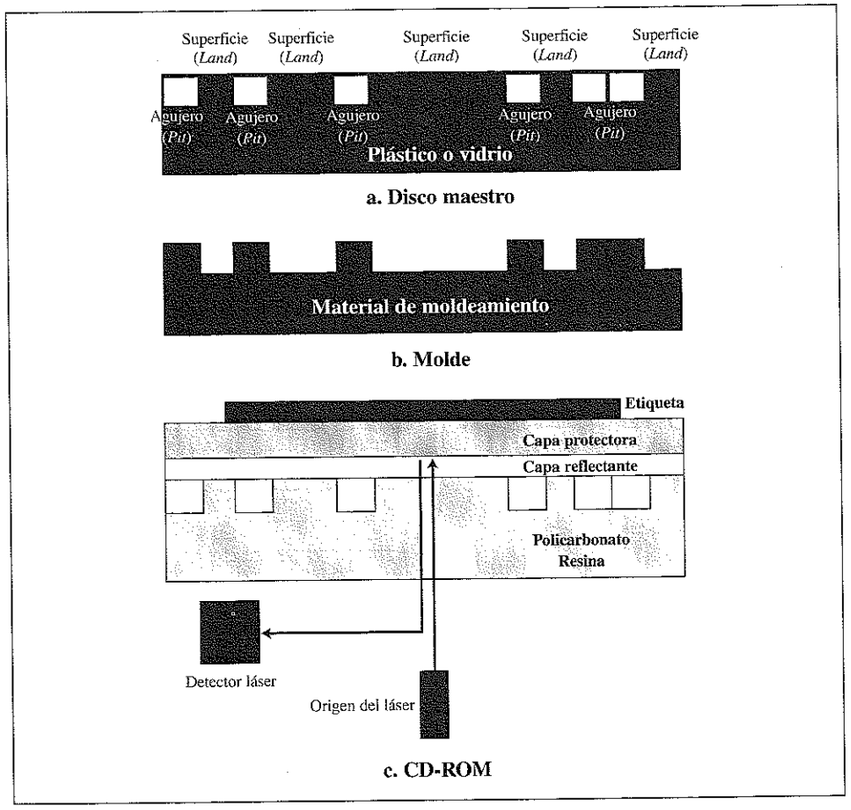


Figura 5.10 Creación y uso de un CD-ROM

- **Creación.** La tecnología de CD-ROM utiliza tres pasos para crear un gran número de discos:
  a. Un **disco maestro** se crea usando un láser infrarrojo de alta potencia que hace patrones de bits en plástico cubierto. El láser traduce el patrón de bits en una secuencia de **agujeros** (*pits*: hoyos) y **superficies** (*lands*: superficies sin hoyos). Los agujeros por lo general representan los 0 y las superficies, los 1. Sin embargo, ésta es sólo una convención y puede ser revertida. Otros esquemas utilizan una transición (agujero a hoyo u hoyo a agujero) para representar el 1 y la falta de transición para representar el 0.
  b. Desde el disco maestro se hace un molde. En el molde, los agujeros (hoyos) se remplazan por protuberancias.
  c. **Resina de policarbonato** fundida se inyecta en el molde para producir los mismos agujeros que el disco maestro. Una capa muy delgada de aluminio (que sirve como una superficie reflectante) se añade al policarbonato. Encima de éste, se aplica una capa protectora de laca y se añade una etiqueta. Se repite sólo este paso para cada disco.

- **Lectura.** El CD-ROM se lee usando un rayo láser de baja potencia que proviene de la unidad de la computadora. El rayo láser se refleja en la superficie de aluminio cuando pasa a través de una superficie. Se refleja dos veces cuando se encuentra un agujero, una vez por el borde del agujero y una vez por el borde del aluminio. Las dos reflexiones tienen un efecto destructivo debido a que la profundidad del agujero se elige para ser exactamente un cuarto de la longitud de onda del rayo. En otras palabras, el sensor instalado en el dispositivo detecta más luz cuando el punto es una zona y menos luz cuando el punto es un agujero; puede leer lo que se registra en el disco maestro original.

- **Formato.** La tecnología de CD-ROM utiliza un formato distinto a un disco magnético (figura 5.11). El formato de datos en un CD-ROM se basa en:
  a. Un bloque de datos de ocho bits se transforma en un símbolo de 14 bits que usa un método de corrección de errores llamado código Hamming.
  b. Se forma un Frame de 42 símbolos (14 bits/símbolo).
  c. Se forma un sector de 96 frames (2352 bytes).

- **Velocidad.** Las unidades de CD-ROM vienen en diferentes velocidades. La velocidad simple se llama 1x, la doble se llama 2x y así sucesivamente. Si la unidad es de velocidad simple puede leer hasta 153 600 bytes por segundo. La tabla 5.2 muestra las velocidades y la tasa de transferencia de datos correspondiente.

Velocidad	Tasa de transferencia	Aproximación
1x	153 600 bytes por segundo	150 KB/s
2x	307 200 bytes por segundo	300 KB/s
4x	614 400 bytes por segundo	600 KB/s
6x	921 600 bytes por segundo	900 KB/s
8x	1 228 800 bytes por segundo	1.2 MB/s
12x	1 843 200 bytes por segundo	1.8 MB/s
16x	2 457 600 bytes por segundo	2.4 MB/s
24x	3 688 400 bytes por segundo	3.6 MB/s
32x	4 915 200 bytes por segundo	4.8 MB/s
40x	6 144 000 bytes por segundo	6 MB/s

Tabla 5.2 Velocidades de CD-ROM

- **Aplicación.** Los gastos involucrados en la creación de un disco maestro, el molde y el disco real pueden justificarse si existe un gran número de clientes potenciales. Dicho de otra forma, esta tecnología es económica si los discos se producen masivamente.

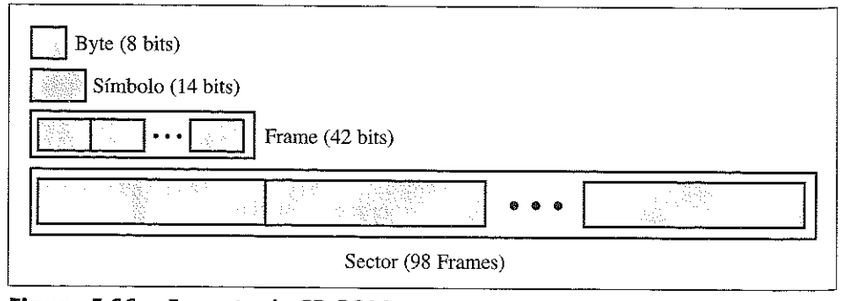


Figura 5.11 Formato de CD-ROM

CD-R Como se mencionó, la tecnología de CD-ROM es justificable sólo si el fabricante puede crear una gran cantidad de discos. Por otra parte, el **disco compacto grabable (CD-R:** *Compact disc recordable*) permite a los usuarios crear uno o más discos sin incurrir en el gasto implicado en la creación de CD-ROM. Es particularmente útil para hacer respaldos. El usuario puede escribir sólo una vez en este disco, pero puede leerlo varias veces. Por esta razón, en ocasiones se le llama **WORM** (*write once, read many*: **escribir una vez, leer muchas**).

- **Creación.** La tecnología de CD-R utiliza los mismos principios que el CD-ROM para crear un disco (figura 5.12). A continuación se listan las diferencias:
  a. No hay disco maestro o molde.
  b. La capa reflectante está hecha de oro en lugar de aluminio.
  c. No hay agujeros (hoyos) físicos en el policarbonato; los agujeros y las superficies sólo se simulan. Para hacerlo, se añade una capa más de tinte, similar al material utilizado en fotografía, entre la capa reflectante y el policarbonato.
  d. Un rayo láser de alta potencia, creado por el quemador de CD de la unidad, forma un punto negro en el tinte (cambiando la composición química) lo cual simula un agujero. Las áreas que no son golpeadas por el rayo son las superficies.

- **Lectura.** Los CD-R pueden leerse mediante una unidad de CD-ROM o de CD-R. Esto significa que cualquier diferencia debe ser transparente para la unidad. El mismo rayo láser de baja potencia pasa enfrente de los agujeros simulados y las superficies. Para una superficie, el rayo alcanza la capa reflectante y se refleja. Para un agujero simulado, el punto es opaco de modo que el rayo no puede reflejarse de regreso.

- **Formato y velocidad.** El formato, la capacidad y la velocidad de los CD-R son los mismos que los del CD-ROM.

- **Aplicación.** Esta tecnología es muy atractiva para la gente que desea crear y distribuir un pequeño número de discos; también es muy útil para hacer depósitos de archivos (*archives*) y respaldos.

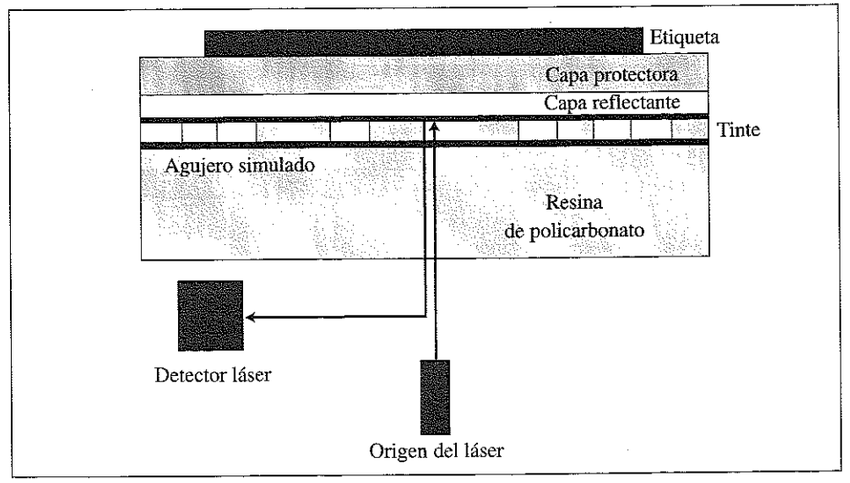


Figura 5.12 Creación de un CD-R

CD-RW Aunque los CD-R se han vuelto muy populares, sólo se puede escribir en ellos una vez. Para sobrescribir materiales previos, existe una nueva tecnología que crea un nuevo tipo de disco llamado **disco compacto de reescritura (CD-RW:** *compact disc rewritable*). A veces se le llama disco óptico borrable.

- **Creación.** La tecnología de CD-RW utiliza los mismos principios que el CD-ROM para crear el disco (figura 5.13). Las diferencias se listan enseguida:
  a. En lugar de tinte la tecnología utiliza una aleación de plata, indio, antimonio y telurio. Esta aleación tiene dos estados estables: cristalino (transparente) y amorfo (no transparente).
  b. La unidad usa láser de alta potencia para crear agujeros simulados en la aleación (cambiándola de cristalina a amorfa).
- **Lectura.** La unidad utiliza el mismo rayo láser de baja potencia que el CD-ROM y el CD-R para detectar agujeros y zonas.
- **Borrado.** La unidad usa un rayo láser de potencia media para cambiar los agujeros a superficies. El rayo cambia un punto del estado amorfo al estado cristalino.
- **Formato y velocidad.** El formato, la capacidad y la velocidad de los CD-RW son los mismos que los del CD-ROM.
- **Aplicación.** Esta tecnología es definitivamente más atractiva que la tecnología de CD-R. Sin embargo, los CD-R son más populares por dos razones. Primero, los discos CD-R vírgenes son menos costosos que los discos CD-RW vírgenes. Segundo, los discos CD-R son preferibles en casos donde el disco creado no debe cambiar, ya sea de manera accidental o intencional.

DVD La industria ha sentido la necesidad de medios de almacenamiento digital con una capacidad aún mayor. La capacidad de un CD-ROM (650 MB) no es suficiente para almacenar información de video. El dispositivo de almacenamiento de memoria óptica más reciente en el mercado se llama **disco versátil digital (DVD**: *digital versatile disc*). Este disco utiliza una tecnología similar al CD-ROM pero con las diferencias siguientes:

1. Los agujeros son más pequeños: 0.4 micras de diámetro en lugar de 0.8 micras usados en los CD.
2. Las pistas están más cercanas entre sí.
3. El rayo es un láser rojo en vez de un infrarrojo.
4. El DVD usa una de dos capas de grabación y puede ser de una cara o de doble cara.

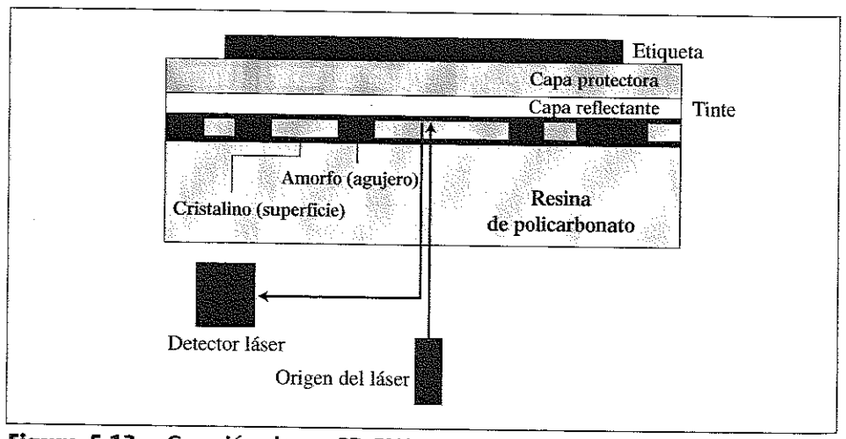


Figura 5.13 Creación de un CD-RW

- **Capacidad.** Estas mejoras dan como resultado mayores capacidades (tabla 5.3).

Característica	Capacidad
Una cara, capa simple	4.7 GB
Una cara, capa dual	8.5 GB
Doble cara, capa simple	9.4 GB
Doble cara, capa dual	17 GB

Tabla 5.3 Capacidades de DVD

- **Compresión.** La tecnología de DVD utiliza el formato MPEG (véase el capítulo 15) para compresión. Esto significa que un DVD de una sola cara y capa simple puede alojar 133 minutos (2 horas y 13 minutos) de video a una alta resolución. Esto también incluye tanto audio como subtítulos.
- **Aplicación.** Actualmente, la gran capacidad de los DVD atrae a muchas aplicaciones que necesitan almacenar un volumen grande de datos.

5.4 INTERCONEXIÓN DE SUBSISTEMAS

Las secciones previas explicaron resumidamente las características de los tres subsistemas (CPU, memoria principal y E/S) en una computadora independiente. En esta sección, exploramos cómo se interconectan estos tres subsistemas. La interconexión representa un importante papel porque la información requiere ser intercambiada entre los tres subsistemas.

CONEXIÓN DEL CPU Y LA MEMORIA

El CPU y la memoria normalmente se conectan por medio de tres grupos de líneas, cada una llamada **bus**: bus de datos, bus de direcciones y bus de control (figura 5.14).

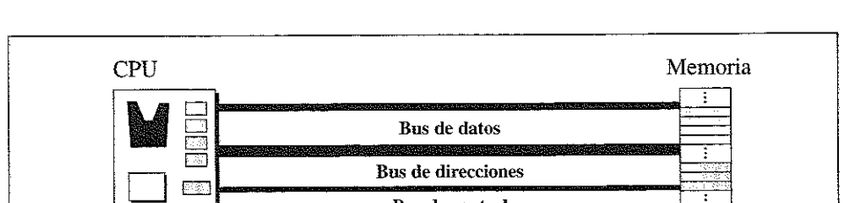


Figura 5.14 Conexión del CPU y la memoria usando tres buses

Bus de datos

El **bus de datos** está formado por varias líneas de control, cada una de las cuales transporta 1 bit a la vez. El número de líneas depende del tamaño de la palabra. Si la palabra mide 32 bits (4 bytes) en una computadora, se necesita un bus de datos con 32 líneas de modo que todos los 32 bits de una palabra puedan transmitirse al mismo tiempo.

Bus de direcciones

El **bus de direcciones** permite el acceso a una palabra en particular en la memoria. El número de líneas en el bus de dirección depende del espacio de direccionamiento de la memoria. Si la memoria tiene 2^n palabras, el bus de direcciones necesita transportar n bits a la vez. Por consiguiente, debe tener n líneas.

Bus de control

El **bus de control** lleva la comunicación entre el CPU y la memoria. Por ejemplo, debe haber un código enviado desde el CPU a la memoria para especificar una operación de lectura y escritura. El número de líneas utilizadas en el bus de control depende del número total de comandos de control que necesita la computadora. Si una computadora tiene 2^m acciones de control, necesita m líneas para el bus de control porque m bits pueden definir 2^m operaciones diferentes.

CONEXIÓN DE DISPOSITIVOS E/S

Los dispositivos E/S no pueden conectarse directamente a los buses que conectan el CPU y la memoria, dado que la naturaleza de los dispositivos E/S es diferente de la naturaleza del CPU y la memoria. Los dispositivos E/S son dispositivos electromecánicos, magnéticos u ópticos, mientras que el CPU y la memoria son dispositivos electrónicos. Los dispositivos E/S operan a una velocidad mucho más lenta que el CPU/memoria. Existe la necesidad de que un intermediario maneje esta diferencia. Los dispositivos de entrada/salida se conectan a los buses a través de lo que se conoce como **controlador de entrada/salida** o interfaz. Existe un controlador específico para cada dispositivo de entrada/salida (figura 5.15).

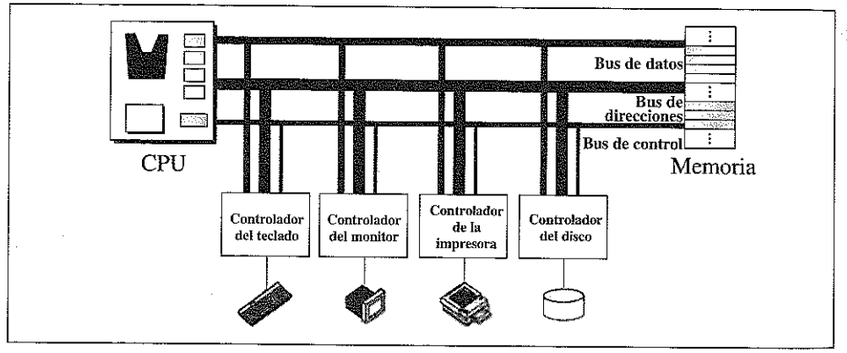


Figura 5.15 Conexión de los dispositivos E/S a los buses

Controladores

Los controladores o interfaces quitan el obstáculo entre la naturaleza del dispositivo E/S y el CPU/memoria. Un controlador puede ser un dispositivo serial o paralelo. Un controlador serial tiene sólo una línea de conexión al dispositivo. Un controlador paralelo tiene varias líneas de conexiones al dispositivo de modo que varios bits pueden transferirse a la vez.

Diversos tipos de controladores están en uso. Los más comunes actualmente son SCSI, FireWire y USB.

SCSI La **interfaz pequeña de sistemas de computadoras** (**SCSI**: *small computer system interface*) fue desarrollada en un principio para las computadoras Macintosh en 1984. Hoy día se utiliza en muchos sistemas. Tiene una interfaz paralela con 8, 16 y 32 líneas. La interfaz SCSI ofrece una conexión en cadena margarita como se muestra en la figura 5.16. Ambos extremos de la cadena deben estar terminados y cada dispositivo debe tener una dirección única (ID objetivo).

FireWire El estándar IEEE 1394 define una interfaz serial comúnmente llamada **FireWire**, la cual es una interfaz serial de alta velocidad que transfiere datos en paquetes, logrando una velocidad de tasa de hasta 50 MB/s. Puede utilizarse para conectar hasta 63 dispositivos en una cadena o una conexión de árbol (usando sólo una línea). La figura 5.17 muestra la conexión del dispositivo de entrada/salida a un controlador FireWire. No hay necesidad de terminación como en el controlador SCSI.

USB Un competidor para el controlador FireWire es el controlador **bus serial universal** (**USB**: *universal serial bus*). USB también es un controlador serial utilizado para conectar a una computadora dispositivos más lentos como el teclado y el ratón. Puede transferir datos

hasta a 1.5 MB/s. Tiene un bus de cuatro líneas de control; dos de ellos llevan energía eléctrica al dispositivo. La figura 5.18 muestra la conexión de un USB a los buses.

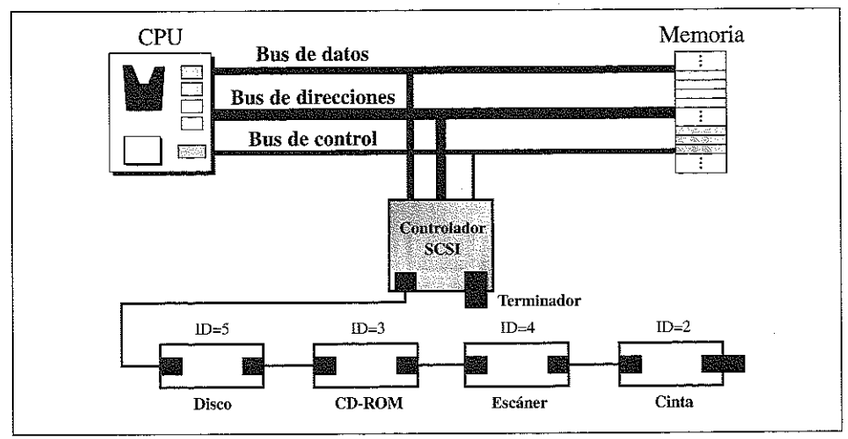


Figura 5.16 Controlador SCSI

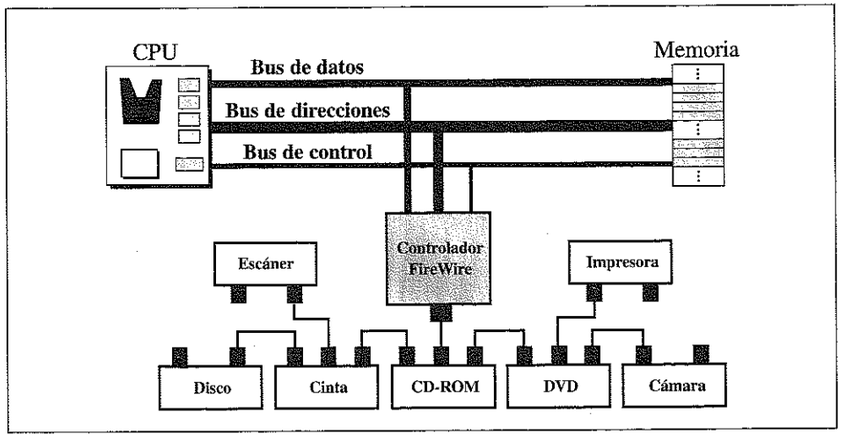


Figura 5.17 Controlador FireWire

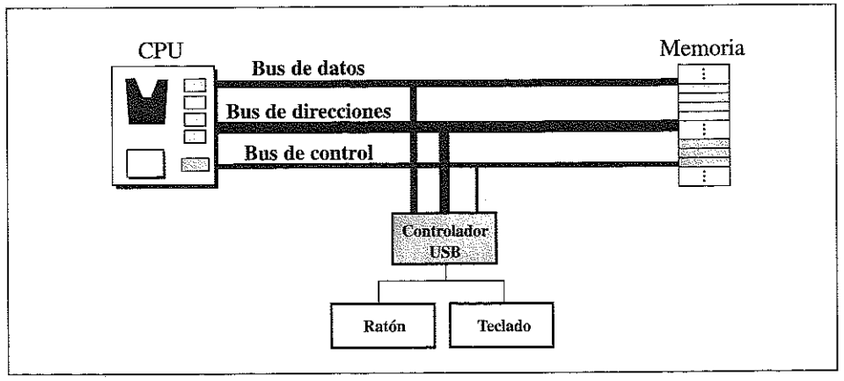


Figura 5.18 Controlador USB

DIRECCIONAMIENTO DE DISPOSITIVOS DE ENTRADA/SALIDA

El CPU por lo general utiliza el mismo bus para leer la memoria principal y el dispositivo E/S o escribir datos en ellos. La única diferencia es la instrucción. Si la instrucción se refiere a una palabra en la memoria principal, la transferencia de datos se realiza entre la memoria principal y el CPU. Si la instrucción identifica un dispositivo de entrada/salida, la transferencia de datos es entre el dispositivo de entrada/salida y el CPU. Existen dos métodos para manejar el direccionamiento de dispositivos E/S: E/S aislado y E/S por mapas de memoria.

E/S aislado

En el método **E/S aislado**, las instrucciones utilizadas para la memoria de lectura/escritura de la memoria son totalmente diferentes de las instrucciones de lectura/escritura usadas para los dispositivos E/S. Hay instrucciones para probar, controlar, leer de y escribir en dispositivos de entrada/salida. Cada dispositivo de entrada/salida tiene su propia dirección. Las direcciones de entrada/salida pueden traslaparse con las direcciones de memoria sin ninguna ambigüedad debido a que la instrucción en sí misma es diferente. Por ejemplo, el CPU puede usar el comando Read 05 para leer de la memoria la palabra 05 y utilizar el comando Input 05 para leer del dispositivo de entrada/salida 05. No hay confusión debido a que el comando Read es para leer de la memoria y el comando Input es para leer desde un dispositivo de entrada/salida (figura 5.19).

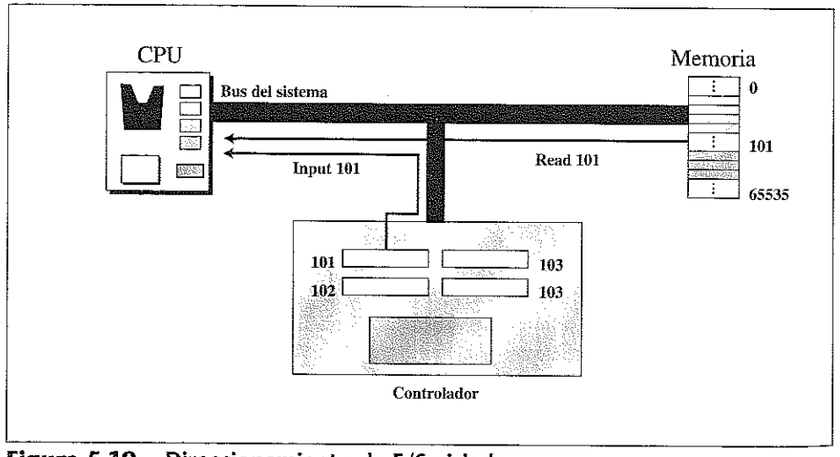


Figura 5.19 Direccionamiento de E/S aislado

E/S por mapas de memoria

En el método **E/S por mapas de memoria**, el CPU trata a cada registro en el controlador de entrada/salida como una palabra en la memoria. En otras palabras, el CPU no tiene instrucciones separadas para transferir datos desde la memoria o desde dispositivos de entrada/salida. Por ejemplo, sólo existe una instrucción Read. Si la dirección define una palabra desde la memoria, los datos se leen desde esa palabra. Si la dirección define un registro de un dispositivo de entrada/salida, los datos se leen desde ese registro. La ventaja de la configuración por mapas de memoria es un pequeño número de instrucciones; todas las instrucciones de la memoria pueden ser utilizadas por los dispositivos de entrada/salida. La desventaja es que la parte del espacio de direccionamiento usado para la memoria se asigna a los registros en los controladores de entrada/salida. Por ejemplo, si usted tiene cinco controladores de entrada/salida y cada uno tiene cuatro registros, se utilizan 20 direcciones para este propósito. El tamaño de la memoria se reduce en 20 palabras. La figura 5.20 muestra el concepto de E/S por mapas de memoria.

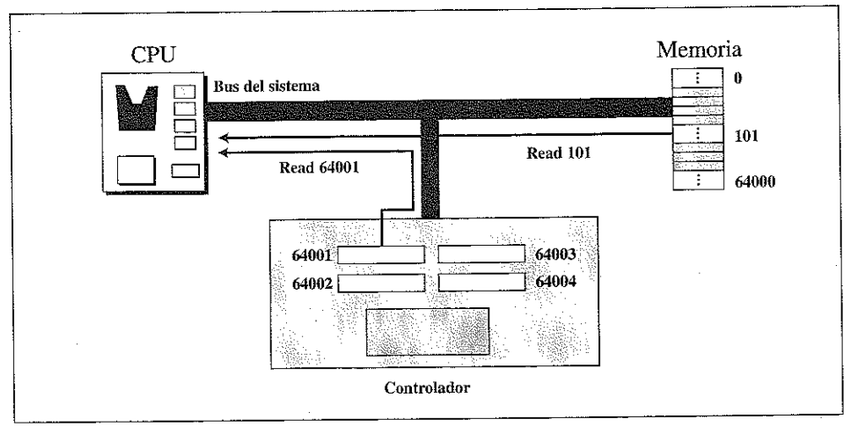


Figura 5.20 Direccionamiento de E/S por mapas de memoria

5.5 EJECUCIÓN DE PROGRAMAS

En la actualidad, las computadoras de uso general utilizan una serie de instrucciones llamada programa para procesar los datos. Una computadora ejecuta el programa para crear datos de salida a partir de los datos de entrada. Tanto el programa como los datos se almacenan en la memoria.

CICLO DE MÁQUINA

El CPU utiliza **ciclos de máquina** repetidos para ejecutar instrucciones en el programa, una por una, de principio a fin. Un ciclo simplificado puede consistir en tres pasos: buscar y traer (fetch), decodificar (decode) y ejecutar (execute) (figura 5.21).

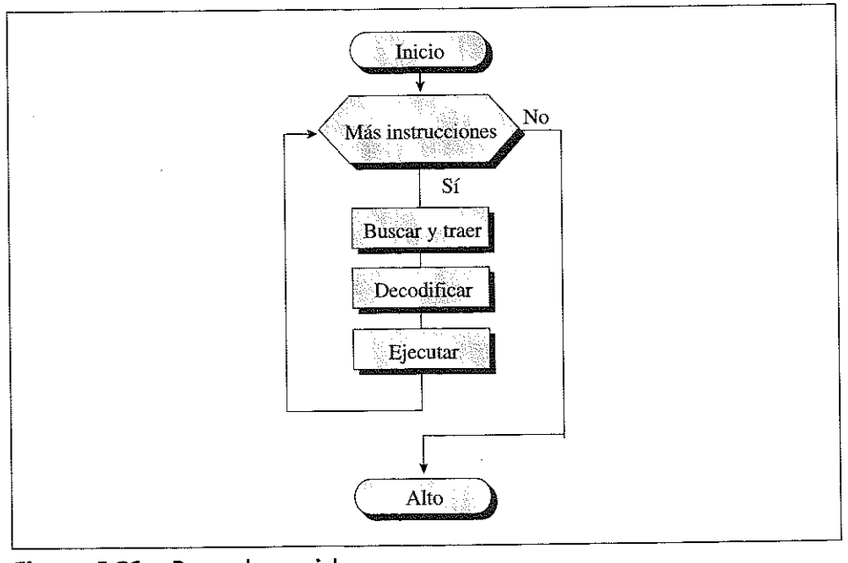


Figura 5.21 Pasos de un ciclo

Buscar y traer

En el paso **buscar y traer**, la unidad de control ordena al sistema copiar la siguiente instrucción en el registro de instrucción en el CPU. La dirección de la instrucción a ser copiada se mantiene en el registro del contador del programa. Después de copiar, el contador del programa se incrementa para referirse a la siguiente instrucción en la memoria.

Decodificar (decode)

Cuando la instrucción está en el registro de instrucción, la unidad de control la decodifica. El resultado de este paso de decodificación es código binario para algunas operaciones que realizará el sistema.

Ejecutar (execute)

Después de que se decodifica la instrucción, la unidad de control envía la orden de la tarea a un componente en el CPU. Por ejemplo, la unidad de control puede indicar al sistema que cargue (lea) un elemento de datos de la memoria, o el CPU puede indicar a la ALU que sume el contenido de dos registros de entrada y coloque el resultado en un registro de salida. Éste es el paso **ejecutar**.

UN EJEMPLO DE CICLO DE MÁQUINA

Veamos una operación muy simple como la suma de dos enteros. Una computadora con una arquitectura simple necesita al menos cuatro instrucciones para este trabajo. Las cuatro instrucciones y los dos enteros de entrada residen en la memoria antes de la ejecución del programa; el resultado estará en la memoria después de la ejecución del programa.

Aunque todo se representa mediante patrones de bits, por simplicidad suponga que los números y las direcciones están en decimales. También suponga que las instrucciones están en las localidades de memoria 70, 71, 72 y 73. Los datos de entrada se almacenan en las localidades 200 y 201. Los datos de salida se almacenarán en la localidad de memoria 202.

La figura 5.22 muestra la memoria y el CPU antes de la ejecución del programa. R1, R2 y R3 son registros generales. R1 y R2 alojan la entrada de datos; R3 aloja la salida de datos. El registro I es el registro de instrucción y PC es el contador del programa. La figura 5.23 muestra los resultados de cuatro operaciones en la memoria y los registros.

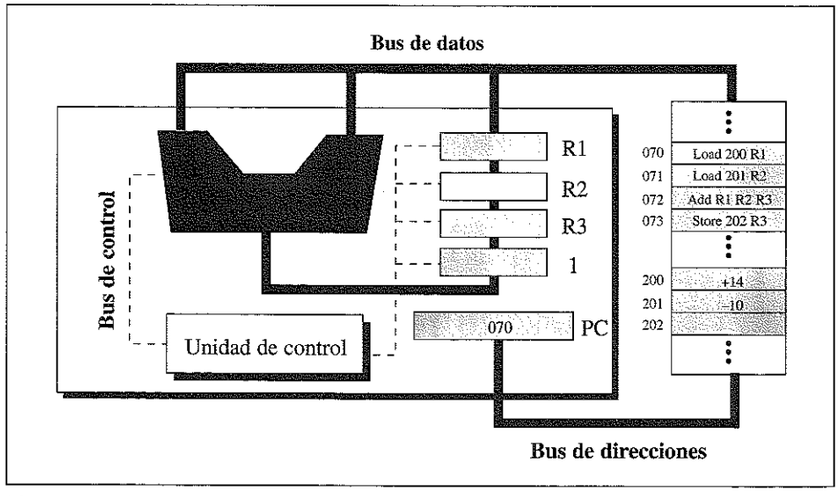


Figura 5.22 Contenido de la memoria y el registro antes de la ejecución

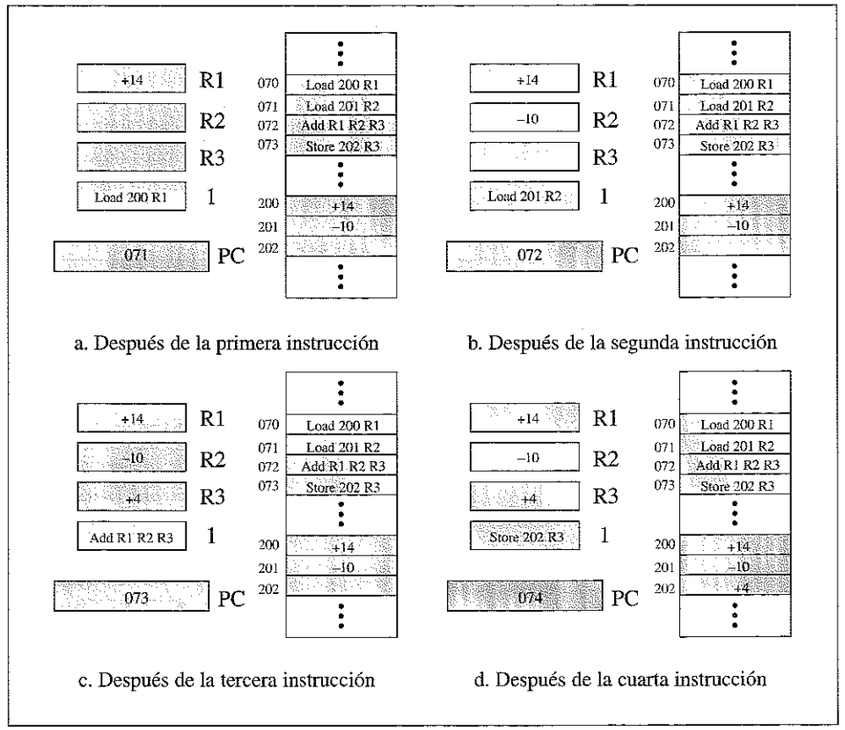


Figura 5.23 Contenido de la memoria y el registro después de cada ciclo

Primera operación

En la primera operación (Load 200 R1), la máquina pasa por los tres pasos de buscar y traer, decodificar y ejecutar para cargar el contenido de la localidad de memoria 200 en el registro de datos R1.

Segunda operación

En la segunda operación (Load 201 R2), la máquina pasa por los tres pasos de buscar y traer, decodificar y ejecutar para cargar el contenido de la localidad de memoria 201 en el registro de datos R2.

Tercera operación

En la tercera operación (Add R1 R2 R3), la máquina pasa por los tres pasos de buscar y traer, decodificar y ejecutar para sumar los datos en R1 y R2, y almacenar el resultado en R3.

Cuarta operación

En la cuarta operación (Store 202 R3), la máquina pasa por los tres pasos de buscar y traer, decodificar y ejecutar para almacenar el resultado de la operación en la localidad de memoria 202.

OPERACIÓN DE ENTRADA/SALIDA

Existe la necesidad de comandos para transferir datos desde los dispositivos E/S al CPU y la memoria. Debido a que los dispositivos de entrada/salida operan a velocidades mucho más lentas que el CPU, la operación del CPU debe sincronizarse de alguna manera con el dispositivo de entrada/salida. Se han ideado tres métodos para esta sincronización: E/S programada, E/S manejada por interrupciones y acceso directo a memoria (DMA: *direct memory access*).

E/S programada

En el método de **E/S programada**, la sincronización es muy primitiva; el CPU espera el dispositivo E/S. La transferencia de datos entre el dispositivo E/S y el CPU se realiza mediante una instrucción en el programa. Cuando el CPU encuentra una instrucción E/S, no hace na-

da más hasta que la transferencia está completa. El CPU revisa constantemente el estado de la unidad de E/S; si el dispositivo está listo para transferir, los datos se transfieren al CPU. Si el dispositivo no está listo, el CPU continúa revisando el estado hasta que el dispositivo E/S está listo (figura 5.24). El gran problema aquí es que el tiempo del CPU se gasta en la revisión del estado del dispositivo E/S para cada unidad de datos a ser transferida. Observe que los datos se transfieren a la memoria después de la operación de entrada; los datos se transfieren desde la memoria antes de la operación de salida.

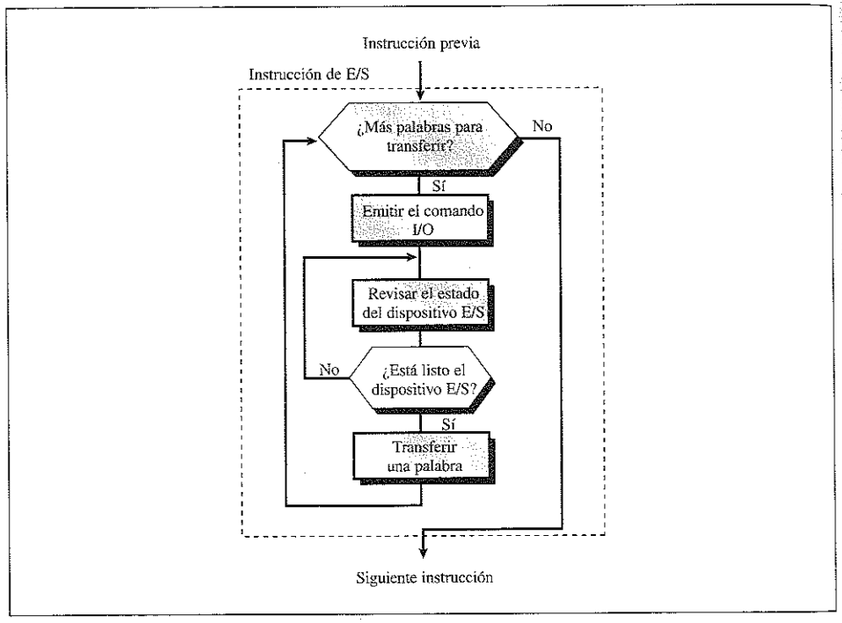


Figura 5.24 E/S programada

E/S manejada por interrupciones

En el método de **E/S manejada por interrupciones**, el CPU informa al dispositivo E/S que va a ocurrir una transferencia, pero no prueba el estado del dispositivo E/S de manera continua. El dispositivo E/S informa (interrumpe) al CPU cuando está listo. Durante este tiempo, el CPU puede hacer otros trabajos como ejecutar otros programas o transferir datos desde o hacia otros dispositivos E/S (figura 5.25).

En este método, el tiempo del CPU no se desperdicia. Mientras que el dispositivo E/S lento está terminando una tarea, el CPU puede hacer algo más. Observe que, al igual que la E/S programada, este método también transfiere datos entre el dispositivo y el CPU. Los datos se transfieren a la memoria después de la operación de entrada; los datos se transfieren desde la memoria después de la operación de salida.

Acceso directo a memoria (DMA)

El tercer método de transferencia de datos es el **acceso directo a memoria (DMA)**. Este método transfiere un bloque grande de datos entre un dispositivo E/S de alta velocidad, por ejemplo un disco, y la memoria directamente (sin pasar por el CPU). Esto requiere un controlador DMA que libera al CPU de algunas de sus funciones. El controlador DMA tiene registros para mantener un bloque de datos antes y después de la transferencia a la memoria. La figura 5.26 muestra la conexión DMA para el bus general. En este método, para una operación E/S el CPU envía un mensaje al DMA. El mensaje contiene el tipo de transferencia (entrada o salida), la dirección de inicio de la localidad de memoria y el número de bytes a ser transferidos. El CPU ahora está disponible para otros trabajos.

Cuando está listo para transferir datos, el controlador DMA informa al CPU que necesita tomar control de los buses. El CPU deja de usar los buses y permite que el controlador los use. Después de la transferencia de datos, directamente entre el DMA y la memoria, el CPU continúa su operación normal (figura 5.27). Observe que en este método el CPU queda inactivo durante un breve lapso.

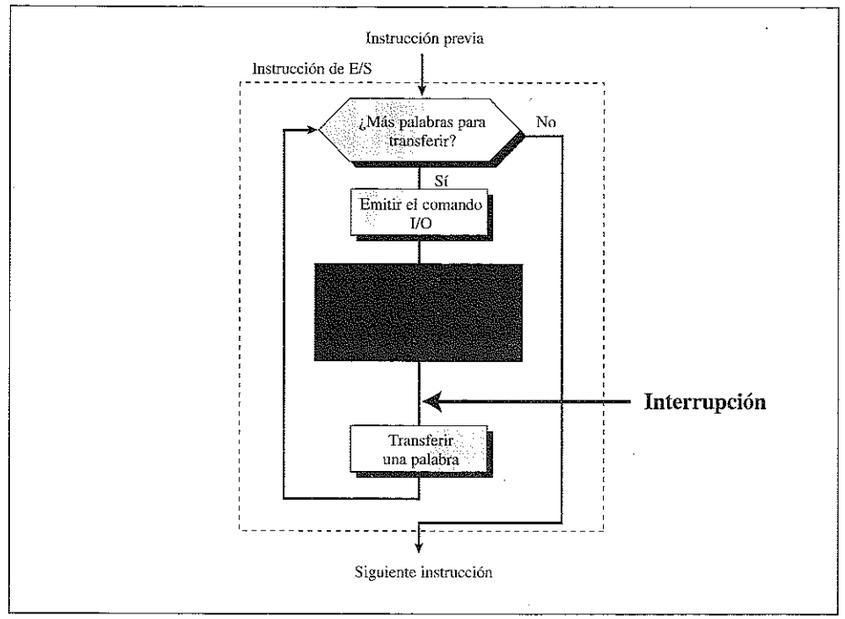


Figura 5.25 E/S manejada por interrupciones

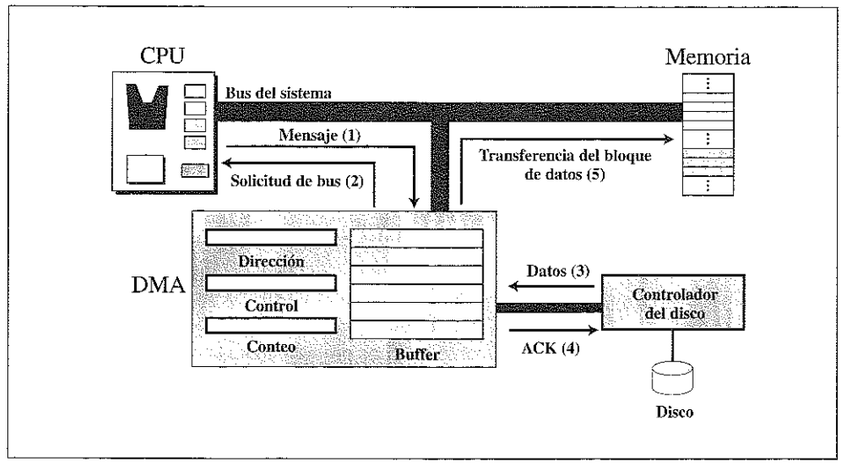


Figura 5.26 Conexión DMA al bus general

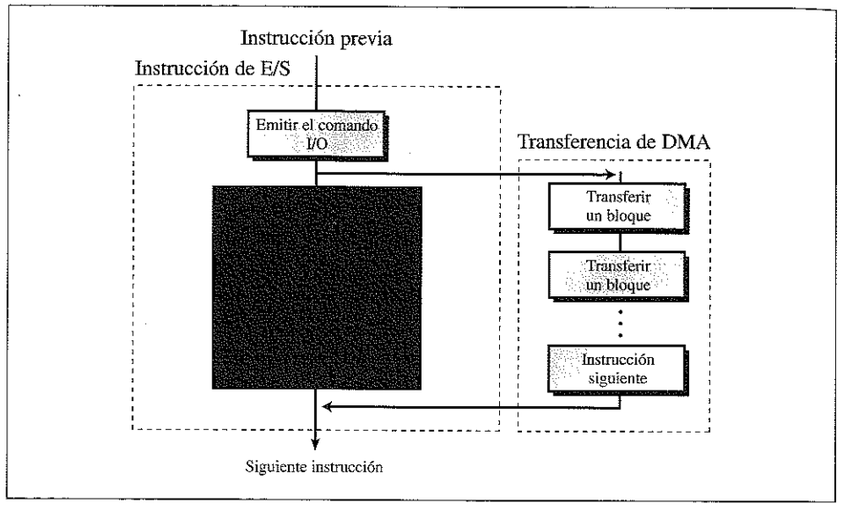


Figura 5.27 Entrada/salida de DMA

No obstante, la duración del tiempo de inactividad es muy corta comparada con otros métodos; el CPU está inactivo sólo durante la transferencia de datos entre el DMA y la memoria, no cuando el dispositivo prepara los datos.

5.6 DOS ARQUITECTURAS DIFERENTES

El diseño de computadoras ha pasado por muchos cambios durante las últimas décadas. Los dos diseños fundamentales que dominan el mercado son CISC y RISC. Estas dos metodologías de diseño se analizan brevemente a continuación.

CISC

CISC son las siglas en inglés de **computadora con conjunto de instrucciones complejas** (*complex instruction set computer*). La estrategia en que se basa la arquitectura CISC es tener un conjunto grande de instrucciones, incluyendo las complejas. La programación en CISC es más fácil que en el otro diseño debido a que hay una instrucción para una tarea simple o compleja. Los programadores no tienen que escribir un conjunto de instrucciones para que realicen una tarea compleja.

La complejidad del conjunto de instrucciones hace que el sistema de circuitos del CPU y la unidad de control sean muy complicadas. Los diseñadores de la arquitectura CISC han ideado una solución para reducir esta complejidad. La programación se realiza en dos niveles. Una instrucción en el lenguaje de máquina no se ejecuta directamente por el CPU. El CPU ejecuta sólo operaciones simples llamadas microoperaciones. Una instrucción compleja se transforma en un conjunto de estas operaciones simples y entonces son ejecutadas por el CPU. Esto requiere la adición de una memoria especial llamada micromemoria que aloja el conjunto de operaciones para cada instrucción compleja el conjunto de la máquina. El tipo de programación que utiliza microoperaciones se llama microprogramación.

Una objeción a la arquitectura CISC es la sobrecarga asociada con la microprogramación y el acceso a la micromemoria. Sin embargo, los defensores de la arquitectura sostienen que esto se compensa por los programas más pequeños en el nivel de la máquina.

Un ejemplo de la arquitectura CISC puede verse en la serie de procesadores Pentium desarrollada por Intel.

RISC

RISC son las siglas en inglés de **computadora con conjunto de instrucciones reducidas** (*reduced instruction set computer*). La estrategia en que se basa la arquitectura RISC es tener un conjunto pequeño de instrucciones que realicen un número mínimo de operaciones simples. Las instrucciones complejas se simulan usando un subconjunto de instrucciones simples. Esto provoca que la programación en RISC sea más difícil y extensa que en el otro diseño, porque la mayoría de las instrucciones complejas son simuladas usando instrucciones simples.

Un ejemplo de la arquitectura RISC es la serie de procesadores PowerPC utilizada en las computadoras Apple.

5.7 TÉRMINOS CLAVE

acceso directo a memoria (DMA)
agujero
bus
bus de datos
bus de control
bus de direcciones
bus serial universal (USB)
buscar y traer
cabeza de lectura/escritura
ciclo de máquina
cinta magnética
contador de programa
controlador de entrada/salida
computadora con conjunto de instrucciones complejas (CISC)
computadora con conjunto de instrucciones reducidas (RISC)
disco compacto de reescritura (CD-RW)
disco compacto grabable (CD-R)
disco maestro
disco magnético
disco versátil digital (DVD)
dispositivo de almacenamiento
dispositvo de almacenamiento óptico
dispositivo que no es de almacenamiento
E/S aislado
E/S manejada por interrupciones
E/S por mapas de memoria
E/S programada
ejecutar
escribir una vez, leer muchas (WORM)
espacio de direccionamiento
espacio entre pistas
espacio entre sectores
FireWire
frame
impresora
interfaz pequeña de sistemas de computadoras (SCSI)
memoria
memoria caché
memoria de acceso aleatorio (RAM)
memoria de sólo lectura de disco compacto (CD-ROM)
memoria de sólo lectura programable y borrable electrónicamente (EEPROM)
memoria de sólo lectura programable y borrable (EPROM)
memoria sólo de lecutra programable (PROM)
memoria de sólo lectura (ROM)
memoria principal
Monitor
operación aritmética
operación lógica
pista
patrón de bits
RAM dinámica (DRAM)
RAM estática (SRAM)
registro
registro de datos
registro de instrucciones
resina de policarbonato
sector
subsistema de entrada/salida (E/S)
superficie
teclado
tiempo de búsqueda
tiempo de transferencia
unidad central de procesamiento (CPU)
unidad de control
unidad lógica aritmética (ALU)
velocidad rotacional

5.8 RESUMEN

- Una computadora tiene tres subsistemas: el CPU, la memoria principal y el subsistema de entrada/salida.
- El CPU realiza operaciones en los datos y tiene un ALU, una unidad de control y una serie de registros.
- El ALU realiza operaciones lógicas y aritméticas.
- Los registros son dispositivos de almacenamiento independientes que alojan los datos temporalmente. Los registros pueden alojar datos e instrucciones, y también funcionan como un contador de programa.
- La unidad de control supervisa las operaciones en una computadora.
- La memoria principal es una colección de localidades de almacenamiento.
- Las direcciones de memoria se definen usando enteros binarios sin signo.
- La RAM provee la mayor parte de la memoria en una computadora. La SRAM utiliza las compuertas flip-flop tradicionales para alojar los datos y la DRAM usa capacitores.

- El contenido de la ROM viene del fabricante; sólo se permite a los usuarios leerla, pero no escribir en ella.
- Las computadoras necesitan una memoria de alta velocidad para los registros, memoria de velocidad media para la memoria caché y memoria de baja velocidad para la memoria principal.
- El subsistema de entrada/salida es una colección de dispositivos que permite a una computadora comunicarse con el mundo exterior.
- Estos dispositivos son ya sea dispositivos de almacenamiento o dispositivos que no son de almacenamiento.
- El teclado, el monitor y la impresora son ejemplos de dispositivos que no son de almacenamiento.
- Un disco magnético es un dispositivo de almacenamiento formado por discos apilados, con cada disco en la pila dividido en pistas y sectores.
- La cinta magnética es un dispositivo de almacenamiento en el cual la cinta se divide en pistas. El acceso a los datos es secuencial.
- Un CD-ROM es un dispositivo de almacenamiento óptico en el cual el fabricante quema los datos en el disco. Los datos no pueden borrarse.
- Un CD-R es un dispositivo de almacenamiento óptico en el cual el usuario quema los datos en el disco. Los datos no pueden borrarse.
- Un CD-RW es un dispositivo de almacenamiento óptico en el cual el usuario quema los datos en el disco. Los datos pueden borrarse y se puede volver a escribir en el disco varias veces.
- Un DVD es un dispositivo de almacenamiento óptico de gran capacidad.
- Un bus de datos, un bus de direcciones y un bus de control conectan el CPU a la memoria.
- Un controlador maneja las operaciones de E/S entre el CPU/memoria y los dispositivos E/S más lentos. SCSI, FireWire y USB son controladores comunes.
- Existen dos métodos para manejar el direccionamiento de dispositivos E/S: E/S aislado y E/S por mapas de memoria.
- Para ejecutar una instrucción en un programa, el CPU primero busca y trae la instrucción, la decodifica y luego la ejecuta.
- Hay tres métodos para sincronizar el CPU con el dispositivo E/S: E/S programada, E/S manejada por interrupciones y DMA.
- Los dos diseños para la arquitectura de CPU son CISC y RISC.

5.9 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuáles son los tres subsistemas que forman una computadora?
2. ¿Cuáles son las partes de un CPU?
3. ¿Cuál es la función del ALU?
4. ¿Cuáles son los diferentes tipos de registros? Descríbalos.
5. ¿Cuál es la función de la unidad de control?
6. ¿Cuál es la diferencia entre una palabra y un byte?
7. ¿Cuál es la función de la memoria principal?
8. ¿Cómo se relaciona la aproximación de un megabyte con el número real de bytes?
9. ¿Qué tipo de representación de números se utiliza para representar direcciones de memoria?
10. ¿Cuál es la diferencia entre RAM y ROM?
11. ¿Cuál es la diferencia entre SRAM y DRAM?
12. Comente las diferencias entre PROM, EPROM y EEPROM.
13. ¿Cuál es el propósito de la memoria caché?
14. Comente los niveles de velocidad de la memoria y el tipo de memoria que usa cada uno.
15. Dé ejemplos de dispositivos E/S que no sean de almacenamiento.
16. ¿Cuáles son las dos clases principales de dispositivos de almacenamiento?
17. Describa los componentes físicos de un disco magnético.
18. ¿Cómo está organizada la superficie de un disco magnético?
19. ¿Qué factores afectan el rendimiento de un disco magnético?
20. Describa los componentes físicos de una cinta magnética.
21. ¿Cómo está organizada la superficie de una cinta magnética?
22. ¿Cómo se accede a los datos en una cinta magnética?
23. Mencione cinco tipos de dispositivos de almacenamiento óptico.
24. ¿Quién escribe los datos en un CD-ROM?, ¿en un CD-R?, ¿en un CD-RW?

25. ¿Cuál es la ventaja del CD-RW sobre el CD-ROM y el CD-R?
26. Compare y contraste los agujeros y las superficies en los tres tipos de discos compactos.
27. Compare y contraste la lectura de los datos en los tres tipos de discos compactos.
28. ¿Cómo se borran los datos en un CD-RW?
29. ¿En qué difiere un DVD de un disco compacto?
30. ¿Cuáles son las funciones de los tres buses que conectan el CPU con la memoria?
31. ¿Cuál es la función de los controladores de dispositivos E/S?
32. ¿Qué es un controlador SCSI?
33. ¿Qué es la interfaz FireWire?
34. ¿Qué es el controlador USB?
35. Compare y contraste los dos métodos para manejar el direccionamiento de dispositivos E/S.
36. ¿Cuáles son los pasos en un ciclo de máquina?
37. Compare y contraste los tres métodos para manejar la sincronización del CPU con los dispositivos E/S.
38. Compare y contraste la arquitectura CISC con la arquitectura RISC.

PREGUNTAS DE OPCIÓN MÚLTIPLE

39. El _____ es un subsistema de computadora que realiza operaciones con los datos.
    - **a.** CPU
    - **b.** memoria
    - **c.** hardware de E/S
    - **d.** ninguno de los anteriores
40. _____ es una localidad de almacenamiento independiente que aloja los datos temporalmente.
    - **a.** Un ALU
    - **b.** Un registro
    - **c.** Una unidad de control
    - **d.** Una unidad de cinta
41. _____ es una unidad que puede sumar dos entradas.
    - **a.** Un ALU
    - **b.** Un registro
    - **c.** Una unidad de control
    - **d.** Una unidad de cinta
42. Un registro en un CPU puede alojar _____.
    - **a.** datos
    - **b.** instrucciones
    - **c.** valores del contador de programa
    - **d.** todos los anteriores
43. Una unidad de control con cinco líneas de control puede definir hasta _____ operaciones.
    - **a.** 5
    - **b.** 10
    - **c.** 16
    - **d.** 32
44. Una palabra es _____ bits.
    - **a.** 8
    - **b.** 16
    - **c.** 32
    - **d.** cualquiera de los anteriores
45. Si el espacio de direccionamiento de la memoria es 16 MB y el tamaño de palabra es ocho bits, entonces se requieren _____ bits para tener acceso a cada palabra.
    - **a.** 8
    - **b.** 16
    - **c.** 24
    - **d.** 32
46. Los datos en _____ se borran si la computadora se apaga.
    - **a.** RAM
    - **b.** ROM
    - **c.** una unidad de cinta
    - **d.** un CD-ROM
47. _____ es un tipo de memoria con capacitores que necesitan refrescarse periódicamente.
    - **a.** SRAM
    - **b.** DRAM
    - **c.** ROM
    - **d.** todas las anteriores
48. _____ es un tipo de memoria con puertas flip-flop tradicionales para alojar datos.
    - **a.** SRAM
    - **b.** DRAM
    - **c.** ROM
    - **d.** todas las anteriores
49. Hay _____ bytes en 16 terabytes.
    - **a.** 2^{16}
    - **b.** 2^{40}
    - **c.** 2^{44}
    - **d.** 2^{56}
50. La _____ puede programarse y borrarse usando impulsos electrónicos, pero puede permanecer en una computadora durante el borrado.
    - **a.** ROM
    - **b.** PROM
    - **c.** EPROM
    - **d.** EEPROM

51. La _____ es un tipo de memoria en la cual el usuario, no el fabricante, almacena programas que no pueden sobrescribirse.
- **a.** ROM
- **b.** PROM
- **c.** EPROM
- **d.** EEPROM

52. Los registros del CPU deben tener memoria de _____.
- **a.** alta velocidad
- **b.** velocidad media
- **c.** baja velocidad
- **d.** cualquiera de las anteriores

53. La memoria principal en una computadora por lo general consiste de grandes cantidades de memoria de _____.
- **a.** alta velocidad
- **b.** velocidad media
- **c.** baja velocidad
- **d.** cualquiera de las anteriores

54. La memoria _____ contiene una copia de una porción de la memoria principal.
- **a.** CPU
- **b.** caché
- **c.** principal
- **d.** ROM

55. _____ es un dispositivo E/S que no es de almacenamiento.
- **a.** el teclado
- **b.** el monitor
- **c.** la impresora
- **d.** todos los anteriores

56. Un _____ es un dispositivo de almacenamiento óptico.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

57. El _____ es un dispositivo de almacenamiento en el cual el fabricante escribe información en el disco.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

58. El _____ es un dispositivo de almacenamiento en el cual el usuario puede escribir información sólo una vez en el disco.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

59. El _____ es un dispositivo de almacenamiento que puede sufrir múltiples escrituras y borrados.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

60. El área de almacenamiento más pequeña en un disco magnético a la cual se puede tener acceso a la vez es _____.
- **a.** pista
- **b.** sector
- **c.** frame
- **d.** cabeza

61. Para un disco magnético, el tiempo de _____ es el tiempo que le toma a la cabeza de lectura/escritura moverse a la pista deseada donde se almacenan los datos.
- **a.** rotación
- **b.** búsqueda
- **c.** transferencia
- **d.** localidad

62. La resina de policarbonato se utiliza en los _____.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

63. En un _____, un rayo láser de alta potencia simula agujeros en una aleación de plata, indio, antimonio y telurio.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

64. En un _____, un rayo láser de alta potencia simula agujeros en la capa de tinte.
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** todos los anteriores

65. ¿Cuál dispositivo de almacenamiento óptico tiene la mayor capacidad?
- **a.** CD-ROM
- **b.** CD-R
- **c.** CD-RW
- **d.** DVD

66. En un DVD, un rayo _____ lee el disco.
- **a.** láser de alta potencia
- **b.** infrarrojo
- **c.** láser rojo
- **d.** láser azul

67. Un bus de _____ conecta el CPU y la memoria.
- **a.** datos
- **b.** direcciones
- **c.** control
- **d.** todos los anteriores

68. Si el tamaño de palabra es 2 bytes, se necesita un bus de datos con _____ líneas.
- **a.** 2
- **b.** 4
- **c.** 8
- **d.** 16

69. Si la memoria tiene 2^{32} palabras, el bus de direcciones necesita tener _____ líneas.
- **a.** 8
- **b.** 16
- **c.** 32
- **d.** 64

70. Un bus de control con ocho líneas puede definir _____ operaciones.
- **a.** 8
- **b.** 16
- **c.** 256
- **d.** 512

71. El controlador _____ presenta una interfaz paralela y una conexión en cadena para dispositivos E/S.
- **a.** SCSI
- **b.** FireWire
- **c.** USB
- **d.** IDE

72. El controlador _____ es un dispositivo serial que conecta dispositivos lentos como el teclado y el ratón a la computadora.
- **a.** SCSI
- **b.** FireWire
- **c.** USB
- **d.** IDE

73. El controlador _____ es una interfaz serial de alta velocidad que transfiere datos en paquetes.
- **a.** SCSI
- **b.** FireWire
- **c.** USB
- **d.** IDE

74. Los tres pasos en la ejecución de un programa en una computadora se realizan en este orden específico: _____.
- **a.** buscar y traer, ejecutar y decodificar
- **b.** decodificar, ejecutar y buscar y traer
- **c.** buscar y traer, decodificar y ejecutar
- **d.** decodificar, buscar y traer, y ejecutar

75. En el método _____ para sincronizar la operación del CPU con el dispositivo E/S, el dispositivo E/S informa al CPU cuando está listo para transferir datos.
- **a.** E/S programada
- **b.** E/S manejada por interrupciones
- **c.** DMA
- **d.** E/S aislado

76. En el método _____ para sincronizar la operación del CPU con el dispositivo E/S, el CPU está inactivo hasta que se termina la operación de E/S.
- **a.** E/S programada
- **b.** E/S manejada por interrupciones
- **c.** DMA
- **d.** E/S aislado

77. En el método _____ para sincronizar la operación del CPU con el dispositivo E/S, un bloque de datos grande puede pasarse directamente de un dispositivo E/S a la memoria.
- **a.** E/S programada
- **b.** E/S manejada por interrupciones
- **c.** DMA
- **d.** E/S aislado

EJERCICIOS

78. Una computadora tiene 64 MB (megabytes) de memoria. Cada palabra mide 4 bytes. ¿Cuántos bits se necesitan para asignar una dirección a cada palabra individual en la memoria?

79. ¿Cuántos bytes de memoria se necesitan para almacenar una pantalla de datos completa si la pantalla se forma de 24 líneas con 80 caracteres en cada línea? El sistema utiliza código ASCII, cada carácter ASCII es almacenado como un byte.

80. Una computadora imaginaria tiene cuatro registros de datos (R0 a R1), 1 024 palabras en la memoria y 16 instrucciones diferentes (sumar, restar, etc.). ¿Cuál es el tamaño mínimo de una instrucción en bits, si una instrucción típica utiliza el formato add 565 R2?

81. Si la computadora del ejercicio 80 utiliza el mismo tamaño de palabra para los datos y las instrucciones, ¿cuál es el tamaño para cada registro de datos?

82. ¿Cuál es el tamaño del registro de instrucciones de la computadora del ejercicio 80?

83. ¿Cuál es el tamaño del contador de programa de la computadora del ejercicio 80?

84. ¿Cuál es el tamaño del bus de datos en el ejercicio 80?

85. ¿Cuál es el tamaño del bus de direcciones en el ejercicio 80?

86. ¿Cuál es el tamaño mínimo del bus de control en el ejercicio 80?

87. Una computadora utiliza direccionamiento de E/S aislado. La memoria tiene 1 024 palabras. Si cada controlador tiene 16 registros, ¿a cuántos controladores se puede tener acceso mediante esta computadora?

88. Una computadora utiliza direccionamiento de E/S por mapas de memoria. El bus de direcciones utiliza 10 líneas (10 bits). Si la memoria está formada por 1 000 palabras, ¿a cuántos controladores de cuatro registros se puede tener acceso mediante esta computadora?

6 Redes de computadoras

En el capítulo anterior analizamos la organización de una computadora. El análisis se centró en la arquitectura de una computadora independiente. Sin embargo, en la actualidad las computadoras se conectan con frecuencia para formar una red, y las redes a su vez se conectan para formar un conjunto de redes interconectadas.

En este capítulo primero definimos una red de computadoras. Luego estudiamos el modelo OSI que define teóricamente cómo deben interactuar entre sí los componentes de una red. Enseguida analizamos las categorías de redes como LAN, MAN y WAN. Asimismo mostramos cómo conectar redes, mediante dispositivos de conexión, para formar un conjunto de redes interconectadas o una interred. Después analizamos Internet y el conjunto de protocolos TCP/IP que la controla, incluyendo algunas de las aplicaciones en Internet.

6.1 REDES, GRANDES Y PEQUEÑAS

Una **red de computadoras** es una combinación de sistemas (por ejemplo, una computadora) conectados mediante un medio de transmisión (por ejemplo, un alambre, un línea de control o el aire). Una red de computadoras puede abarcar un área geográfica pequeña, mediana o grande. En el primer caso, la red es una red de área local (LAN: *local area network*). En el segundo, la red es una red de área metropolitana (MAN: *metropolitan area network*), y en el tercero se trata de una red de área amplia (WAN: *wide area network*). Discutiremos las redes LAN, MAN y WAN más adelante en este capítulo. Estos tres tipos de redes también pueden conectarse por medio de dispositivos de conexión para formar un conjunto de redes interconectadas (o interred).

MODELO Y PROTOCOLO

En este capítulo, usamos con frecuencia dos términos: modelo y protocolo. Un **modelo** es la especificación establecida por una organización de estándares como un estándar para el diseño de redes. Un **protocolo**, por otra parte, es un conjunto de reglas que controla la interacción de diferentes dispositivos en una red o en un conjunto de redes interconectadas. La sección siguiente presenta la recomendación OSI como un modelo. Posteriormente, se define TCP/IP como el conjunto de protocolos oficial de Internet.

6.2 MODELO OSI

Para lograr que todos los componentes de una red o de un conjunto de redes interconectadas se coordinen correctamente, se requiere un modelo que muestre la relación entre los componentes y la función de cada componente. La **Interconexión de sistemas abiertos** (**OSI**: *Open Systems Interconnection*) es un modelo de éstos. El modelo OSI fue diseñado por la Organización para la Estandarización Internacional (ISO: *International Standard Organization*). En teoría, el modelo permite que dos sistemas distintos (por ejemplo, computadoras) se comuniquen sin importar su arquitectura subyacente.

El modelo de Interconexión Abierta de Sistemas (OSI) es un modelo teórico que muestra cómo dos sistemas diferentes cualesquiera pueden comunicarse entre sí.

SIETE CAPAS

El modelo OSI es un marco de referencia de siete capas que da a los diseñadores de redes una idea de la funcionalidad de cada capa independiente pero relacionada. Cada capa tiene un nombre: físico (capa 1), de enlace de datos (capa 2), de red (capa 3), de transporte (capa 4), de sesión (capa 5), de presentación (capa 6) y de aplicación (capa 7) (figura 6.1). El modelo OSI no establece que cada dispositivo involucrado en una red debe implementar las siete capas. Un dispositivo puede necesitar sólo una capa, dos capas, tres capas o todas las capas. El número de capas depende de la funcionalidad del dispositivo y de su ubicación en la red.

La figura 6.2 representa la función de las capas cuando un mensaje se envía desde un dispositivo A hasta un dispositivo B. Conforme el mensaje viaja de A a B, puede atravesar muchos **nodos** intermedios (no aparecen en la figura). Estos nodos intermedios por lo general involucran sólo las primeras tres capas del modelo OSI.

Como lo muestra la figura, antes de que los datos se envíen al medio de transmisión, éstos se desplazan hacia abajo a través de las siete capas hasta que llegan a la capa física. En cada capa se añade información de control a los datos en forma de **encabezados** (*headers*) o **caracteres de control** (*trailers*). Los encabezados se añaden a los datos en las capas 7, 6, 5, 4, 3 y 2. Los caracteres de control se añaden en la capa 2. En la máquina receptora, el encabezado o los caracteres de control se dejan caer en cada capa a medida que se desplazan hacia la séptima capa.

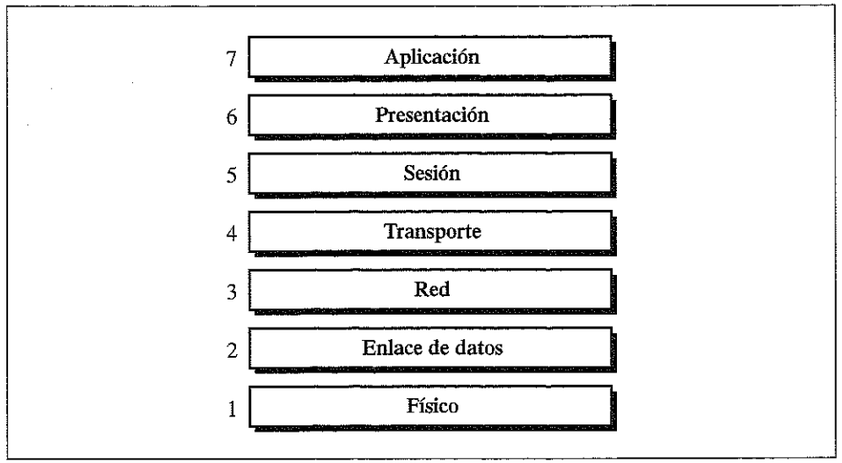


Figura 6.1 El modelo OSI

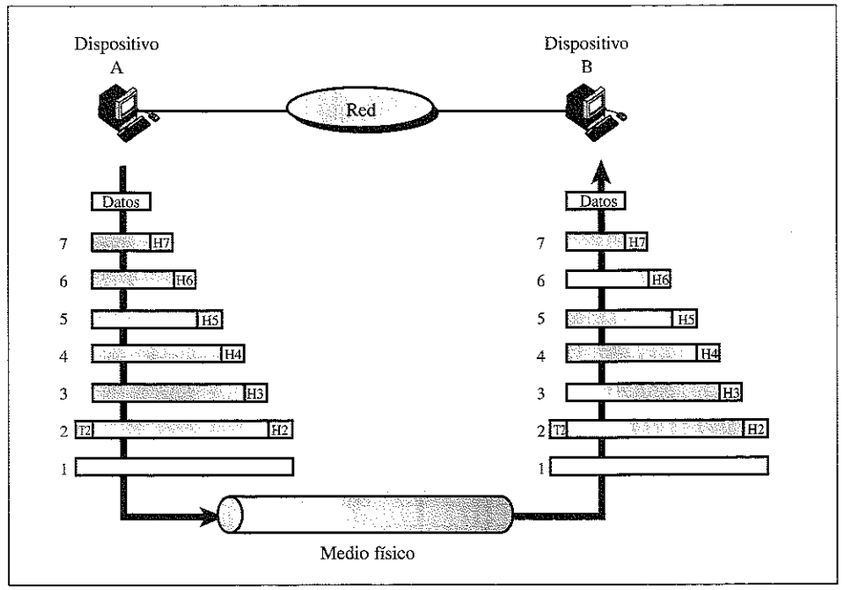


Figura 6.2 Flujo de datos en el modelo OSI

FUNCIONES DE LAS CAPAS

Esta sección describe brevemente las funciones de cada capa en el modelo OSI.

Capa física

La **capa física** es responsable de la transmisión de un flujo de bits a través de un medio físico. Codifica y decodifica los bits en grupos de bits. Luego transforma un flujo de bits en una señal. Las especificaciones físicas y mecánicas de los dispositivos físicos se determinan mediante la capa física.

Capa de enlace de datos

La **capa de enlace de datos** organiza los bits en unidades lógicas llamadas **bloques de datos** (*frames*)[1]. Un bloque de datos contiene información de la capa de red. La capa de enlace de datos añade un encabezado y caracteres de control para definir el bloque de datos para las estaciones receptora o intermedia. En particular, a los datos se añade información de direccionamiento, por lo general en forma de dos direcciones. Esto define las direcciones de dos estaciones contiguas, una que envía y otra que recibe. Observe que la capa de enlace de datos es responsable sólo de la **entrega nodo a nodo** del bloque de datos (de una estación a otra). Cuando una estación recibe un bloque de datos (no destinado para sí misma), cambia la dirección de origen a su propia dirección y la dirección de destino a la dirección de la estación siguiente. La capa de enlace de datos a menudo es responsable del manejo de errores entre dos estaciones contiguas. Los datos redundantes se añaden en los caracteres de control ya sea para detectar errores o para corregirlos.

Capa de red

La capa de enlace de datos se encarga de la entrega nodo a nodo de un bloque de datos entre dos estaciones contiguas; la **capa de red** es responsable de la entrega de un paquete (la unidad de datos manejada por la capa de red se llama *paquete*) entre el origen y el destino final. Para realizar esta tarea, la capa de red añade un encabezado a la unidad de datos proveniente de la capa superior que incluye, entre otras cosas, una dirección de origen y una dirección de destino. Estas direcciones comúnmente se llaman direcciones lógicas (o, como se verá posteriormente en este capítulo, direcciones IP) para distinguirlas de las direcciones físicas. Para la comunicación global, una dirección lógica debe ser única. Note que cuando un paquete se traslada del origen a un destino, la **dirección física** (añadida a la capa de enlace de datos) cambia de estación a estación, pero la dirección lógica permanece intacta del origen al destino.

Capa de transporte

La capa **de transporte** es responsable de la **entrega del origen al destino** (punto a punto) del mensaje completo. Observe la diferencia entre la responsabilidad de la capa de red y la capa de transporte. La capa de red es responsable de la entrega punto a punto de paquetes individuales. La capa de transporte, en cambio, es responsable de la entrega punto a punto de todo el mensaje. Un mensaje puede conformarse por uno o más paquetes. La capa de transporte es responsable de dividir el mensaje en varios paquetes y entregarlos a la capa de red, la cual envía los paquetes al exterior uno a uno, independientes uno del otro. Algunos paquetes pueden llegar desordenados a su destino, mientras que otros pueden perderse en el camino. La capa de transporte es responsable de asegurar que se transmita el mensaje completo desde el origen al destino. Si los paquetes se pierden, deben retransmitirse. Si los paquetes llegan desordenados, deben reorganizarse. En resumen, la capa de transporte considera al mensaje como una entidad integral que debe entregarse a la capa de transporte en el destino.

Capa de sesión

La capa **de sesión** está diseñada para controlar el diálogo entre los usuarios. Establece, mantiene y sincroniza el diálogo entre sistemas que se comunican. También añade lo que se llama **puntos de sincronización** para respaldar la entrega en caso de un fallo en el sistema o la red. Los puntos de sincronización dividen un mensaje largo en mensajes más pequeños y se aseguran de que el receptor reciba y reconozca cada sección. En este caso, si hay una falla en el sistema o la red, no es necesario que el emisor reenvíe el mensaje completo. El emisor puede desplazarse al último punto de sincronización y reenviar el mensaje desde ese punto. La mayoría de las implementaciones de red actuales no utiliza una capa de sesión separada. Si se requieren los servicios de una capa de sesión, por lo general se incluyen en la capa de aplicación.

Capa de presentación

La capa **de presentación** se ocupa de la sintaxis (formato) y la semántica (significado) de la información intercambiada entre dos sistemas. Se enfrenta al hecho de que diferentes sistemas utilizan métodos de codificación distintos (por ejemplo, ASCII y Unicode). Comprime y

[1] *N. del traductor.* En la literatura de computación a los bloques de datos también se les conoce como tramas.

descomprime los datos para un mejor rendimiento. Cifra y descifra los datos por razones de seguridad (véase el capítulo 16). Al igual que ocurre con la capa de sesión, actualmente la mayoría de las implementaciones no utiliza una capa de presentación. Esto no significa que las funciones definidas para la capa de presentación no sean necesarias, simplemente significa que las redes actuales han asignado estas responsabilidades a otras capas. Por ejemplo, el cifrado/descifrado se realiza tanto en la capa de red como en la capa de aplicación.

Capa de aplicación

La capa **de aplicación** permite que el usuario, ya sea una persona o software, tenga acceso a la red. Define aplicaciones comunes que pueden implementarse para simplificar el trabajo del usuario. Analizaremos algunas de estas aplicaciones más adelante en este capítulo en la sección sobre Internet.

6.3 CATEGORÍAS DE REDES

Ahora que se tiene un modelo para la comunicación, podemos dividir las redes en tres categorías principales: redes de área local (LAN), redes de área metropolitana (MAN) y redes de área amplia (WAN) (figura 6.3).

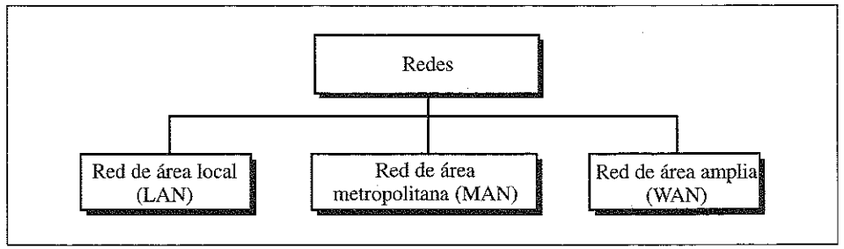


Figura 6.3 Categorías de redes

RED DE ÁREA LOCAL (LAN)

Una **red de área local** (**LAN**) está diseñada para permitir el uso compartido de recursos (hardware, software y datos) entre computadoras. Una LAN puede definirse simplemente como una combinación de computadoras y dispositivos periféricos (por ejemplo, impresoras) conectados mediante un medio de transmisión (por ejemplo, línea de control). La figura 6.4 muestra tres ejemplos de LAN.

La primera LAN en la figura utiliza una **topología de bus** en la cual las computadoras se conectan a través de un medio común llamado bus. En esta configuración, cuando una estación envía un bloque de datos a otra computadora, todas las computadoras reciben el bloque de datos y revisan su dirección de destino. Si la dirección de destino en el encabezado del bloque de datos coincide con la dirección física de la estación, el bloque de datos se acepta y los datos contenidos en él se procesan; de lo contrario, el bloque de datos se desecha. Un problema importante en este tipo de **topología** es la eliminación del bloque de datos. Una topología de bus utiliza terminadores de línea de control diseñados para dar fin electrónicamente a la señal que reciben. Si los terminadores de línea de control no están funcionando, la señal rebotará de ida y vuelta entre los dos puntos de manera que cada estación la recibirá una y otra vez, lo cual es una situación poco deseable.

La segunda LAN en la figura utiliza una **topología de estrella** en la cual las computadoras se conectan a través de un concentrador (*hub*), un dispositivo que facilita la conexión, o un conmutador (*switch*), un concentrador sofisticado que controla el envío del bloque de da-

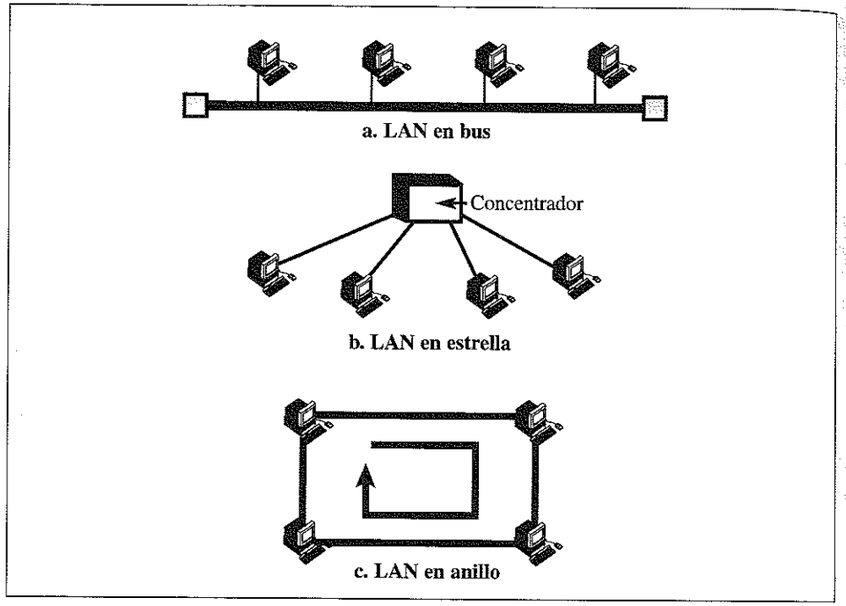


Figura 6.4 LAN

tos. Cuando se utiliza un concentrador, la LAN se comporta lógicamente como un bus. El concentrador sólo envía los datos fuera de todas sus interfaces. Cuando se utiliza un conmutador, éste revisa la dirección en el bloque de datos y el bloque se envía hacia fuera sólo a través de la interfaz del destino. En una topología de estrella, cada estación que recibe un bloque de datos es responsable de quitar el bloque de la red.

La tercera LAN utiliza una **topología de anillo**. En esta topología, cuando una computadora necesita enviar un bloque de datos a otra computadora, la envía a su computadora vecina. En ésta el bloque se regenera y se envía a la siguiente computadora vecina, repitiendo el proceso hasta que el bloque llega a su destino final. El destino abre el bloque, copia los datos y lo elimina del anillo o le añade un reconocimiento para enviarlo de regreso (a través del anillo) al emisor original. En este último caso, el emisor elimina el bloque de datos más tarde.

Las LAN instaladas en el pasado se configuraban en cualquiera de las topologías mencionadas; pero en la actualidad la topología dominante es la de estrella.

REDES DE ÁREA METROPOLITANA (MAN)

Una **red de área metropolitana** (**MAN**) utiliza servicios proporcionados por una empresa de comunicaciones (proveedor de servicios de red) común, tal como la compañía telefónica. Abarca una ciudad o un pueblo y ofrece sus servicios a usuarios individuales u organizaciones. Los usuarios individuales pueden conectar sus computadoras a la red y las organizaciones pueden conectar sus LAN a la red (figura 6.5). Muchas compañías telefónicas proveen un servicio de MAN generalizado llamado servicios de datos conmutados mutlimegabit (SMDS: *switched multimegabit data services*).

RED DE ÁREA AMPLIA (WAN)

Una **red de área amplia** (**WAN**) es la conexión de computadoras individuales o LAN distribuidas en una gran área (estado, país, el mundo). Las WAN, al igual que las MAN, están instaladas y administradas por empresas de comunicaciones comunes (figura 6.6). Observe que una persona que usa una línea telefónica para conectarse a un proveedor de servicios de Internet (PSI) utiliza una WAN. El PSI negocia las cuotas de los servicios directamente con la compañía telefónica y recibe el pago de sus clientes (usuarios de Internet).

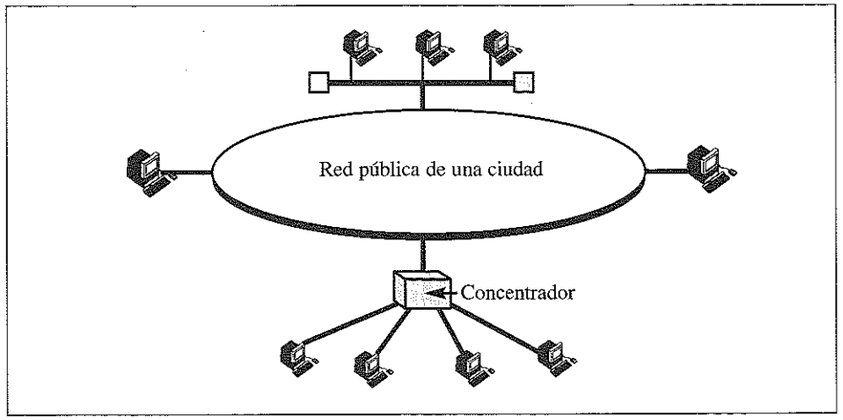


Figura 6.5 MAN

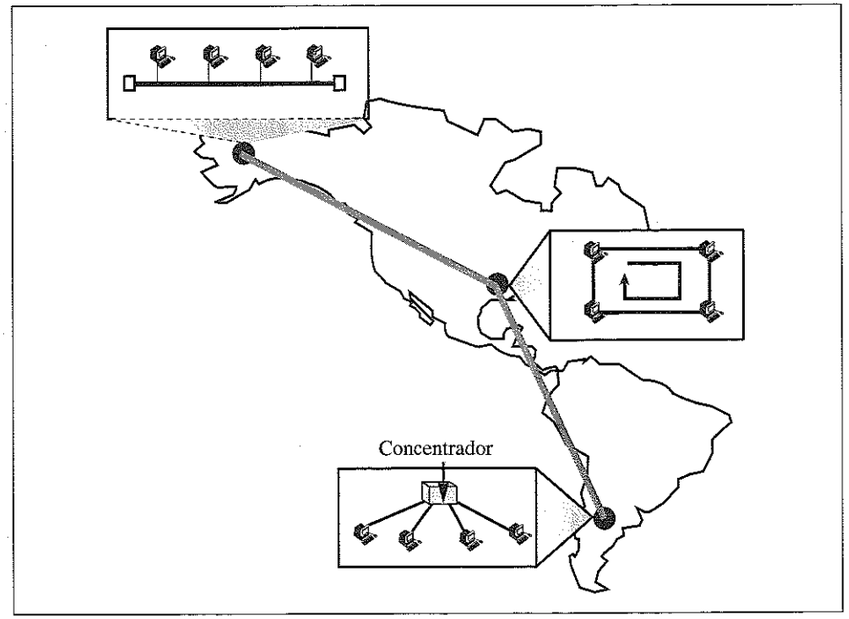


Figura 6.6 WAN

6.4 DISPOSITIVOS DE CONEXIÓN

Los tres tipos de redes que acabamos de presentar pueden conectarse usando **dispositivos de conexión**. La interconexión de redes hace posible la comunicación global desde un extremo del mundo al otro. Los dispositivos de conexión pueden dividirse en cuatro tipos con base en su funcionalidad según su relación con las capas del modelo OSI: repetidores, puentes, enrutadores y gateways. Los repetidores y los puentes por lo general conectan dispositivos en una

red. Los enrutadores y las gateways habitualmente interconectan redes en un conjunto de redes (interred) (figura 6.7).

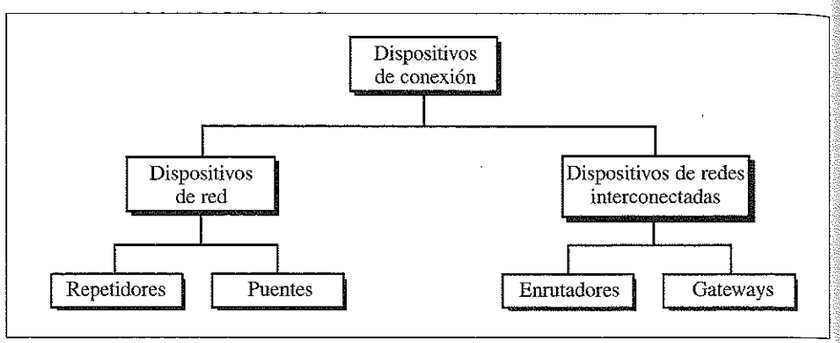


Figura 6.7 Dispositivos de conexión

REPETIDORES

Un **repetidor** es un dispositivo electrónico que regenera los datos. Extiende la longitud física de una red. Conforme se transmite una señal, ésta puede perder fuerza y un receptor puede interpretar una señal débil erróneamente. Un repetidor puede regenerar una señal y enviarla al resto de la red. La figura 6.8 muestra una red con y sin repetidor.

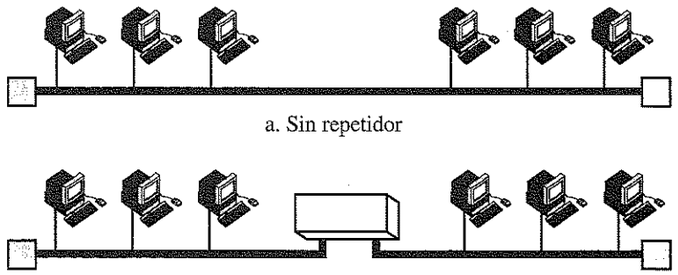


Figura 6.8 Repetidor

Los repetidores operan sólo en la capa física del modelo OSI. No reconocen las direcciones físicas ni lógicas. Simplemente regeneran cada señal que reciben. Los repetidores, cuyo uso se volvió generalizado cuando la topología de bus era la dominante, con frecuencia conectaban dos buses para ampliar la longitud de la red.

Los repetidores operan en la primera capa del modelo OSI.

PUENTES

Cuando una red utiliza una topología de bus, todas las estaciones comparten el medio. En otras palabras, cuando una estación envía un bloque de datos, esta estación ocupa el bus común y ninguna otra estación tiene permitido enviar un bloque de datos (si lo hace, los dos bloques colisionan). Esto implica una disminución en el rendimiento. Las estaciones necesitan esperar a que se libere el bus. Esto es similar a un aeropuerto que sólo tiene una pista de aterrizaje; cuando un avión utiliza la pista, otro avión listo para despegar debe esperar.

Un **puente** es un controlador de tráfico. Puede dividir un bus grande en segmentos más pequeños de manera que cada segmento sea independiente respecto al tráfico. Un puente instalado entre dos segmentos puede pasar o bloquear el paso de bloques de datos con base en la dirección de destino que tengan los mismos. Si se origina un bloque de datos en un segmento y la dirección de destino está en el mismo segmento, no hay razón para que el bloque pase el puente y viaje por otros segmentos. El puente utiliza una tabla para decidir si el bloque debe reenviarse a otro segmento. Con un puente, dos o más pares de estaciones pueden comunicarse al mismo tiempo (figura 6.9).

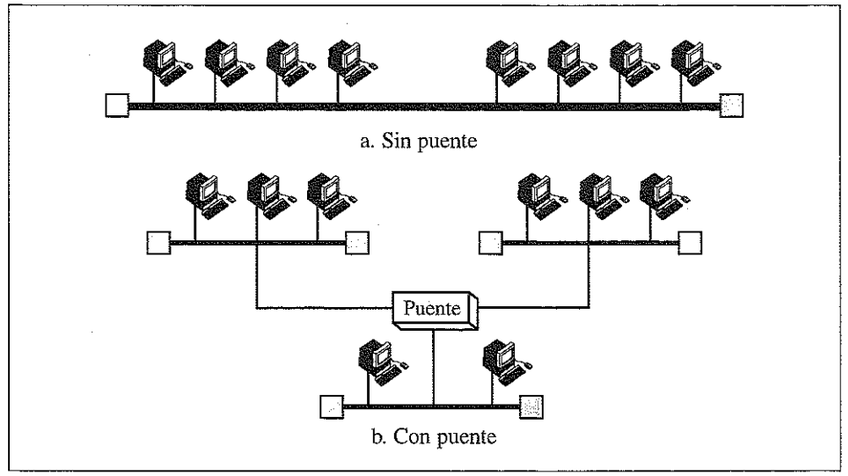


Figura 6.9 Puente

Además de sus deberes de control de tráfico, un puente también funciona como un repetidor al regenerar el bloque de datos. Como se vio anteriormente, esto significa que un puente opera en la capa física, pero debido a que necesita interpretar la dirección incrustada en el bloque de datos para tomar decisiones de filtrado, también opera en la capa de enlace de datos del modelo OSI.

Los puentes operan en las dos primeras capas del modelo OSI.

En años recientes, la necesidad de un mejor rendimiento ha conducido al diseño de un nuevo dispositivo llamado **conmutador** de segunda capa, el cual es simplemente un puente sofisticado con varias interfaces. Por ejemplo, una red con 20 estaciones puede dividirse en cuatro segmentos usando un puente de cuatro interfaces. O la misma red puede dividirse en 20 seg-

mentos (con una estación por segmento) usando un conmutador de 20 interfaces. En este caso, un conmutador aumenta el rendimiento; una estación que necesita enviar un bloque de datos los envía directamente al conmutador. Los medios no se comparten; cada estación se conecta directamente al conmutador (figura 6.10).

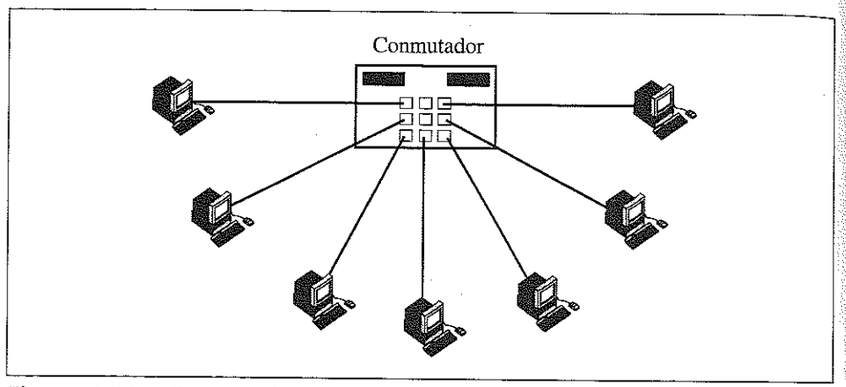


Figura 6.10 Conmutador

ENRUTADORES

Los **enrutadores** son dispositivos que conectan redes LAN, MAN y WAN. Un enrutador opera en la tercera capa del modelo OSI. Mientras que un puente filtra un bloque de datos con base en la dirección física (capa de enlace de datos) del bloque, un enrutador elige la ruta que seguirá un paquete con base en la dirección lógica (capa de red) del paquete.

> Los enrutadores operan en las primeras tres capas del modelo OSI.

En tanto que un puente puede conectar dos segmentos de un LAN o dos LAN pertenecientes a la misma organización, un enrutador puede conectar dos redes independientes: una LAN a una WAN, una LAN a una MAN, una WAN a otra WAN, y así sucesivamente. El resultado es un conjunto de redes interconectadas (o interred). Usted verá en breve que Internet (la única interred a escala mundial), la interred que conecta al mundo entero, es un ejemplo de un conjunto de redes interconectadas donde muchas redes se conectan entre sí por medio de enrutadores. La figura 6.11 muestra un ejemplo de un conjunto de redes interconectadas.

GATEWAYS

Comúnmente, una **gateway** es un dispositivo de conexión que se desempeña como convertidor de protocolo. Permite conectar entre sí dos redes, cada una con un conjunto diferente de protocolos para las siete capas OSI, y la comunicación entre ellas. Por lo general, una gateway es una computadora instalada con el software necesario. La gateway entiende los protocolos usados por cada red conectada y por tanto es capaz de traducir de uno a otro. Por ejemplo, una gateway puede conectar una red usando el protocolo AppleTalk a una red que utiliza el protocolo Novell Netware.

Hoy día, no obstante, los términos *gateway* y *enrutador* se utilizan de manera indistinta. Algunas personas se refieren a la gateway como un enrutador y otras se refieren a un enrutador como gateway. La distinción entre los dos términos está desapareciendo.

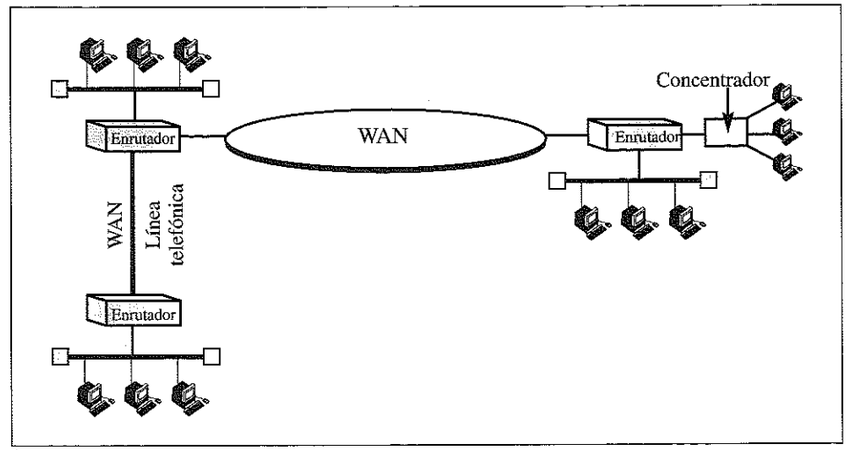


Figura 6.11 Enrutadores en una interred

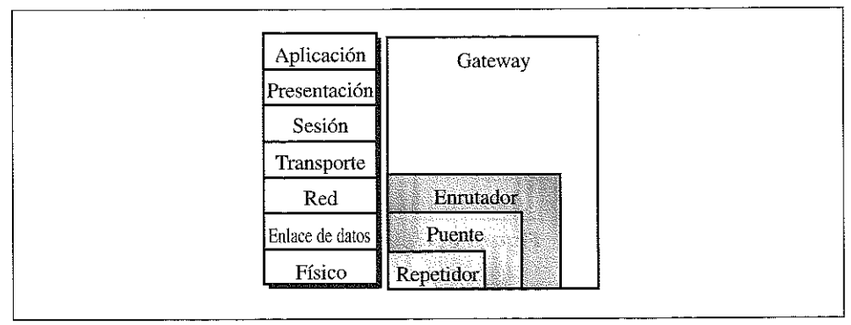


Figura 6.12 Dispositivos de conexión y el modelo OSI

MODELO OSI Y DISPOSITIVOS DE CONEXIÓN

La figura 6.12 presenta la relación entre los dispositivos de conexión y el modelo OSI.

6.5 INTERNET Y TCP/IP

Como se mencionó antes, usted puede conectar LAN, MAN y WAN individuales (usando enrutadores o gateways) para formar una red de redes, llamada **conjunto de redes interconectadas** o interred. En la actualidad, existen muchas interredes privadas y públicas, la más famosa de las cuales es **Internet** (con I mayúscula). Originalmente, Internet era un conjunto de redes interconectadas de investigación diseñado para conectar varias redes heterogéneas distintas. Estaba auspiciada por la Agencia de Proyectos de Investigación Avanzada para la Defensa (DARPA: *Defense Advanced Research Projects Agency*). Sin embargo, hoy día Internet es un conjunto de redes interconectadas que conecta millones de computadoras en todo el mundo.

El **protocolo de control de transmisión/protocolo Internet** (**TCP/IP**) es un conjunto o una pila de protocolos que controla a Internet de manera oficial. TCP/IP se desarrolló antes que el modelo OSI. Por lo tanto, las capas en el protocolo TCP/IP no coinciden exactamente con las del modelo OSI (figura 6.13).

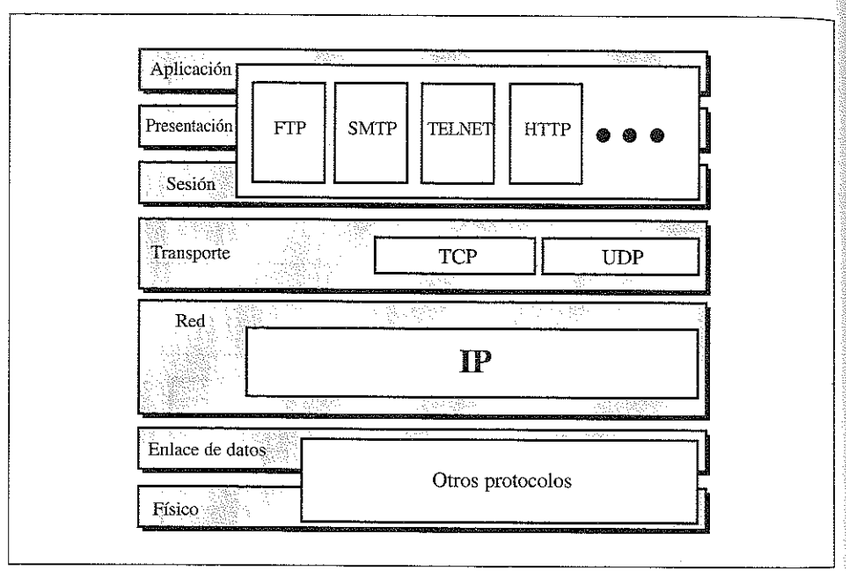


Figura 6.13 TCP/IP y el modelo OSI

CAPA FÍSICA Y DE ENLACE DE DATOS

En las capas físicas y de enlace de datos, TCP/IP no define ningún protocolo específico. Soporta todos los protocolos estándar y propietarios.

CAPA DE RED

En la capa de red (o más precisamente, en la capa de redes interconectadas o capa de red), TCP/IP soporta el **protocolo Internet** (**IP**). IP es un protocolo poco formal y un servicio de entrega del mejor esfuerzo. El término *mejor esfuerzo* significa que IP no proporciona verificación o seguimiento de errores. La unidad de datos en la capa IP se conoce como **datagrama IP**, un paquete independiente que viaja del origen al destino. **Datagramas** que pertenecen al mismo mensaje o a mensajes distintos pueden viajar a lo largo de diferentes rutas y pueden llegar sin un orden o duplicados. IP no sigue la pista de las rutas y no cuenta con la posibilidad para reordenar los datagramas una vez que éstos llegan.

Direccionamiento

TCP/IP requiere que cada computadora conectada a Internet se identifique mediante una dirección internacional única. Esta dirección a veces se conoce como dirección Internet o **dirección IP**.

Cada **dirección Internet** consiste de 4 bytes (32 bits). Para abreviar la forma de 32 bits y facilitar su lectura, las direcciones Internet por lo general se escriben en forma decimal con puntos decimales que separan los bytes: **notación con punto decimal**. La figura 6.14 muestra el patrón de bits y el formato decimal de una posible dirección.

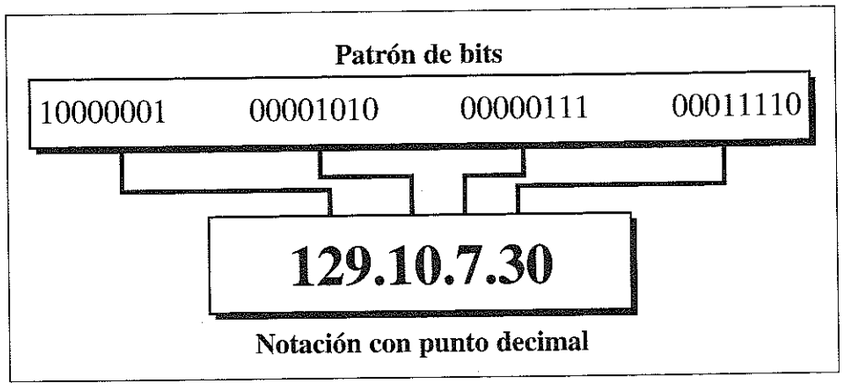


Figura 6.14 Direcciones IP en notación con punto decimal

CAPA DE TRANSPORTE

En la capa de transporte, TCP/IP define dos protocolos: el **protocolo de control de transmisión (TCP)** y el **protocolo de datagrama de usuario (UDP**: *user datagram protocol*). El protocolo de datagrama de usuario es el más simple de los dos. Es un protocolo de capa de transporte de punto a punto que proporciona sólo las necesidades básicas para la entrega punto a punto de una transmisión.

El Protocolo de Control de Transmisión provee servicios completos de capa de transporte para las aplicaciones. TCP es un protocolo de transporte confiable. Divide un mensaje en una secuencia de segmentos que se numeran en forma secuencial. Si un segmento se pierde, se envía de nuevo. Si un segmento se recibe fuera de orden, se ordena con la ayuda del mecanismo de numeración en secuencia.

CAPA DE APLICACIÓN

La capa de aplicación TCP/IP es equivalente a las capas de sesión, presentación y aplicación del modelo OSI combinados. Esto significa que todas las funcionalidades asociadas con estas tres capas se manejan en una sola capa, la capa de aplicación.

La comunicación en Internet utiliza el **modelo cliente-servidor** (figura 6.15). Un **cliente**, un programa de aplicación que se ejecuta en una máquina local, solicita un servicio de un **servidor,** un programa de aplicación que se ejecuta en una máquina remota. Por lo general, el programa servidor está siempre en ejecución y el programa cliente sólo se ejecuta cuando es necesario.

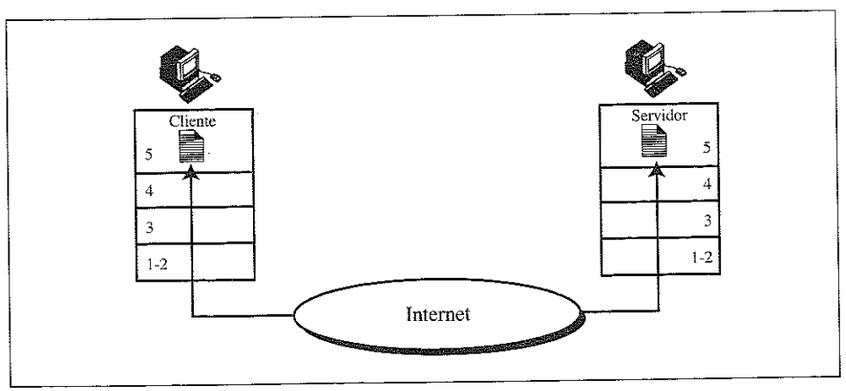


Figura 6.15 Modelo cliente-servidor

Protocolo de transferencia de archivos (FTP)

El protocolo estándar en Internet para transferir un archivo desde una máquina a otra es el **protocolo de transferencia de archivos** (**FTP**: *file transfer protocol*). FTP se diseñó para responder a los problemas tradicionales relativos a la transferencia de archivos. Uno de los problemas son los diferentes sistemas de codificación en uso; una máquina puede utilizar ASCII y la otra, Unicode. Otro problema son los distintos formatos de archivo en uso. FTP se diseñó para resolver estos problemas.

FTP establece dos conexiones entre las computadoras que se comunican: uno para la transferencia de datos y el otro para información de control (comandos y respuestas). La conexión de control está presente durante toda la sesión FTP; la conexión de datos está presente sólo cuando hay datos a transferir (figura 6.16).

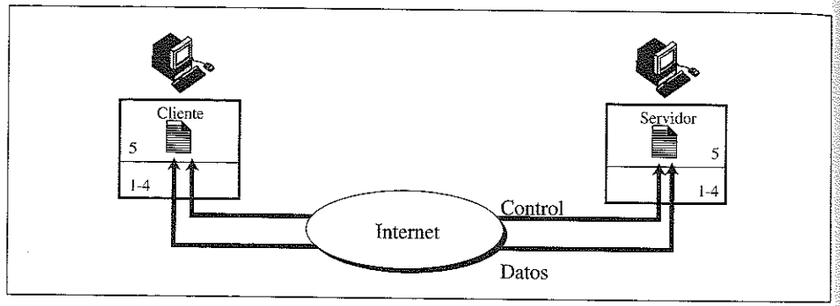


Figura 6.16 FTP

Protocolo simple de transferencia de correo (SMTP)

Por mucho, la aplicación más popular en Internet de hoy es el **correo electrónico** (**email**). El protocolo que soporta el correo electrónico en Internet es el **protocolo simple de transferencia de correo** (**SMTP**: *simple mail transfer protocol*) (figura 6.17). El servicio de correo electrónico por naturaleza es diferente de otras aplicaciones. Para enviar y recibir correo electrónico, el usuario debe instalar software SMTP tanto cliente como servidor en su computadora. Además, SMTP siempre debe ejecutarse para recibir correo electrónico. Una máquina apagada no recibirá correo electrónico. Para resolver este problema, SMTP siempre se utiliza con otro protocolo tal como el **protocolo de oficina postal** (**POP**: *post office protocol*). El usuario aún utiliza el cliente SMTP para enviar mensajes por correo electrónico. Sin embargo, los mensajes no se reciben directamente sino que en vez de ello se dirigen a otra computadora (la cual siempre está en ejecución). El servidor SMTP se ejecuta en esta computadora y recibe los mensajes y los almacena en el buzón de correo del usuario (un archivo especial). Cuando el usuario quiere recuperar el correo electrónico, utiliza el cliente POP para entrar en contacto con el servidor POP (en la misma computadora que ejecuta el servidor SMTP) y transfiere el correo electrónico. Normalmente hay una interfaz de usuario llamada agente de usuario (UA: *user agent*) que facilita estas transacciones.

Direcciones SMTP utiliza un sistema de direccionamiento único que consiste de dos partes: una *parte local* y un *nombre de dominio* separado por un signo @ (figura 6.18). La parte local define el nombre de un archivo especial, llamado buzón de correo del usuario, donde todo el correo recibido por un usuario se almacena para su recuperación posterior por el agente de usuario. El nombre de dominio define la computadora (a menudo de manera simbólica) que sirve como servidor SMTP.

TELNET

FTP y SMTP proporcionan servicios específicos para el usuario. FTP se utiliza para transferir archivos; SMTP se usa para enviar correo electrónico. Existen otros servicios que necesita un usuario en Internet. Por ejemplo, los estudiantes que toman un curso de programación necesitan tener acceso a una de las computadoras del laboratorio de la universidad para hacer la programación. Necesitan, además, crear un programa, compilarlo y ejecutarlo (véase el capítulo 9). Todas estas tareas deben realizarse en forma remota.

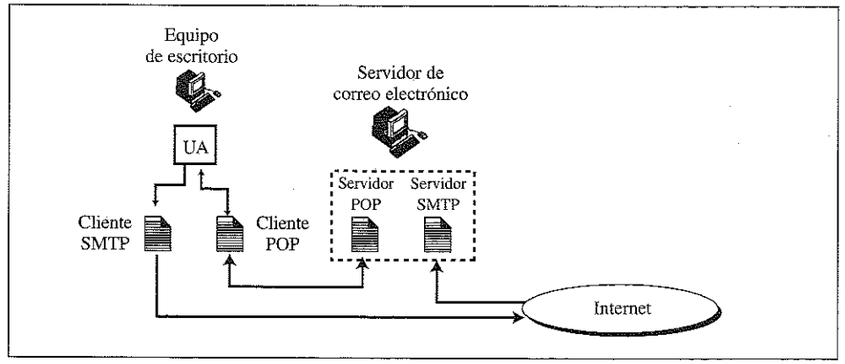


Figura 6.17 SMTP

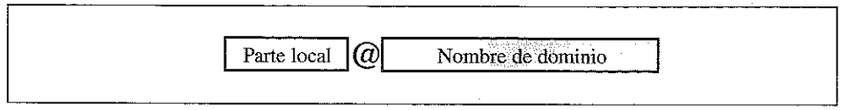


Figura 6.18 Direcciones de correo electrónico

Pero no existe un programa cliente-servidor en Internet llamado "crear un programa", "compilar un programa" o "ejecutar un programa". Actualmente hay muchas tareas diferentes que un usuario necesita realizar en una computadora remota. No hay manera de tener un programa cliente-servidor específico para cada tarea.

La solución es un programa cliente-servidor de uso general que permita a un usuario ejecutar un programa de aplicación en una computadora remota como si el usuario estuviera accediendo a esa computadora localmente. Cuando un usuario va a un laboratorio de computadoras y accede directamente a una computadora, se le llama **inicio de sesión local**. Por otro lado, cuando un usuario permanece en casa y accede a la misma computadora en forma remota, se le llama **inicio de sesión remoto.**

TELNET (*TErminaL NETwork*: red terminal) es un programa cliente-servidor general en Internet que permite el inicio de sesión remoto. TELNET permite el establecimiento de una conexión desde un sistema local a un sistema remoto de tal manera que la terminal local parece ser una terminal en el sistema remoto (figura 6.19).

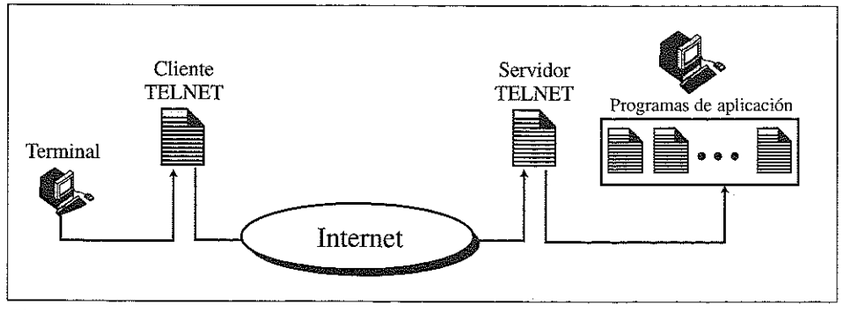


Figura 6.19 TELNET

Protocolo de transferencia de hipertexto (HTTP)

El **protocolo de transferencia de hipertexto** (**HTTP**: *hypertext transfer protocol*) es un programa cliente-servidor que se utiliza para tener acceso a y transferir documentos en el World Wide Web. Aunque transfiere los datos en forma de texto simple, hipertexto, audio, video y así por el estilo, está diseñado particularmente para transferir documentos de hipertexto (véase la sección siguiente).

Un cliente HTTP envía una solicitud al servidor. El servidor envía la respuesta al cliente. Los comandos del cliente al servidor se incrustan en un mensaje tipo carta. El contenido del archivo solicitado u otra información se incrusta en un mensaje de respuesta tipo carta.

Localizador uniforme de recursos (URL) HTTP utiliza un tipo especial de direccionamiento llamado **localizador uniforme de recursos** (**URL**: *uniform resource locator*), el cual es un estándar para especificar cualquier tipo de información en Internet. El URL define cuatro cosas: método, computadora anfitrión, puerto y ruta (figura 6.20).

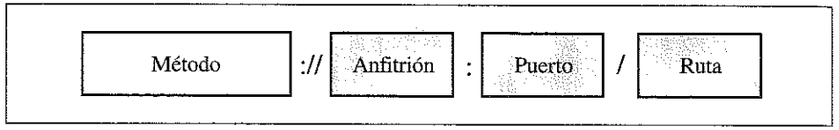


Figura 6.20 URL

El *método* es el programa cliente-servidor usado para transferir los documentos, HTTP en este caso (el URL puede utilizarse por otros programas de aplicación). El *anfitrión* es la computadora donde se localiza la información, aunque el nombre de la computadora puede ser un alias, que por lo general comienza con los caracteres "www".

El *puerto*, el cual define el número de puerto del servidor, es opcional. La *ruta* es el nombre de ruta del archivo donde se localiza la información. Observe que la ruta puede contener diagonales que, en el sistema operativo UNIX, separan los directorios de los subdirectorios y archivos.

World Wide Web (WWW)

El **World Wide Web** (**WWW**), o **Web**, se basa en la idea de información distribuida. En lugar de mantener toda la información en un lugar, cada entidad (individuo u organización) que tiene información para compartir almacena esa información en su propia computadora y permite el acceso a la misma a los usuarios de Internet. El WWW es una colección de documentos multimedia.

Hipertexto El WWW utiliza el concepto de hipertexto, el cual es un documento que contiene texto, palabras y frases especiales que pueden crear un vínculo a otros documentos que contienen texto, imágenes, audio o video. Un documento de hipertexto disponible en el Web se llama *página*. La página central de una organización o individuo se conoce como **página principal**.

Explorador Para obtener acceso a una página en el WWW, se necesita un **explorador** que por lo general consiste de tres partes: un controlador, un método y un intérprete (figura 6.21). El controlador es la parte medular del explorador; coordina todas las actividades. El método es un programa de aplicación cliente que recupera el documento. Aunque puede ser cualquiera de los programas de aplicación que hemos analizado, por lo general es HTTP. El intérprete despliega el documento en la pantalla.

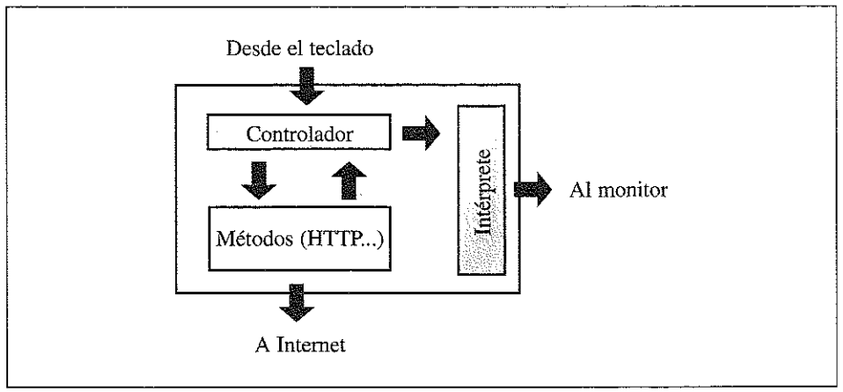


Figura 6.21 Explorador

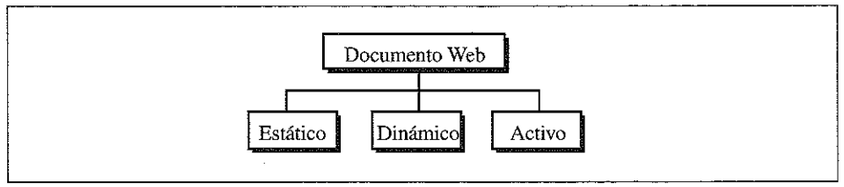


Figura 6.22 Categorías de documentos Web

Tipos de documentos Generalmente hay tres tipos distintos de documentos en Internet: estático, dinámico y activo (figura 6.22).

- Los **documentos estáticos** tienen contenido fijo. Se crean en el lado del servidor y sólo pueden copiarse. Un documento estático normalmente utiliza el lenguaje para marcado de hipertexto (HTML: *hypertext markup language*) para el formato de pantalla y los vínculos en el documento. En el capítulo 9 se estudia el HTML.

- Los **documentos dinámicos** son programas que residen en el lado del servidor. Cuando un explorador envía una solicitud el servidor ejecuta el programa y envía el resultado (no el programa en sí) al explorador. Por ejemplo, el explorador puede solicitar al servidor que ejecute el programa de fecha y envíe el resultado del programa al cliente. Los documentos dinámicos usan una tecnología llamada Interfaz de Compuerta Común (CGI: *common gateway interface*) que incluye lenguajes de programación como Perl (véase el capítulo 9) y HTML para manejar la creación y la interpretación del documento.

- Los **documentos activos** también son programas, pero no es posible ejecutarlos en el servidor. En vez de ello, el explorador debe solicitar la transferencia del programa. Después de la transferencia, el programa se ejecuta en el lado del explorador. La diferencia entre un documento dinámico y un documento activo es que el primero se ejecuta en el lado del servidor y su resultado se transfiere al explorador, y el último se transfiere al lado del explorador y luego se ejecuta. Un documento activo es necesario cuando el programa no puede ejecutarse en el lado del servidor. Por ejemplo, cuando un programa involucra animación, debe ejecutarse en el lado del explorador. Un documento activo por lo general se escribe en lenguaje Java (véase el capítulo 9). El explorador requiere un intérprete Java para interpretar (ejecutar) un documento activo.

6.6 TÉRMINOS CLAVE

bloque de datos
capa de aplicación
capa de enlace de datos
capa física
capa de presentación
capa de red
capa de sesión
capa de transporte
caracteres de control (*trailer*)
cliente
conjunto de redes interconectadas
conmutador
correo electrónico (*email*)
datagrama
datagrama IP
dirección física
dirección Internet
dirección IP
dispositivos de conexión
documento activo
documento dinámico
documento estático
encabezado
enrutador
entrega de nodo a nodo
entrega de origen a destino
explorador
gateway
inicio de sesión local
inicio de sesión remoto
Interconexión abierta de sistemas (OSI)
Internet
lenguaje para marcado de hipertexto (HTML)
Localizador Uniforme de Recursos (URL)
topología de bus
modelo
modelo cliente-servidor
nodo
notación con punto decimal
página principal
protocolo
Protocolo de Control de Transmisión (TCP)
Protocolo de Control de Transmisión/ Protocolo Internet (TCP/IP)
Protocolo de Datagrama de Usuario (UDP)
Protocolo de Transferencia de Archivos (FTP)
protocolo Internet (IP)
protocolo para transferencia de hipertexto (HTTP)
protocolo simple de transferencia de correo (SMTP)
puente
punto de sincronización
red de área amplia (WAN)
red de área local (LAN)
red de área metropolitana (MAN)
red de computadoras
repetidor
segmento
servidor
TELNET (red terminal)
topología
topología de anillo
topología de estrella
Web
World Wide Web (WWW)

6.7 RESUMEN

- Una red de computadoras es una combinación de dispositivos conectados mediante un medio de transmisión.
- El modelo de Interconexión de sistemas abiertos (OSI) es un modelo teórico que muestra cómo dos sistemas diferentes cualesquiera pueden comunicarse entre sí.
- Las siete capas del modelo OSI son la capa física, la capa de enlace de datos, la capa de red, la capa de transporte, la capa de sesión, la capa de presentación y la capa de aplicación.
- Una red de área local (LAN) permite el uso compartido de recursos (hardware, software y datos) entre computadoras.
- Una LAN puede configurarse en una topología de bus, anillo o estrella.
- Una red de área metropolitana (MAN) utiliza los servicios proporcionados por una empresa de comunicaciones común.
- Una red de área amplia (WAN) es la conexión de computadoras individuales o LAN sobre un área de gran extensión. Las WAN se instalan y ejecutan por empresas de comunicaciones comunes.
- Un repetidor es un dispositivo de conexión que regenera los datos y extiende la longitud física de una red.
- Un puente es un dispositivo de conexión que filtra el tráfico.
- Un enrutador es un dispositivo de conexión que enruta los paquetes.
- Una gateway permite a dos redes, cada una con un conjunto de protocolos completamente diferentes, comunicarse entre sí.
- Un conjunto de redes interconectadas es dos o más LAN, MAN o WAN.
- El Protocolo de Control deTransmisión/Protocolo Internet (TCP/IP) es el conjunto de protocolos utilizados por Internet, un conjunto de redes de computadoras interconectadas en todo el mundo.
- El Protocolo Internet (IP) es el protocolo poco confiable de TCP/IP en la capa de interred.
- Una dirección IP identifica a cada computadora conectada a Internet.
- El Protocolo de Datagrama de Usuario (UDP) y el Protocolo de Control de Transmisión (TCP) son los protocolos de TCP/IP en la capa de transporte.
- El Protocolo de Transferencia de Archivos (FTP) es una aplicación cliente-servidor para copiar archivos de un anfitrión a otro.
- El protocolo que soporta correo electrónico (*email*) en Internet es el Protocolo Simple de Transferencia de correo (SMTP).
- TELNET es una aplicación cliente-servidor que permite a un usuario iniciar una sesión en una máquina remota, dando al usuario acceso al sistema remoto.
- El Protocolo de Transferencia de Hipertexto (HTTP) es un programa cliente-servidor para tener acceso y transferir documentos en el World Wide Web (WWW), una colección de documentos multimedia.
- El Localizador Uniforme de Recursos (URL) es un identificador estándar para especificar información en Internet.
- Se requiere un explorador para tener acceso a una página en el WWW.
- Un documento en Internet puede clasificarse como estático, dinámico o activo.

6.8 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuál es la diferencia entre un modelo y un protocolo? Dé un ejemplo de cada uno.
2. ¿Cuáles son las capas del modelo OSI?
3. ¿Cuáles son las capas del conjunto de protocolos TCP/IP?
4. ¿Qué función tiene cada capa del modelo OSI?
5. ¿Cuál es la diferencia entre la entrega nodo a nodo y la entrega origen a destino?
6. ¿Cuál es la diferencia entre bloque de datos y paquete?
7. ¿Cuál es el propósito de un punto de sincronización?
8. ¿Cuáles son las tres topologías comunes? ¿Cuál de ellas es la más popular hoy en día?
9. Mencione cuatro tipos de dispositivos de conexión y sus funciones respectivas.
10. ¿Cuál es la diferencia entre TCP y UDP?
11. ¿Por qué las direcciones Internet son necesarias?
12. ¿Cómo difiere la capa de aplicación de TCP/IP de aquella del modelo OSI?
13. ¿Cuál es el propósito de FTP?
14. ¿Cuál es el propósito de TELNET?
15. ¿Cuál es el propósito de SMTP?
16. ¿Cuál es la diferencia entre un inicio de sesión local y un inicio de sesión remoto?
17. Compare y contraste los tres tipos de documentos Internet.

PREGUNTAS DE OPCIÓN MÚLTIPLE

18. Un _____ es un conjunto de reglas que controla la interacción de distintos dispositivos en una red o en un conjunto de redes interconectadas.
    a. modelo
    b. protocolo
    c. diálogo
    d. punto de sincronización
19. El modelo OSI tiene _____ capas.
    a. cinco
    b. seis
    c. siete
    d. cualquiera de las anteriores
20. La capa _____ del modelo OSI organiza los bits en unidades de datos lógicas llamadas bloques de datos (*frames*).
    a. física
    b. de enlace de datos
    c. de red
    d. de transporte
21. La capa _____ del modelo OSI permite a una persona tener acceso a Internet.
    a. de enlace de datos
    b. de transporte
    c. de aplicación
    d. física
22. La capa _____ del modelo OSI comprime y descomprime los datos.
    a. física
    b. de enlace de datos
    c. de sesión
    d. de presentación

23. La capa _____ del modelo OSI cifra los datos.

a. física
b. de enlace de datos
c. de sesión
d. de presentación

24. La capa _____ del modelo OSI transmite un flujo de bits sobre un medio físico.

a. física
b. de enlace de datos
c. de red
d. de transporte

25. La capa _____ del modelo OSI es responsable de la entrega nodo a nodo de un bloque de datos entre dos estaciones contiguas.

a. de transporte
b. de red
c. de enlace de datos
d. de sesión

26. La capa _____ del modelo OSI es responsable de la entrega de origen a destino del mensaje completo.

a. de transporte
b. de red
c. de enlace de datos
d. de sesión

27. La capa _____ del modelo OSI es responsable de la entrega de origen a destino de un paquete individual.

a. de transporte
b. de red
c. de enlace de datos
d. de sesión

28. La capa _____ del modelo OSI controla el diálogo entre los usuarios.

a. de transporte
b. de sesión
c. de presentación
d. de aplicación

29. A través de la capa _____ del modelo OSI los servicios de correo y de directorio están disponibles para los usuarios de red.

a. de conexión de datos
b. de sesión
c. de transporte
d. de aplicación

30. Los Postres Divinos de Cindy es una panadería con cuatro puntos de venta en San Francisco. Los puntos de venta necesitan comunicarse unos con otros. Este tipo de red probablemente es una _____.

a. LAN
b. MAN
c. WAN
d. ninguna de las anteriores

31. La compañía Papas Irlandesas de Kate tiene su base en Irlanda pero tiene sucursales en Boston y San Francisco. Las sucursales se comunican entre sí mediante una _____.

a. LAN
b. MAN
c. WAN
d. ninguna de las anteriores

32. Vecchiarelli Consultores ocupa dos áreas contiguas en el edificio Coffland. La red, que consiste en cuatro estaciones de trabajo y una impresora, probablemente es una _____.

a. LAN
b. MAN
c. WAN
d. ninguna de las anteriores

33. ¿Cuál es el nombre de dominio de la dirección de correo electrónico kayla@pit.arc.nasa.gov?

a. kayla
b. pit.arc.nasa.gov
c. kayla@pit.arc.nasa.gov
d. nasa.gov

34. ¿Cuál topología utiliza un concentrador o un conmutador?

a. bus
b. anillo
c. estrella
d. todas las anteriores

35. ¿Cuál topología necesita terminadores de cable?

a. bus
b. anillo
c. estrella
d. todas las anteriores

36. Un(a) _____ es un dispositivo de conexión que sólo regenera la señal.

a. repetidor
b. puente
c. enrutador
d. gateway

37. Un(a) _____ es un dispositivo de conexión que actúa como un convertidor de protocolo.

a. repetidor
b. puente
c. enrutador
d. gateway

38. Un _____ es un dispositivo de conexión que segmenta el tráfico.

a. repetidor
b. puente
c. enrutador
d. gateway

39. Un _____ es un dispositivo que puede enrutar un paquete con base en su dirección de capa de red.

a. puente
b. enrutador
c. repetidor
d. todos los anteriores

40. Un puente opera en _____ del modelo OSI.

a. la primera capa
b. las primeras dos capas
c. las primeras tres capas
d. todas las capas

41. Una gateway opera en _____ del modelo OSI.

a. la primera capa
b. las primeras dos capas
c. las primeras tres capas
d. todas las capas

42. La capa de _____ de TCP/IP tiene funciones similares a aquellas de las capas de aplicación, presentación y sesión del modelo OSI.

a. transporte
b. red
c. aplicación
d. sesión

43. La dirección IP actualmente mide _____ de longitud.

a. 4
b. 8
c. 32
d. ninguno de los anteriores

44. El protocolo de la capa de transporte de TCP/IP se llama _____.

a. TCP
b. UDP
c. IP
d. incisos a y b

45. _____ es un protocolo para transferencia de archivos.

a. FTP
b. SMTP
c. TELNET
d. HTTP

46. _____ es un protocolo para servicio de correo electrónico (email)

a. FTP
b. SMTP
c. TELNET
d. HTTP

47. _____ es un protocolo para tener acceso y transferir documentos en el WWW.

a. FTP
b. SMTP
c. TELNET
d. HTTP

48. Un documento _____ tiene contenido fijo.

a. estático
b. dinámico
c. activo
d. todos los anteriores

EJERCICIOS

49. Cuál o cuáles de las capas OSI están involucradas en cada una de las actividades siguientes:

a. envío de un bloque de datos a la siguiente estación
b. envío de un paquete desde el origen al destino
c. envío de un mensaje largo desde el origen al destino
d. inicio de una sesión en una computadora remota
e. cifrado y descifrado de datos
f. cambio de los datos de código de máquina a Unicode

50. Una pequeña parte de una LAN en bus con 200 estaciones está dañada. ¿A cuántas estaciones afecta este daño?

51. Una pequeña parte de una LAN en estrella con 200 estaciones está dañada. ¿A cuántas estaciones afecta este daño?

52. Una pequeña parte de una LAN en anillo con 200 estaciones está dañada. ¿A cuántas estaciones afecta este daño?

53. Si usted tiene una habitación cuadrada con una computadora en cada esquina, ¿cuál topología necesita menos cableado? Justifique su respuesta.

a. una LAN en bus
b. una LAN en anillo
c. una LAN en estrella con un concentrador en el centro de la habitación

54. Si usted tiene una habitación cuadrada con una computadora en cada esquina, ¿cuál topología es más confiable? Justifique su respuesta.

- **a.** una LAN en bus
- **b.** una LAN en anillo
- **c.** una LAN en estrella con un concentrador en el centro de la habitación

55. Un ingeniero observa que los datos que reciben las computadoras en los dos extremos de una LAN en bus contienen muchos errores. ¿Cuál cree que sea el problema? ¿Qué se puede hacer para resolver el problema?

56. Un ingeniero nota una gran cantidad de tráfico en una LAN en bus grande. ¿Qué puede hacer para mejorar la situación?

57. ¿Cuál es la ventaja de tener dos protocolos de transporte en TCP/IP?

58. ¿Cuáles son las ventajas y desventajas de tener una capa de aplicación en el conjunto de protocolos TCP/IP en vez de tres capas (sesión, presentación y aplicación) como en el modelo OSI?

59. Cambie las siguientes direcciones IP de notación de punto decimal a notación binaria:

- **a.** 112.32.7.28
- **b.** 129.4.6.8
- **c.** 208.3.54.12
- **d.** 38.34.2.1
- **e.** 255.255.255.255

60. Cambie las siguientes direcciones IP de notación binaria a notación de punto decimal:

- **a.** 01111110 11110001 01100111 01111111
- **b.** 10111111 11011100 11100000 00000101
- **c.** 0001111111110000 00111111 11011101
- **d.** 10001111 11110101 11000011 00011101
- **e.** 11110111 10010011 11100111 01011101

61. Explique el modelo cliente-servidor en Internet. ¿En cuál capa del conjunto de protocolos TCP/IP está implementado el modelo?

62. Separe la parte local y el nombre de dominio en las siguientes direcciones de correo electrónico:

- **a.** madeleine@belle.gov
- **b.** lindsey@jasmine.com
- **c.** wuteh@hunan.int
- **d.** honoris@queen.org

63. Explique la diferencia entre una dirección de correo electrónico y una dirección IP. ¿Existe una relación uno a uno entre las dos direcciones?

64. Explique la diferencia entre FTP y TELNET. ¿Cuándo se utiliza FTP y cuándo TELNET?

65. Un usuario utiliza un explorador para descargar un juego. ¿Qué tipo de documento descarga?

66. Un usuario utiliza un explorador para descargar un documento técnico. ¿Qué tipo de documento descarga?

67. Escriba un URL que utilice HTTP, el cual tenga acceso a un archivo con la ruta /user/general en una computadora con el alias www.hadb.

II Software de computadora

Sistemas operativos

Una computadora es un sistema formado por dos componentes importantes: hardware y software. El hardware de computadora es el equipo físico. El software es la colección de programas que permiten que el hardware realice su trabajo.

El **software** de computadora se divide en dos amplias categorías: el sistema operativo y los programas de aplicación (figura 7.1). Los programas de aplicación utilizan el hardware de computadora para resolver los problemas de los usuarios. El sistema operativo, por otro lado, controla el acceso al hardware por parte de los usuarios.

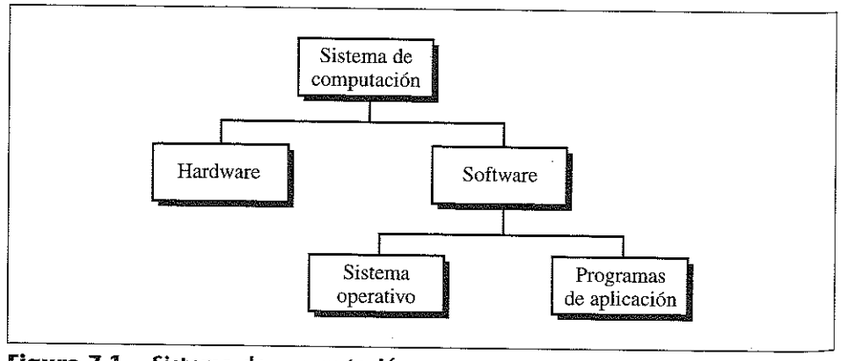


Figura 7.1 Sistema de computación

7.1 DEFINICIÓN

Un **sistema operativo** es tan complejo que es difícil dar una simple definición universal. En vez de ello, he aquí algunas definiciones comunes:

- Un sistema operativo es una interfaz entre el hardware de una computadora y el usuario (programas o personas).
- Un sistema operativo es un **programa** (o una serie de programas) que facilita la ejecución de otros programas.
- Un sistema operativo actúa como un gerente general que supervisa la actividad de cada componente en el sistema de computación. Como gerente general, el sistema operativo revisa que los recursos de hardware y software se utilicen de manera eficiente y cuando existe un conflicto al usar un recurso, el sistema operativo actúa como mediador para resolverlo.

Un sistema operativo es una interfaz entre el hardware de una computadora y el usuario (programas o personas) que facilita la ejecución de otros programas y el acceso a recursos de hardware y software.

Dos objetivos de diseño importantes de un sistema operativo son:

1. El uso eficiente del hardware.
2. Facilidad para usar los recursos.

7.2 EVOLUCIÓN

Los sistemas operativos han pasado por una larga historia de evolución, la cual resumimos a continuación.

SISTEMAS POR LOTES

Los **sistemas operativos por lotes** se diseñaron en la década de 1950 para controlar a las computadoras mainframe. En ese entonces, una computadora era una máquina grande que utilizaba tarjetas perforadas para entrada de datos, impresoras de línea para salida y unidades de cinta para medios de almacenamiento secundarios.

Cada programa a ejecutarse se llamaba **tarea**. Un programador que deseaba ejecutar una tarea enviaba una solicitud a la sala de operaciones junto con tarjetas perforadas para el programa y los datos. El programador no tenía ningún control o interacción con el sistema. Un operador procesaba las tarjetas perforadas. Si el programa era satisfactorio, el resultado se enviaba al programador; de lo contrario, se enviaba un listado del error.

Los sistemas operativos de esta época eran muy simples; sólo se aseguraban de que todos los recursos se transfirieran de una tarea a la siguiente.

SISTEMAS DE TIEMPO COMPARTIDO

Para utilizar los recursos del sistema de computación eficientemente, se introdujo la **multiprogramación**. La idea es mantener varias tareas en memoria y sólo asignar un recurso a una tarea que lo necesite con la condición de que el recurso esté disponible. Por ejemplo, cuando

un programa está usando un dispositivo de entrada/salida, el CPU está libre y puede ser utilizado por otro programa. La multiprogramación se estudia posteriormente en este capítulo.

La multiprogramación trajo la idea del **tiempo compartido**: los recursos pueden compartirse entre diferentes tareas. A cada tarea puede asignársele una parte del tiempo para usar el recurso. Como una computadora es mucho más rápida que una persona, el tiempo compartido es transparente para el usuario. Cada usuario tiene la impresión de que todo el sistema está trabajando para él o para ella.

La multiprogramación y finalmente el tiempo compartido mejoraron en gran medida la eficiencia de un sistema de computación. No obstante, ésta requería un sistema operativo más complejo. El sistema operativo ahora tenía que hacer una **planificación**: asignación de los recursos a los distintos programas y decidir cuál programa debería utilizar cuál recurso y cuándo.

Durante esta época, la relación entre una computadora y un usuario también cambió. El usuario podía interactuar directamente con el sistema sin pasar por un operador. Un nuevo término también se acuñó: **proceso**. Una tarea es un programa a ejecutar; un proceso es un programa que está en la memoria en espera de recursos.

SISTEMAS PERSONALES

Cuando las computadoras personales se introdujeron, había una necesidad de un sistema operativo de este tipo de computadora. Durante esta época, se introdujeron los **sistemas operativos monousuario** tales como DOS (*disk operating system*: sistema operativo de disco).

SISTEMAS PARALELOS

La necesidad de una mayor velocidad y eficiencia condujo al diseño de los **sistemas paralelos**: varios CPU en la misma máquina. Cada CPU podía utilizarse para servir a un programa o a una parte de un programa, lo cual significa que muchas tareas pueden lograrse en paralelo en vez de en forma serial. El sistema operativo para estos sistemas es más complejo que aquellos con un solo CPU.

SISTEMAS DISTRIBUIDOS

La conectividad en red y la interconectividad en red, como se vio en el capítulo 6, han creado una nueva dimensión en sistemas operativos. Una tarea realizada previamente por completo en una computadora ahora podía compartirse entre computadoras que podían estar a miles de millas de distancia. Un programa puede ejecutarse parcialmente en una computadora y parcialmente en otra si éstas están conectadas mediante un conjunto de redes interconectadas como Internet. Además, los recursos pueden distribuirse. Un programa puede necesitar archivos localizados en distintas partes del mundo. Los **sistemas distribuidos** combinan características de la generación anterior con nuevos servicios tales como el control de la seguridad.

7.3 COMPONENTES

En la actualidad, los sistemas operativos son muy complejos. Un sistema operativo necesita administrar diferentes recursos en un sistema de computadoras. Se parece a una organización con varios gerentes de alto nivel. Cada gerente es responsable de administrar su departamento, pero también necesita cooperar con otros y coordinar las actividades. Un sistema operativo moderno tiene al menos cuatro funciones: administrador de memoria, administrador de procesos, administrador de dispositivos y administrador de archivos. Al igual que muchas organizaciones que tienen un departamento, el cual no necesariamente está bajo las órdenes de algún administrador, un sistema operativo también tiene un componente semejante, el cual por lo general se conoce como interfaz de usuario o consola de comandos (*shell*). La interfaz de usuario es responsable de la comunicación del sistema operativo con el exterior (como un departamento de relaciones públicas). La figura 7.2 muestra los componentes de un sistema operativo.

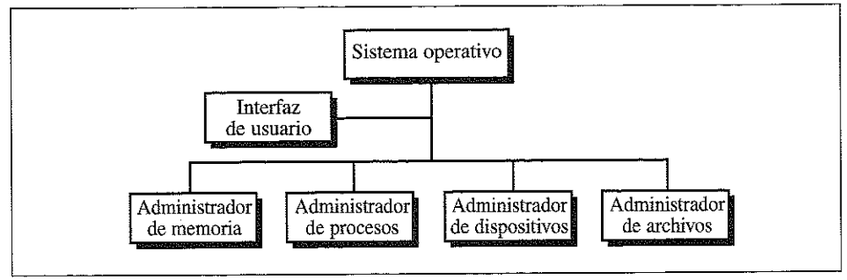


Figura 7.2 Componentes de un sistema operativo

ADMINISTRADOR DE MEMORIA

Una de las responsabilidades de un sistema de computación moderno es la administración de la memoria. Aun cuando en años recientes el tamaño de la memoria de las computadoras ha aumentado enormemente, también lo ha hecho el tamaño de los programas y los datos a ser procesados. La asignación de memoria debe ser administrada para evitar el síndrome de "quedarse sin memoria". Los sistemas operativos pueden dividirse en dos amplias categorías de administración de memoria: la monoprogramación y la multiprogramación.

Monoprogramación

La **monoprogramación** pertenece al pasado, pero vale la pena mencionarla puesto que le ayudará a comprender la multiprogramación. En la monoprogramación, la mayor parte de la capacidad de la memoria se dedica a un solo programa; únicamente se necesita una pequeña parte para alojar el sistema operativo. En esta configuración, todo el programa está en la memoria para su ejecución. Cuando el programa termina de ejecutarse, se reemplaza con otro programa (figura 7.3).

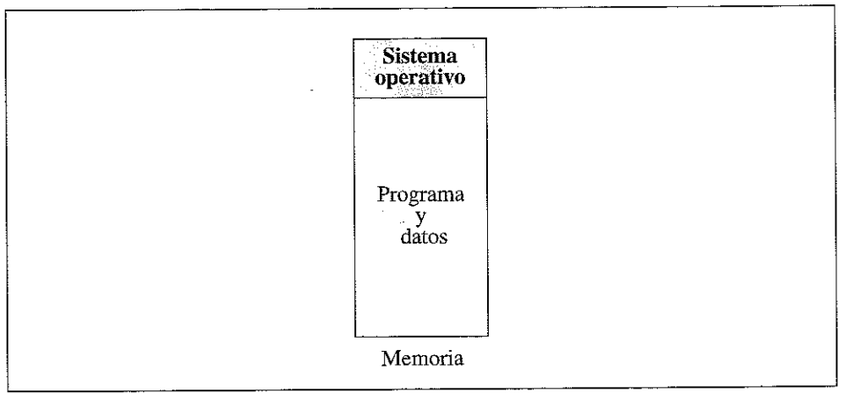


Figura 7.3 Monoprogramación

El **administrador de memoria** entra en acción justo aquí, ya que carga el programa en la memoria, lo ejecuta y lo remplaza con el programa siguiente. Sin embargo, hay varios problemas con esta técnica:

- El programa debe caber en la memoria. Si el tamaño de la memoria es menor que el tamaño del programa, éste no puede ejecutarse.

- Cuando un programa está en ejecución, ningún otro programa puede ejecutarse. Durante su ejecución, un programa a menudo necesita recibir datos de dispositivos de entrada y enviar datos a dispositivos de salida. Los dispositivos de entrada/salida en realidad son lentos comparados con el CPU. Así que cuando se llevan a cabo las operaciones de entrada/salida el CPU está inactivo. No puede atender a otro programa porque este otro programa no está en memoria. Se hace un uso poco eficiente de la memoria y del CPU.

Multiprogramación

En la multiprogramación, en la memoria hay más de un programa al mismo tiempo, los cuales se ejecutan concurrentemente. El CPU alterna entre los programas. La figura 7.4 muestra la memoria en un entorno de multiprogramación.

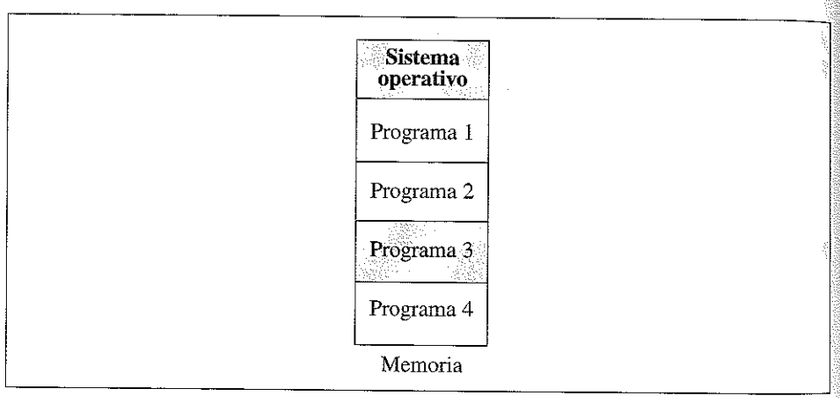


Figura 7.4 Multiprogramación

Desde la década de 1960, la multiprogramación ha sufrido varias mejoras que pueden verse en la taxonomía de la figura 7.5. En las secciones siguientes analizamos cada esquema de una manera muy breve. Dos técnicas pertenecen a la categoría de no intercambio; esto significa que el programa permanece en la memoria durante toda la ejecución. Las otras dos técnicas pertenecen a la categoría de intercambio. Esto significa que, durante la ejecución, el programa puede intercambiarse entre la memoria y el disco una o más veces.

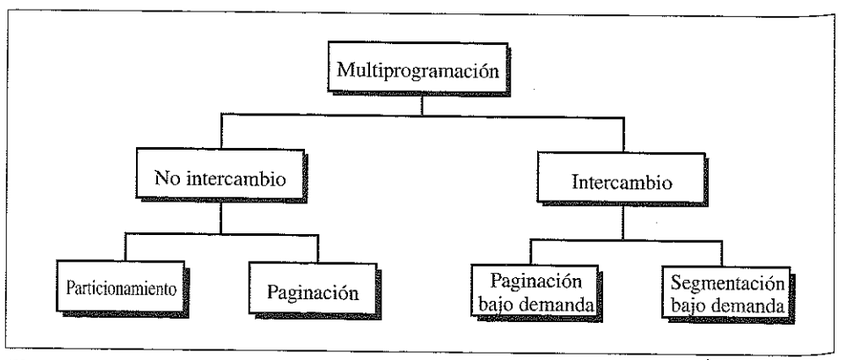
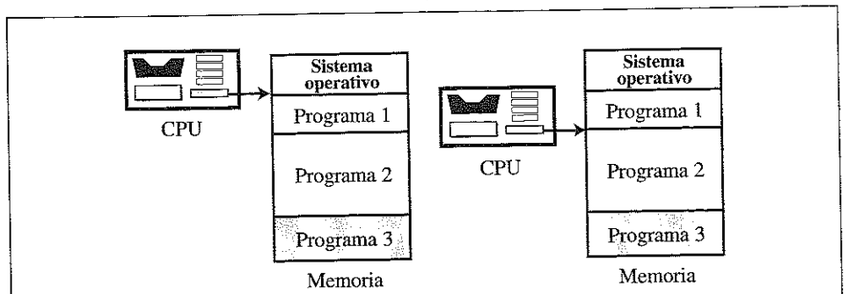


Figura 7.5 Categorías de la multiprogramación

Particionamiento La primera técnica usada en la multiprogramación se llama **particionamiento**. Bajo este esquema, la memoria se divide en secciones de longitud variable. Cada sección o partición aloja un programa. El CPU alterna entre los programas. Comienza con un programa. Ejecuta algunas instrucciones hasta que se encuentra una operación de entrada/salida o el tiempo asignado a ese programa expira. El CPU guarda la dirección de la localidad de memoria donde se ejecutó la última instrucción y se traslada al siguiente programa. El mismo procedimiento se repite con el segundo programa. Después de que se han atendido todos los programas, el CPU regresa al primer programa. Desde luego, puede haber niveles de prioridad en el acceso al CPU (figura 7.6).



a. El CPU inicia la ejecución del programa 1. b. El CPU inicia la ejecución del programa 2.

Figura 7.6 Particionamiento

Con esta técnica, cada programa está completamente en la memoria y ocupa localidades contiguas. El particionamiento mejora la eficiencia del CPU, pero aún hay otros problemas:

- El tamaño de las particiones debe ser determinado de antemano por el administrador de memoria. Si los tamaños de partición son pequeños, algunos programas no pueden cargarse en la memoria. Si los tamaños de partición son grandes, podría haber algunos espacios vacíos (localidades sin utilizar) en la memoria.

- Aun si el particionamiento es perfecto cuando la computadora se inicia, debe haber algunos espacios vacíos después de que los programas se remplazan con otro nuevos.

- Cuando hay muchos espacios vacíos, el administrador de memoria puede compactar las particiones para eliminar los espacios vacíos y crear particiones nuevas, pero esto crea una sobrecarga adicional en el sistema.

Paginación La **paginación** mejora la eficiencia del particionamiento. En la paginación, la memoria se divide en secciones de igual tamaño llamadas **bloques**. El programa se divide en secciones de igual tamaño llamadas **páginas**. El tamaño de una página y un bloque por lo general es el mismo y es igual al tamaño del bloque utilizado por el sistema para recuperar información de un dispositivo de almacenamiento (figura 7.7).

Una página se carga en un bloque en la memoria. Si un programa tiene tres páginas, ocupa tres bloques en la memoria. Con esta técnica, el programa no tiene que estar contiguo en la memoria. Dos páginas consecutivas pueden ocupar dos bloques en memoria no consecutivos.

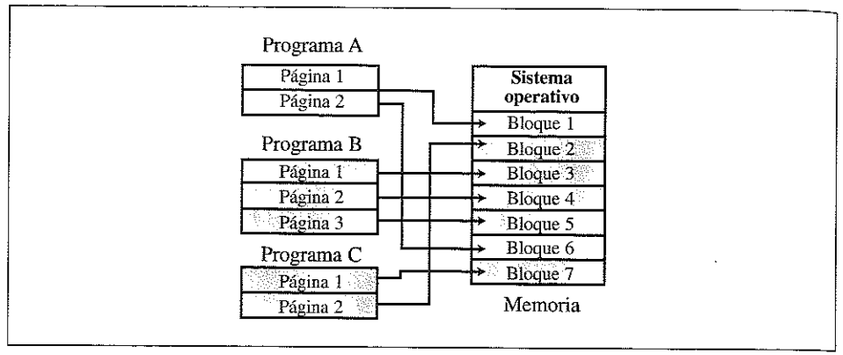


Figura 7.7 Paginación

La ventaja de la paginación sobre el particionamiento es que dos programas, cada uno de los cuales utiliza tres bloques no contiguos, pueden remplazarse por un programa que necesita seis bloques. No hay necesidad de que un nuevo programa espere hasta que seis bloques contiguos estén libres antes de ser cargados en la memoria.

La paginación mejora la eficiencia en cierto grado, pero el programa completo aún necesita estar en la memoria antes de ser ejecutado. Esto significa que un programa que necesita seis bloques no puede cargarse en la memoria si sólo hay cuatro bloques desocupados.

Paginación bajo demanda La paginación no requiere que el programa esté en localidades contiguas de la memoria, pero requiere que todo el programa esté en la memoria para su ejecución. La paginación bajo demanda ha eliminado esta última restricción, ya que en este tipo de paginación el programa se divide en páginas, pero las páginas pueden cargarse en la memoria una a una, ejecutarse y remplazarse por otra página. En otras palabras, la memoria puede alojar una página de varios programas al mismo tiempo. Además, no es necesario que en el mismo bloque estén cargadas páginas consecutivas del mismo programa. Una página puede cargarse en cualquier bloque libre.

Segmentación bajo demanda Una técnica similar a la paginación es la segmentación. En la paginación, un programa se divide en páginas de igual tamaño, lo cual no es la manera como piensa un programador, ya que éste piensa en términos de módulos, no de páginas de igual tamaño. Como verá en capítulos posteriores, un programa por lo general se compone de un programa principal y de subprogramas. En la **segmentación bajo demanda** el programa se divide en segmentos que coinciden con la idea del programador. Éstos se cargan en la memoria, se ejecutan, se remplazan por otros módulos del mismo programa o de un programa distinto.

Paginación y segmentación bajo demanda La **paginación y segmentación bajo demanda** incluso pueden combinarse para mejorar aún más la eficiencia del sistema. Un segmento puede ser demasiado grande para caber en cualquier espacio libre disponible en la memoria. La memoria puede dividirse en bloques y un módulo puede dividirse en páginas. Las páginas de un módulo pueden cargarse después en la memoria y ejecutarse una a una.

Memoria virtual

La paginación bajo demanda y la segmentación bajo demanda implican que parte del programa esté en la memoria principal y parte en el disco mientras el programa está en ejecución. Esto significa que, por ejemplo, una memoria que mida 10 MB puede ejecutar 10 programas, cada uno de 3 MB, para un total de 30 MB. En cualquier momento, los 10 MB de los 10 programas están en la memoria y 20 MB están en el disco. El tamaño real de la memoria es 10 MB pero hay una **memoria virtual** de 30 MB. La figura 7.8 muestra el concepto. La memoria virtual, la cual implica la paginación bajo demanda, la segmentación bajo demanda o ambas, se utiliza en casi todos los sistemas operativos actuales.

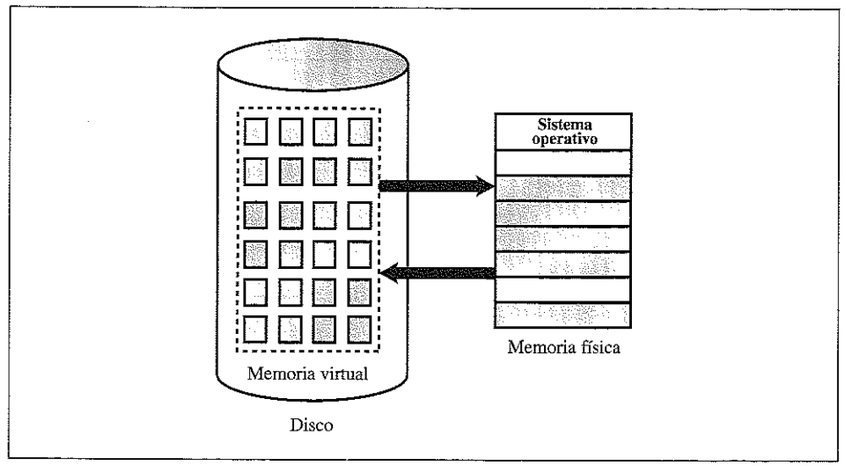


Figura 7.8 Memoria virtual

ADMINISTRADOR DE PROCESOS

Una segunda función de un sistema operativo es el administrador de procesos, pero antes de analizar este concepto, se definen algunos términos.

Programa, tarea y proceso

Los sistemas operativos modernos utilizan tres términos que se refieren a una serie de instrucciones: programa, tarea y proceso. Aunque la terminología es vaga y varía de un sistema operativo a otro, podemos definir estos términos de manera informal.

Programa Un programa es una serie de instrucciones no activas escritas por un programador y almacenadas en disco (o cinta). Esto puede o no convertirse en una tarea.

Tarea Un programa se vuelve una tarea a partir del momento en que éste se selecciona para su ejecución hasta que finaliza la misma y se convierte de nuevo en un programa. Durante este periodo, una tarea puede o no ejecutarse. Puede residir en el disco esperando a ser cargada en la memoria o puede residir en la memoria esperando que el CPU la ejecute. También puede residir en el disco o en la memoria en espera de un evento de entrada/salida o puede residir en la memoria cuando el CPU la esté ejecutando. El programa se comporta como una tarea en todas estas situaciones. Cuando un trabajo ha terminado de ejecutarse (ya sea normalmente o de manera anormal), se vuelve un programa y de nuevo reside en el disco. El sistema operativo ya no gobierna el programa. Observe que cada tarea es un programa, pero no todos los programas son tareas.

Proceso Un proceso es un programa en ejecución. Es un programa que ha iniciado pero no se ha terminado. En otras palabras, un proceso es una tarea que reside en la memoria. Se ha seleccionado entre otras tareas en espera y se ha cargado en la memoria. Un proceso puede ejecutarse en el momento o puede estar esperando tiempo del CPU. Mientras que la tarea esté en la memoria, es un proceso. Observe que cada proceso es una tarea, pero no todas las tareas son procesos.

Diagrama de estado

La relación entre un programa, una tarea y un proceso se esclarece si se considera cómo un programa se vuelve una tarea y cómo una tarea se vuelve un proceso. Esto puede ilustrarse con un **diagrama de estado** que muestra los diferentes estados de cada una de estas entidades. La figura 7.9 es un diagrama de estado que utiliza líneas divididas para los límites entre un programa, una tarea y un proceso.

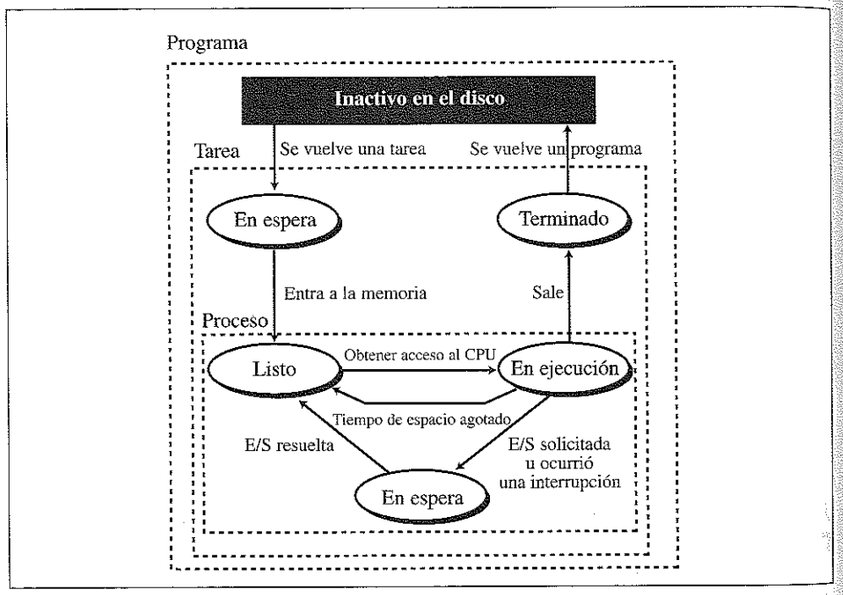


Figura 7.9 Diagrama de estado con los límites entre un programa, una tarea y un proceso

Un programa se vuelve una tarea cuando es seleccionado por el sistema operativo y llevado al **estado de espera**. Permanece en este estado hasta que puede cargarse en la memoria. Cuando hay espacio disponible en la memoria para cargar el programa total o parcialmente, la tarea cambia al **estado de listo.** Ahora se convierte en un proceso. Permanece en la memoria y en este estado hasta que el CPU puede ejecutarlo; cambia al **estado de ejecución** en este momento. Cuando está en el estado en ejecución, ocurre una de las tres cosas siguientes:

- El proceso se ejecuta hasta que necesita E/S.
- El proceso agota su espacio de tiempo asignado.
- El proceso termina.

En el primer caso el proceso va al **estado de espera** y espera hasta que la E/S se haya completado. En el segundo caso, va directamente al estado listo. En el tercer caso, va al **estado de terminación** y ya no es más un proceso. Un proceso puede moverse entre los estados de ejecución, espera y listo varias veces antes de que entre en el estado de terminación. Observe que el diagrama puede ser mucho más complejo si el sistema utiliza memoria virtual y alterna los programas dentro y fuera de la memoria principal.

Planificadores

Para cambiar una tarea o proceso de un estado a otro, el **administrador de procesos** utiliza dos planificadores: el planificador de tareas y el planificador de procesos.

Planificador de tareas El **planificador de tareas** cambia una tarea del estado de espera al estado de listo, o del estado de ejecución al estado de terminación. En otras palabras, un planificador de tareas es responsable de crear un proceso a partir de una tarea y de terminar un proceso. La figura 7.10 muestra el planificador de tareas con respecto al diagrama de estado.

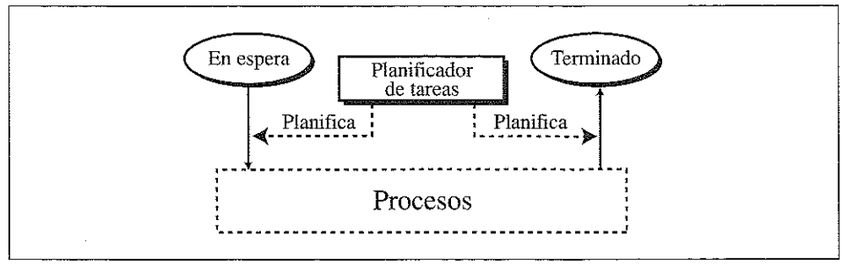


Figura 7.10 Planificador de tareas

Planificador de procesos El **planificador de procesos** cambia un proceso de un estado a otro. Cambia un proceso del estado de ejecución al estado de espera cuando el proceso está esperando que ocurra un evento. Cambia un proceso del estado de ejecución al estado de listo si ha expirado el tiempo asignado. Cambia el proceso del estado de espera al estado de listo cuando el evento ha ocurrido. Cuando el CPU está listo para ejecutar el proceso, el planificador de procesos cambia el proceso del estado de listo al estado de ejecución. La figura 7.11 muestra el planificador de procesos con respecto a un diagrama de estado.

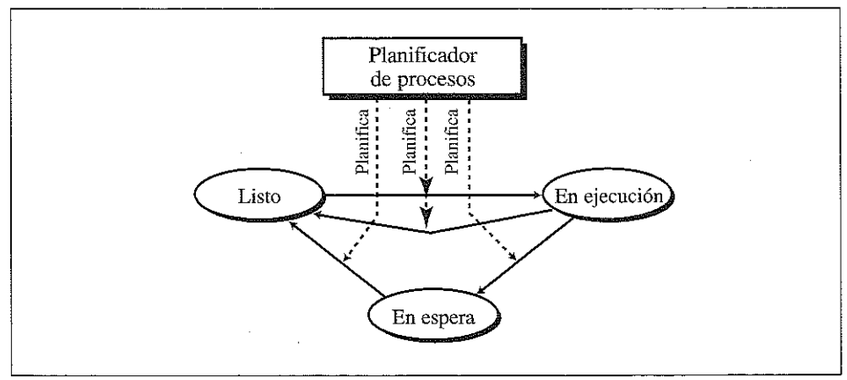


Figura 7.11 Planificador de procesos

Otros planificadores Algunos sistemas operativos utilizan otros tipos de planificadores para alternar entre procesos de una manera más eficiente.

Colas de espera

Nuestro diagrama de estado muestra una tarea o proceso que cambia de un estado a otro. En realidad, hay muchas tareas y muchos procesos que compiten unos con otros por los recursos de la computadora. Por ejemplo, cuando algunas tareas están en la memoria, otras deben esperar hasta que haya espacio disponible. O cuando un proceso está en ejecución usando el CPU, otros deben esperar hasta que el CPU esté libre. Para manejar varios procesos y tareas, el administrador de procesos utiliza **colas de espera** (listas de espera). Asociada con cada tarea o proceso hay un *bloque de control de tareas* o *bloque de control de procesos* que guarda información sobre esa tarea o proceso. El administrador de procesos almacena el bloque de control de tareas o de procesos en las colas de espera, en vez de la tarea o el proceso mismo. La tarea o el proceso permanece en la memoria o en el disco, es demasiado grande para ser duplicado en una cola de espera. El bloque de control de tareas o el bloque de control de procesos es el representante de la tarea o el proceso en espera.

Un sistema operativo puede tener varias colas de espera. Por ejemplo, la figura 7.12 muestra la circulación de las tareas y los procesos a través de tres colas de espera: la cola de tareas, la cola de listo y la cola de E/S. La cola de tareas aloja las tareas que están esperando memoria. La cola de listo aloja los procesos que están en la memoria, listos para su ejecución y esperando al CPU. La cola de E/S aloja los procesos que están esperando un dispositivo de entrada/salida (puede haber varias colas de E/S, una para cada dispositivo de entrada/salida, pero por razones de simplicidad sólo mostramos una).

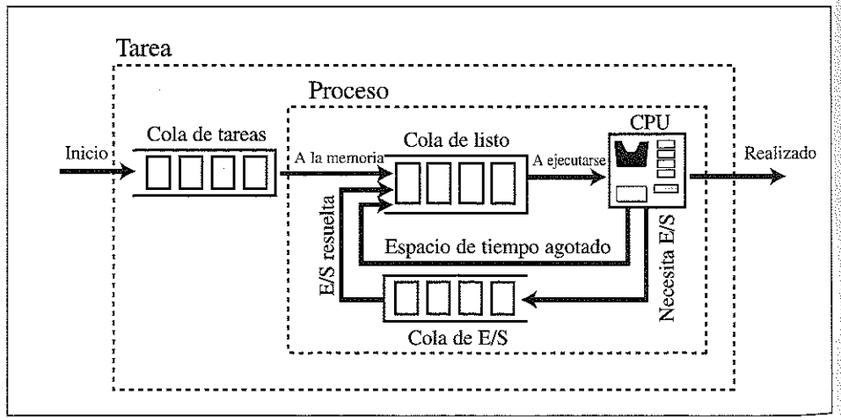


Figura 7.12 Colas para el administrador de procesos

El administrador de procesos puede tener diferentes políticas para seleccionar la siguiente tarea o proceso de una cola de espera; podría ser primero en llegar, primero en salir (FIFO: *first in, first out*), primero el más corto, aquel con mayor prioridad y así por el estilo.

Proceso de sincronización

La idea global que sustenta al administrador de procesos es sincronizar distintos procesos con diferentes recursos. Los recursos pueden utilizarse en cualquier momento por cualquier usuario (procesos en este caso); se pueden presentar dos situaciones: punto muerto y privación. A continuación se presenta una breve explicación de ambas situaciones.

Punto muerto En lugar de una definición formal de **punto muerto**, daremos un ejemplo. Suponga que hay dos procesos llamados A y B. El proceso A aguarda un archivo llamado Archivo 1 (el Archivo 1 se asignó a A) y no puede liberarse hasta que tenga otro archivo llamado Archivo 2 (A solicitó el Archivo 2). El proceso B aguarda al Archivo 2 (el Archivo 2 se asignó a B) y no puede liberarlo hasta que tenga el Archivo 1 (B ha solicitado el Archivo 1). Los archivos en la mayoría de los sistemas no son compartidos; cuando un proceso utiliza un archivo, este último no puede ser usado por otro proceso. Si no existe una condición en esta situación para forzar a un proceso a liberar un archivo, se crea un punto muerto (figura 7.13).

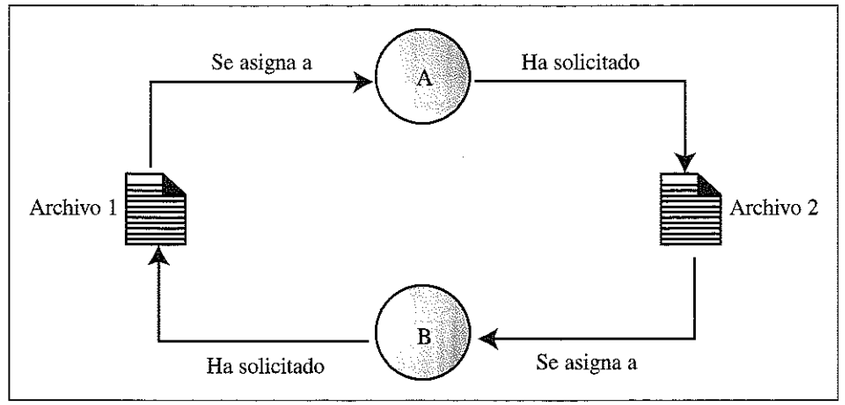


Figura 7.13 Punto muerto

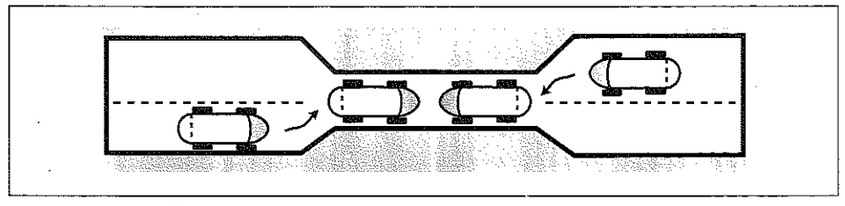


Figura 7.14 Punto muerto en un puente

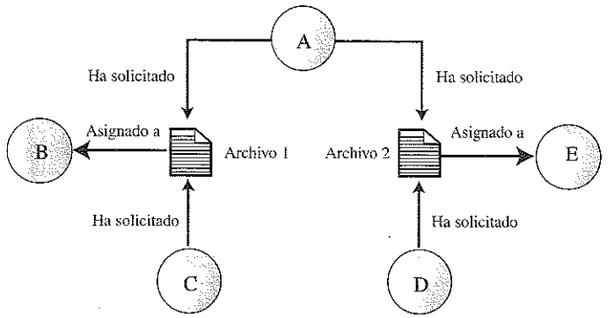
Como analogía, la figura 7.14 muestra un punto muerto en un puente estrecho. La situación es similar debido a que el recurso (parte del puente) está ocupado por un vehículo que no lo libera hasta que llega a la otra parte del puente, la cual está ocupada por el otro vehículo, y viceversa.

El punto muerto ocurre si el sistema operativo permite que empiece la ejecución de un proceso sin revisar primero si los recursos requeridos están listos y permite al proceso ocupar ese recurso todo el tiempo que lo desee. Debe haber alguna condición en el sistema para impedir los puntos muertos. Una solución es no permitir que un proceso inicie su ejecución hasta que los recursos se liberen, pero posteriormente verá que esta solución genera otro problema. La segunda solución es limitar el tiempo que un proceso puede ocupar un recurso.

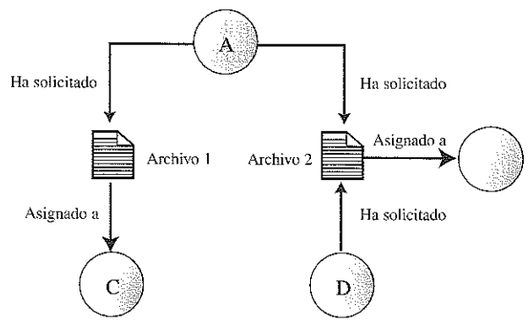
Los puntos muertos ocurren cuando el sistema operativo no pone restricciones de recursos a los procesos.

El punto muerto no siempre ocurre. Hay cuatro condiciones necesarias para un punto muerto: *exclusión mutua* (sólo un proceso puede ocupar un recurso), *ocupación de un recurso* (un proceso ocupa un recurso aun cuando éste no pueda utilizarse hasta que otros recursos estén disponibles), *sin derechos preferentes* (el sistema operativo no puede reasignar un recurso temporalmente) y *espera circular* (todos los procesos y recursos involucrados forman un ciclo, como en la figura 7.13). Se requiere que ocurran las cuatro condiciones para que haya un punto muerto. Sin embargo, estas condiciones sólo son condiciones necesarias (no suficientes), lo cual significa que deben estar presentes para que ocurra un punto muerto, pero podría no ser suficiente. En otras palabras, si falta una de estas condiciones, el punto muerto nunca ocurre. Esto le proporciona un método para impedir o evitar el punto muerto: No permita que alguna de estas condiciones ocurra.

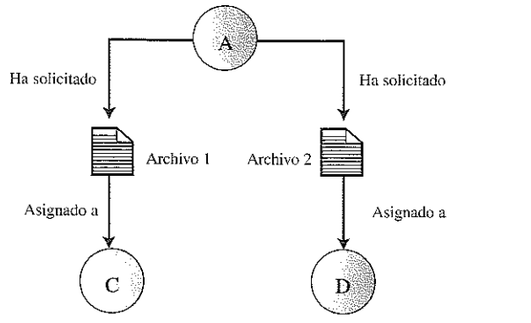
Privación La **privación** es lo opuesto al punto muerto. Puede ocurrir cuando el sistema operativo pone demasiadas restricciones a los recursos en un proceso. Por ejemplo, imagine un sistema operativo que especifica la posesión de todos los recursos requeridos antes de que un proceso pueda ejecutarse. En la figura 7.15 imagine que el proceso A necesita dos archivos, el Archivo 1 y el Archivo 2. El Archivo 1 está en uso por el proceso B y el Archivo 2, por el proceso E. El proceso B termina primero y libera al Archivo 1. El proceso A no puede iniciarse porque el Archivo 2 aún no está disponible. En este momento se permite la ejecución del proceso C, el cual necesita sólo al Archivo 1. Ahora el proceso E termina y libera al Archivo 2, pero el proceso A todavía no puede ejecutarse porque el Archivo 1 no está disponible.



a. El proceso A necesita tanto al Archivo 1 como al Archivo 2.



b. El proceso A aún necesita tanto al Archivo 1 como al Archivo 2.



c. El proceso A todavía necesita tanto al Archivo 1 como al Archivo 2 (privación).

Figura 7.15 Privación

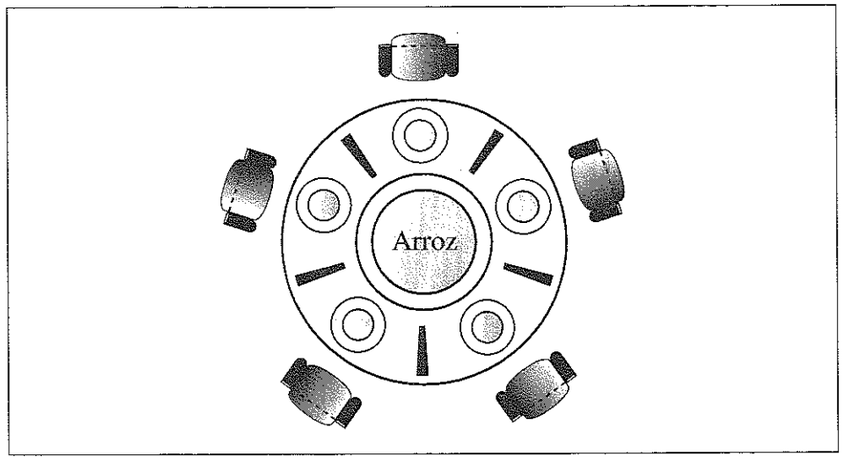


Figura 7.16 Cena de filósofos

Un problema de privación clásico es aquel presentado por Dijkstra. Cinco filósofos están sentados a una mesa redonda (figura 7.16). Cada filósofo necesita dos palillos para comer un tazón de arroz. Sin embargo, uno o ambos palillos podrían ser usados por su vecino. Un filósofo podría "pasar hambre" si no tiene dos palillos disponibles al mismo tiempo.

ADMINISTRADOR DE DISPOSITIVOS

El **administrador de dispositivos**, o administrador de entrada/salida, es responsable del acceso a los dispositivos de entrada/salida. Hay limitaciones en el número y la velocidad de los dispositivos de entrada/salida en un sistema de computación. Debido a que estos dispositivos son más lentos comparados con el CPU y la memoria, cuando un proceso accede a un dispositivo de entrada/salida, no está disponible para otros procesos durante un periodo. El administrador de dispositivos es responsable del uso eficiente de los dispositivos de entrada/salida.

Un análisis detallado del administrador de dispositivos requiere un conocimiento avanzado de los principios del sistema operativo y está más allá del ámbito de este libro. Sin embargo, podemos describir brevemente las responsabilidades del administrador de dispositivos:

- El administrador de dispositivos monitorea constantemente cada dispositivo de entrada/salida para asegurarse de que funcione apropiadamente. También el administrador debe saber cuándo un dispositivo ha terminado un proceso y está listo para tomar el siguiente en la lista.
- El administrador matiene constantemente una fila para cada dispositivo de entrada/salida o una o varias filas para dispositivos similares. Por ejemplo, si hay dos impresoras rápidas en el sistema, el administrador puede generar una fila para cada una, o una sola para ambas.
- El administrador controla las políticas de acceso a los dispositivos de entrada/salida. Por ejemplo, puede utilizar FIFO para un dispositivo y primero el más corto para otro.

ADMINISTRADOR DE ARCHIVOS

Hoy los sistemas operativos utilizan el **administrador de archivos** para controlar el acceso a éstos. Un análisis detallado del administrador de archivos requiere un conocimiento profundo de los principios de sistemas operativos, lo cual queda más allá de los propósitos de este libro. En el capítulo 13 se estudian algunos problemas relacionados con el acceso a archivos, pero esto no es adecuado para comprender la operación real de un administrador de archivos. A continuación se listan las principales responsabilidades de un administrador de archivos:

- El administrador de archivos controla el acceso a los archivos. El acceso está permitido sólo a aquellos que cuentan con permiso, y el tipo de acceso puede variar. Por ejemplo, un proceso (o un usuario que llama a un proceso) puede tener permiso para leer un archivo pero no para escribir (hacer cambios) en él. A otro proceso se le puede permitir ejecutar un archivo pero no se le permite ver el contenido.
- El administrador de archivos supervisa la creación, eliminación y modificación de los archivos.
- El administrador de archivos puede controlar la asignación de nombres a los archivos.
- El administrador de archivos supervisa el almacenamiento de los archivos: cómo se almacenan, dónde se almacenan y así por el estilo.
- El administrador de archivos es responsable de archivar y hacer los respaldos.

INTERFAZ DE USUARIO

Cada sistema operativo tiene una **interfaz de usuario**, la cual es un programa que acepta solicitudes de los usuarios (procesos) y las interpreta para el resto del sistema operativo. Una interfaz de usuario en algunos sistemas operativos, tales como UNIX, se conoce como **intérprete de comandos** (Shell). En algunos otros sistemas se le llama **ventana** para denotar que está manejada por menús y tiene un componente GUI (*graphical user interface*: interfaz gráfica de usuario).

7.4 SISTEMAS OPERATIVOS MÁS COMUNES

En esta sección presentamos algunos sistemas operativos comunes como una motivación para un estudio posterior. Elegimos tres sistemas operativos que son familiares para la mayoría de los usuarios de computadoras.

WINDOWS 2000

Windows 2000 (de Microsoft) es un buen ejemplo de un sistema operativo que ha avanzado un gran tramo en su evolución. Comenzó como una interfaz para ejecutarla con el DOS (sistema operativo de disco). Ahora es una sistema operativo complejo manejado por menús que incluye una GUI robusta, la cual utiliza memoria virtual que permite la multiprogramación. Tiene una capacidad de conectividad en red integral que lo vuelve un sistema operativo para redes. Windows 2000 también incluye una característica de seguridad que lo vuelven conveniente para situaciones donde la seguridad es un problema.

UNIX

UNIX es un sistema operativo conocido entre los programadores de computadoras y científicos de la computación. Es un sistema operativo muy poderoso con tres características sobresalientes. Primero, UNIX es un sistema operativo portátil que puede trasladarse de una plataforma a otra sin muchos cambios. La razón es que está escrito en su mayoría en lenguaje C (en lugar de un lenguaje de máquina para un sistema específico). Segundo, UNIX tiene una serie de utilerías (comandos) poderosas que pueden combinarse (en un archivo ejecutable llamado *script*) para resolver muchos de los problemas que en otros sistemas operativos requieren programación. Tercero, es independiente de los dispositivos debido a que incluye los controladores de los dispositivos en el sistema operativo mismo, lo cual significa que puede configurarse fácilmente para ejecutar cualquier dispositivo. En resumen, UNIX cuenta con todas las características que tiene un sistema operativo poderoso, incluyendo la multiprogramación, la memoria virtual y sistemas de directorios y archivos muy bien diseñados. La única crítica que se hace a UNIX con frecuencia es que los comandos son cortos y complicados para los usuarios normales; de hecho es muy conveniente para los programadores que necesitan comandos cortos.

LINUX

Linux, desarrollado por Linus Torvalds en Finlandia, está basado en UNIX. De hecho, es tan parecido a UNIX que algunas personas lo consideran un clon de UNIX. La idea era lograr que UNIX fuera más eficiente cuando se ejecutara en un microprocesador Intel. Linux ahora está disponible para todas las plataformas y se está volviendo más y más popular entre los programadores y los usuarios comerciales.

7.5 TÉRMINOS CLAVE

administrador de archivos
administrador de dispositivos
administrador de memoria
administrador de procesos
bloque de datos (frame)
cola de espera
diagrama de estado
estado de ejecución
estado de espera
estado de listo
estado de terminación
interfaz de usuario
Linux
memoria virtual
monoprogramación
multiprogramación
página
paginación
paginación bajo demanda
paginación y segmentación bajo demanda
particionamiento
planificación
planificador de procesos
planificador de tareas
privación
proceso
programa
punto puerto
segmentación bajo demanda
sincronización de procesos
sistema distribuido
sistema operativo
sistema operativo monousuario
sistema paralelo
sistema operativo por lotes
software
tarea
tiempo compartido
UNIX

7.6 RESUMEN

- Un sistema operativo facilita la ejecución de otro software, actúa como el administrador general de un sistema de computación y garantiza el uso eficiente de los recursos de hardware y software.
- La evolución de los sistemas operativos ha incluido sistemas operativos por lotes, sistemas de tiempo compartido, sistemas monousuario y sistemas distribuidos.
- El sistema operativo supervisa al administrador de memoria, el administrador de procesos, el administrador de dispositivos, el administrador de archivos y la interfaz de usuario.
- En la monoprogramación, la mayor parte de la capacidad de la memoria está dedicada a un solo programa.
- En la multiprogramación, hay más de un programa en memoria al mismo tiempo.
- En el particionamiento, la memoria se divide en secciones de longitud variable, cada una de las cuales aloja un programa.
- En la paginación, la memoria se divide en secciones de igual tamaño llamadas bloques y el programa se divide en secciones de igual tamaño llamadas páginas. Las páginas de un programa no necesitan ser contiguas, pero todas ellas deben estar presentes en la memoria para la ejecución del programa.
- La paginación bajo demanda es similar a la paginación con la excepción de que no es necesario que todas las páginas estén en la memoria.
- La segmentación bajo demanda es similar a la paginación con la excepción de que en lugar de secciones de igual tamaño, el programa se divide para corresponder con las divisiones del programa.
- La paginación bajo demanda y la segmentación bajo demanda pueden combinarse para mejorar la eficiencia de un sistema de computación.
- La suma de los tamaños de todos los programas en la memoria es la memoria virtual.

- Un programa es una serie de instrucciones no activas escritas por un programador y almacenadas en un disco o cinta.
- Una tarea es un programa seleccionado para su ejecución.
- Un proceso es una tarea que reside en la memoria.
- Un diagrama de estado muestra la relación entre un programa, una tarea y un proceso. Una tarea puede estar en los modos de espera, terminación, listo, ejecución o espera. Un proceso puede estar en uno de los tres últimos estados.
- El planificador de tareas crea un proceso a partir de una tarea y cambia un proceso de regreso a una tarea.
- El administrador de procesos cambia un proceso de un estado a otro.
- Las tareas y los procesos esperan en colas de espera.
- El punto muerto es una situación en la cual un proceso es incapaz de ejecutarse debido al uso irrestricto de recursos por otros procesos.
- La privación es una situación en la cual un proceso es incapaz de ejecutarse debido a que hay demasiadas restricciones en los recursos.
- El administrador de dispositivos controla el acceso a los dispositivos de E/S.
- El administrador de archivos controla el acceso a los archivos.
- La interfaz de usuario es software que acepta solicitudes de los procesos y los interpreta para el resto del sistema operativo.
- Windows 2000, UNIX y Linux son tres sistemas operativos muy comunes.

7.7 PRÁCTICAS

PREGUNTAS DE REPASO

1. ¿Cuál es la diferencia entre un programa de aplicación y un sistema operativo?
2. ¿Cómo ha cambiado la conectividad en red y la interconectividad entre redes a los sistemas operativos?
3. ¿Cuál es la diferencia entre monoprogramación y multiprogramación?
4. ¿Cuáles son los componentes de un sistema operativo?
5. ¿En qué difiere la paginación del particionamiento?
6. ¿Cuál es la diferencia entre una página y un bloque?
7. ¿Por qué la paginación bajo demanda es más eficiente que la paginación normal?
8. ¿Qué es la segmentación bajo demanda?
9. ¿Cómo se relaciona la memoria virtual con la memoria física?
10. ¿Cómo se relaciona un programa con una tarea? ¿Cómo se relaciona una tarea con un proceso? ¿Cómo se relaciona un programa con un proceso?
11. ¿Dónde reside un programa? ¿Dónde reside una tarea? ¿Dónde reside un proceso?
12. ¿Cuál es el propósito de un diagrama de estado?
13. ¿En qué tipos de estado se puede encontrar un proceso?
14. ¿En qué tipos de estado se puede encontrar una tarea?
15. Si un proceso está en el estado de ejecución, ¿a qué estados puede cambiar después?
16. ¿Cuál es la diferencia entre un planificador de tareas y un planificador de procesos?
17. ¿Por qué un sistema operativo necesita colas de espera?
18. ¿Cuál es la diferencia entre un punto muerto y la privación?
19. ¿Cuáles son las funciones de un administrador de dispositivos?
20. ¿Cuáles son las funciones de un administrador de archivos?

PREGUNTAS DE OPCIÓN MÚLTIPLE

21. _____ es un programa que facilita la ejecución de otros programas.
    a. un sistema operativo
    b. el hardware
    c. una cola de espera
    d. un programa de aplicación

22. _____ supervisa la actividad de cada componente en un sistema de cómputo.
    a. un sistema operativo
    b. el hardware
    c. una cola de espera
    d. un programa de aplicación

23. El primer sistema operativo, llamado sistema operativo _____, sólo tenía que garantizar que los recursos se transfirieran de una tarea a la siguiente.
    a. por lotes
    b. de tiempo compartido
    c. personal
    d. paralelo

24. Un sistema operativo _____ se necesita para tareas compartidas entre computadoras conectadas a distancia.
    a. por lotes
    b. de tiempo compartido
    c. paralelo
    d. distribuido

25. La multiprogramación requiere un sistema operativo _____.
    a. por lotes
    b. de tiempo compartido
    c. paralelo
    d. distribuido

26. DOS se considera un sistema operativo _____.
    a. por lotes
    b. de tiempo compartido
    c. paralelo
    d. personal

27. Un sistema con más de un CPU requiere un sistema operativo _____.
    a. por lotes
    b. de tiempo compartido
    c. paralelo
    d. distribuido

28. _____ es multiprogramación con intercambio.
    a. el particionamiento
    b. la paginación
    c. la paginación bajo demanda
    d. la asignación a colas de espera

29. _____ es multiprogramación sin intercambio.
    a. el particionamiento
    b. la memoria virtual
    c. la paginación bajo demanda
    d. la asignación a colas de espera

30. En la (el) _____ sólo un programa puede residir en la memoria para su ejecución.
    a. monoprogramación
    b. multiprogramación
    c. particionamiento
    d. paginación

31. _____ es un método de multiprogramación en el cual varios programas están completamente en la memoria, cada uno de los cuales ocupa un espacio contiguo.
    a. el particionamiento
    b. la paginación
    c. la paginación bajo demanda
    d. la segmentación bajo demanda

32. En la paginación un programa se divide en secciones de igual tamaño llamadas _____.
    a. páginas
    b. bloques
    c. segmentos
    d. particiones

33. En el (la)_____ el programa puede dividirse en secciones de diferente tamaño.
    a. particionamiento
    b. paginación
    c. paginación bajo demanda
    d. segmentación bajo demanda

34. En el (la)_____ el programa puede dividirse en secciones de igual tamaño llamadas páginas, pero no es necesario que las páginas estén en la memoria al mismo tiempo para la ejecución del programa.
    a. particionamiento
    b. paginación
    c. paginación bajo demanda
    d. segmentación bajo demanda

35. Un proceso en el estado de _____ puede entrar ya sea en el estado de listo, terminación o espera.
    a. espera
    b. virtual
    c. ejecución
    d. incisos a y c

36. Un proceso en el estado de listo entra en el estado de ejecución cuando _____.
    a. entra a la memoria
    b. solicita E/S
    c. obtiene acceso al CPU
    d. termina su ejecución

37. Un programa se convierte en un(a) _____ cuando el sistema operativo lo selecciona y lo lleva al estado de espera.
    a. tarea
    b. proceso
    c. punto muerto
    d. partición

38. Todos los procesos son _____.
    a. tareas
    b. programas
    c. particiones
    d. incisos a y b

39. El planificador _____ crea un proceso a partir de una tarea y cambia un proceso de regreso a tarea.
    a. de tareas
    b. de procesos
    c. virtual
    d. de colas de espera

40. El planificador _____ cambia un proceso de un estado a otro.
    a. de tareas
    b. de procesos
    c. virtual
    d. de colas de espera

41. Para evitar la (un) _____ un sistema operativo puede poner restricciones a los recursos en los procesos.
- **a.** privación
- **b.** sincronización
- **c.** paginación
- **d.** punto muerto

42. _____ puede ocurrir si un proceso tiene demasiadas restricciones en los recursos.
- **a.** la privación
- **b.** la sincronización
- **c.** la paginación
- **d.** un punto muerto

43. El administrador de _____ es responsable del archivo y el respaldo.
- **a.** memoria
- **b.** procesos
- **c.** dispositivos
- **d.** archivos

44. El administrador de _____ es responsable del acceso a los dispositivos de E/S.
- **a.** memoria
- **b.** procesos
- **c.** dispositivos
- **d.** archivos

45. El planificador de tareas y el planificador de procesos están bajo el control del administrador de _____.
- **a.** memoria
- **b.** procesos
- **c.** dispositivos
- **d.** archivos

EJERCICIOS

46. Una computadora tiene un sistema operativo de monoprogramación. Si el tamaño de la memoria es 64 MB y el sistema operativo residente necesita 4 MB, ¿cuál es el tamaño máximo de un programa que puede ejecutar esta computadora?

47. Repita el ejercicio 46 si el sistema operativo asigna automáticamente 10 MB de memoria a los datos.

48. Un sistema operativo de monoprogramación ejecuta programas que en promedio necesitan 10 microsegundos para tener acceso al CPU y 70 microsegundos para acceder a los dispositivos de E/S. ¿Qué porcentaje de tiempo está inactivo el CPU?

49. Un sistema operativo multiprogramación utiliza un esquema de repartición y divide los 60 MB de memoria disponibles en cuatro particiones de 10 MB, 12 MB, 18 MB y 20 MB. El primer programa a ser ejecutado necesita 17 MB y ocupa la tercera partición. El segundo programa necesita 8 MB y ocupa la primera partición. El tercero necesita 10.5 MB y ocupa la segunda partición. Finalmente, el cuarto programa necesita 20 MB y ocupa la cuarta partición. ¿Cuál es el total de memoria utilizada? ¿Cuál es el total de memoria desperdiciada? ¿Qué porcentaje de memoria se desperdicia?

50. Repita el ejercicio 49 si todos los programas necesitan 10 MB de memoria.

51. Un sistema operativo multiprogramación utiliza la paginación. La memoria disponible es 60 MB divididos en 15 páginas, cada una de 4 MB. El primer programa necesita 13 MB. El segundo programa necesita 12 MB. El tercer programa necesita 27 MB.
- **a.** ¿Cuántas páginas utiliza el primer programa?
- **b.** ¿Cuántas páginas utiliza el segundo programa?
- **c.** ¿Cuántas páginas utiliza el tercer programa?
- **d.** ¿Cuántas páginas quedan sin utilizarse?
- **e.** ¿Qué cantidad total de memoria se desperdicia?
- **f.** ¿Qué porcentaje de memoria se desperdicia?

52. Un sistema operativo utiliza 0 memoria virtual pero requiere que todo el programa esté en la memoria física durante la ejecución (sin paginación o segmentación). El tamaño de la memoria física es 100 MB. El tamaño de la memoria virtual es 1 GB. ¿Cuántos programas de 10 MB puede ejecutar simultáneamente este sistema operativo? ¿Cuántos de ellos pueden estar en memoria en cualquier momento? ¿Cuántos de ellos deben estar en disco?

53. ¿Cuál es el estado de un proceso en cada una de las siguientes situaciones (según la figura 7.9)?
- **a.** El proceso está utilizando al CPU.
- **b.** El proceso ha terminado de imprimir y necesita la atención del CPU de nuevo.
- **c.** El proceso se ha detenido porque su espacio de tiempo ha terminado.
- **d.** El proceso está leyendo los datos desde el teclado.
- **e.** El proceso está imprimiendo los datos.

54. Tres procesos (A, B y C) se están ejecutando simultáneamente. El proceso A tiene al Archivo 1, pero necesita el Archivo 2. El proceso B tiene al Archivo 3, pero necesita el Archivo 1. El proceso C tiene al Archivo 2, pero necesita el Archivo 3. Dibuje un diagrama similar al de la figura 7.13 para estos procesos. ¿Es una situación de punto muerto?

55. Tres procesos (A, B y C) se están ejecutando de manera simultánea. El proceso A tiene al Archivo 1. El proceso B tiene al Archivo 2, pero necesita el Archivo 1. El proceso C tiene al Archivo 3, pero necesita el Archivo 2. Dibuje un diagrama similar al de la figura 7.13 para estos procesos. ¿Se trata de una situación de punto muerto? Si su respuesta es "No", muestre cómo los procesos terminarían sus tareas finalmente.

Algoritmos

En este capítulo presentamos el concepto de algoritmos, los cuales son procedimientos paso a paso para resolver un problema. Después analizamos las herramientas utilizadas para desarrollar algoritmos. Finalmente, damos algunos ejemplos comunes de algoritmos iterativos y recursivos.

8.1 CONCEPTO

En esta sección definimos de manera informal lo que es un **algoritmo** y elaboramos el concepto utilizando un ejemplo.

DEFINICIÓN INFORMAL

Una definición informal de un algoritmo es:

Algoritmo: un método paso a paso para resolver un problema o realizar una tarea.

En esta definición un algoritmo es independiente del sistema de cómputo. Más específicamente, también debemos notar que el algoritmo acepta una lista de **datos de entrada** y crea una lista de **datos de salida** (figura 8.1).

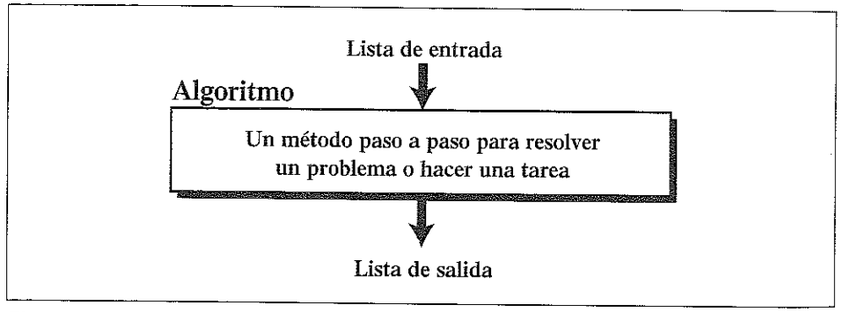


Figura 8.1 Definición informal de un algoritmo utilizado en una computadora

EJEMPLO

Elaboremos esta simple definición con un ejemplo. Queremos desarrollar un algoritmo para encontrar el entero más grande entre una lista de enteros positivos. El algoritmo debe encontrar el entero mayor entre una lista de enteros de cualquier valor (5, 1000, 10 000, 1 000 000, etc.). El algoritmo debe ser general y no depender del número de enteros.

Es obvio que encontrar el entero más grande entre muchos enteros (por ejemplo, 1 millón) es una tarea que no puede realizarse en un solo paso (por una persona o por una computadora). El algoritmo necesita probar cada entero uno por uno.

Para resolver este problema usted necesita un método intuitivo. Primero utilice un número pequeño de enteros (por ejemplo, cinco) y luego extienda la solución a cualquier número de enteros. Su solución de cinco enteros sigue los mismos principios y restricciones para 1000 o 1 000 000 de enteros. Suponga, incluso para el caso de cinco enteros, que el algoritmo maneja los enteros uno a uno. Observa el primer entero sin conocer los valores de los enteros restantes. Después de observar el primero, calcula el segundo entero y así sucesivamente. La figura 8.2 muestra una manera de resolver este problema.

Llamamos al algoritmo **EncontrarMayor**. Cada algoritmo tiene un nombre que lo distingue de otros algoritmos. El algoritmo recibe una lista de cinco enteros (como entrada) y proporciona el mayor como salida.

Entrada

El algoritmo acepta la lista de cinco enteros como entrada.

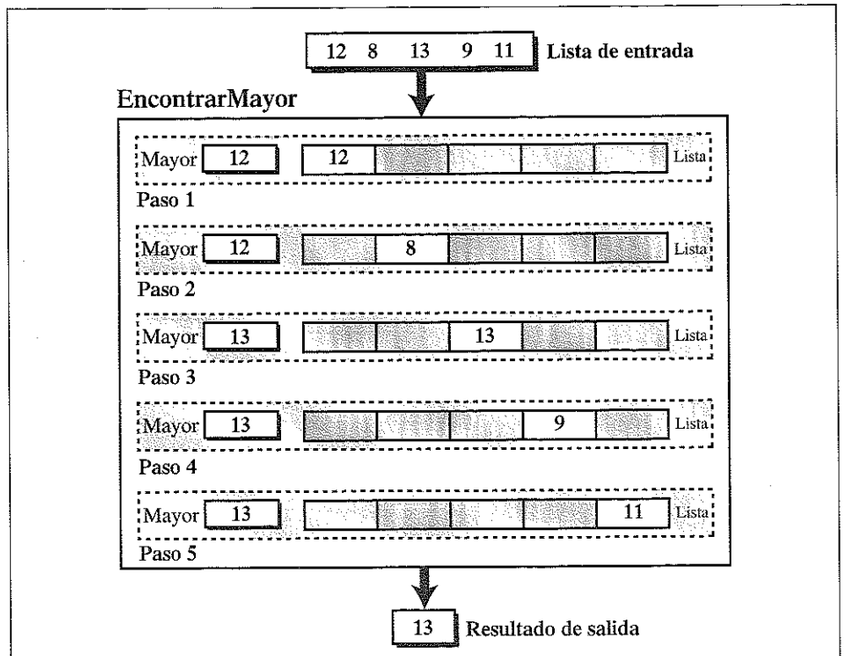


Figura 8.2 Encontrar el entero mayor entre los cinco enteros

Procesamiento

El algoritmo utiliza los siguientes cinco pasos para encontrar el entero mayor.

PASO 1 En este paso, el algoritmo examina el primer entero (12). Puesto que no ha manejado al resto de los enteros (observe que no revelamos los enteros restantes debido a que el algoritmo no los examina en este momento), decide que el entero mayor (hasta ahora), es el primer entero. El algoritmo define un elemento de datos, llamado Mayor y establece el primer entero (12) como su valor.

PASO 2 El entero mayor hasta ahora es 12, pero el nuevo número puede cambiar la situación. El algoritmo hace una comparación entre el valor del Mayor (12) y el valor del segundo entero (8). Encuentra que el Mayor es más grande que el segundo entero, lo cual significa que el Mayor aún está manteniendo al entero más grande. No hay necesidad de cambiar el valor del Mayor.

PASO 3 El entero mayor hasta ahora es 12, pero el nuevo número (13) es más grande que el Mayor. Esto significa que el valor del Mayor ya no es válido. El valor del Mayor debe remplazarse con el tercer número (13). El algoritmo cambia el valor del mayor a 13 y se mueve al paso siguiente.

PASO 4 Nada se cambia en este paso debido a que el Mayor es más grande que el cuarto entero (9).

PASO 5 De nuevo nada se cambia debido a que el Mayor es más grande que el quinto entero (11).

Salida

Como ya no hay más enteros a procesar, el algoritmo da el resultado del valor del Mayor, el cual es 13.

DEFINICIÓN DE ACCIONES

La figura 8.2 no muestra qué debe hacerse en cada paso. Usted puede modificar la figura para que muestre más detalles. En el paso 1, establezca el valor del primer entero como el Mayor. No obstante, en los pasos 2 a 5 se requieren acciones adicionales para comparar el valor del Mayor con el entero que se esté procesando en ese momento. Si el entero actual es más grande que el Mayor, se establece el entero actual como el valor del Mayor (figura 8.3).

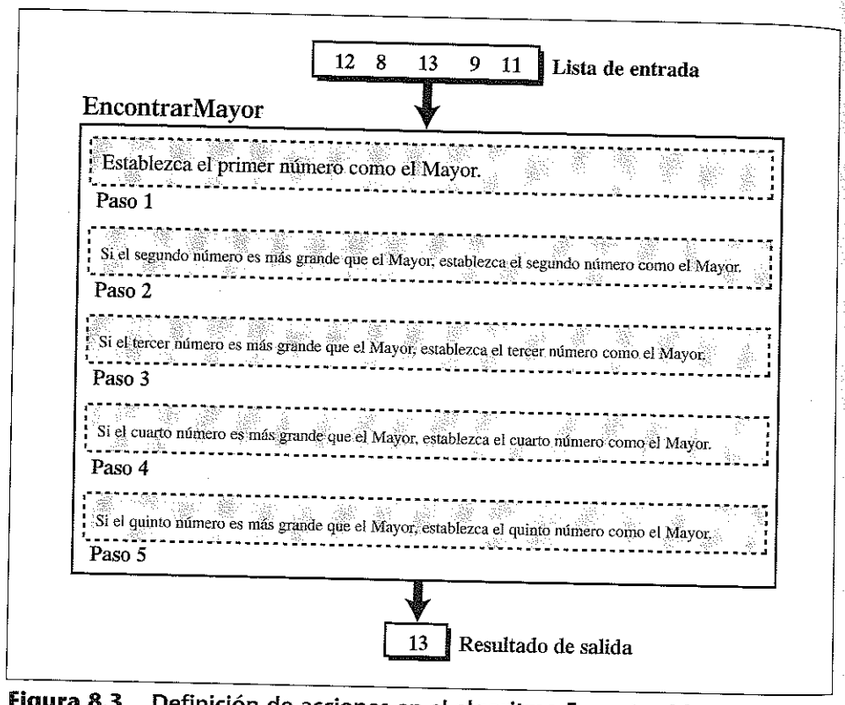


Figura 8.3 Definición de acciones en el algoritmo EncontrarMayor

REFINAMIENTO

Este algoritmo necesita refinarse para que sea aceptable para la comunidad de programación. Existen dos problemas. Primero, la acción para el primer paso es diferente de aquellas para los otros pasos. Segundo, la formulación no es la misma en los pasos 2 a 5. Usted puede redefinir fácilmente el algoritmo para eliminar estos dos inconvenientes. Cambie la formulación en los pasos 2 a 5 a "Si el número actual es más grande que el Mayor, establezca el número actual como el Mayor". La razón de que el primer paso sea diferente de los otros pasos se debe a que el Mayor no se ha inicializado. Si usted inicializa el Mayor en 0 (ningún entero positivo puede ser menor que 0), entonces el primer paso puede ser igual que los otros pasos. Añada un nuevo paso (llámelo paso 0 para mostrar que debe realizarse antes que el procesamiento de cualquier número). La figura 8.4 muestra el resultado de este refinamiento. Observe que no tiene que mostrar todos los pasos debido a que ahora son iguales.

GENERALIZACIÓN

¿Es posible generalizar el algoritmo? Usted quiere encontrar el número mayor de N enteros positivos, donde N puede ser 1 000, 1 000 000 o incluso más. Desde luego puede seguir los pasos de la figura 8.4 y repetir cada paso. Pero si usted cambia el algoritmo a un programa, entonces ¡aunque parezca mentira necesita teclear las acciones para N pasos! Hay una mejor forma de hacer esto. Usted puede indicar a la computadora que repita los pasos N veces. Ahora incluimos esta característica en nuestro algoritmo pictórico (figura 8.5).

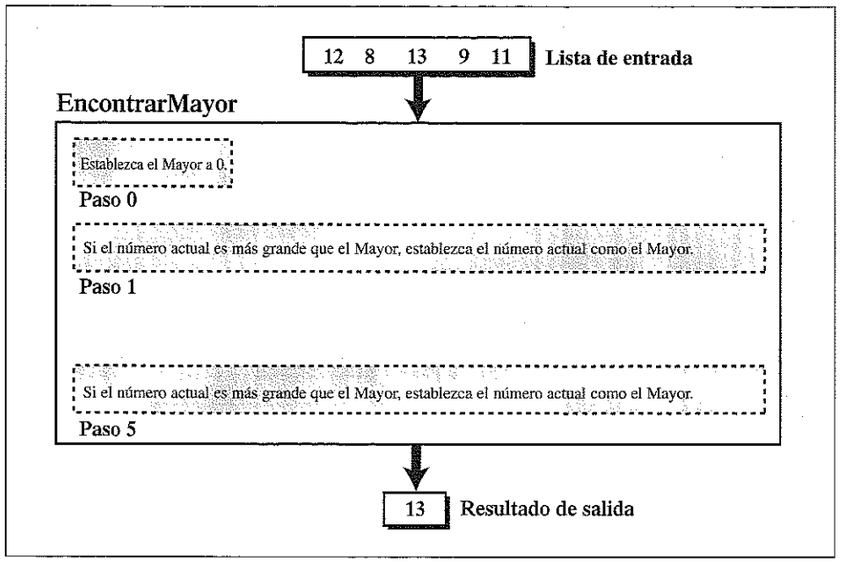


Figura 8.4 EncontrarMayor refinado

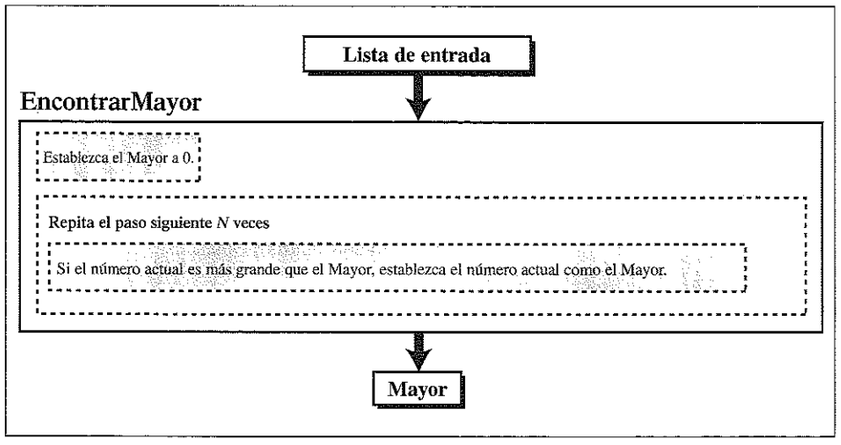


Figura 8.5 Generalización de EncontrarMayor

8.2 TRES ESTRUCTURAS DE CONTROL

Los científicos de la computación han definido tres estructuras de control para un programa o algoritmo estructurado. La idea es que un programa debe conformarse de una combinación de sólo estas tres estructuras: secuencia, decisión (selección) y repetición (figura 8.6). Se ha probado que no hay necesidad de ninguna otra estructura. El usar sólo estas estructuras hace que un programa o algoritmo sea fácil de entender, depurar o cambiar.

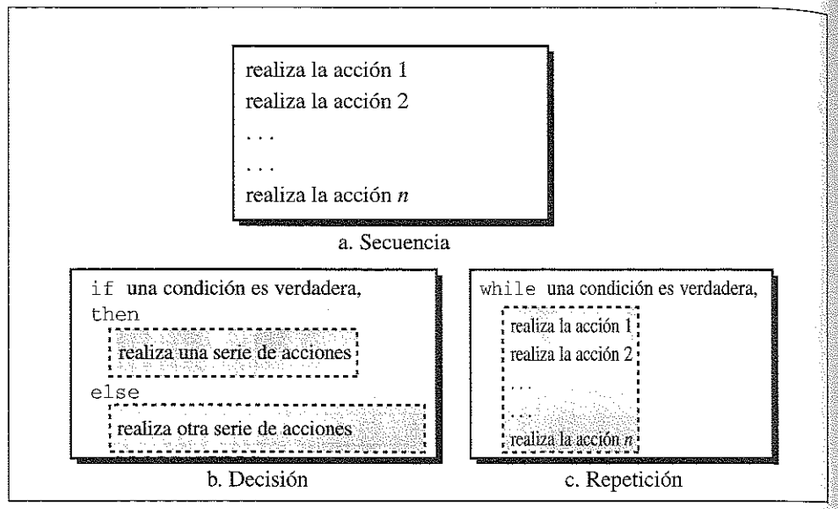


Figura 8.6 Tres estructuras de control

SECUENCIA

La primera estructura se llama secuencia. Un algoritmo y finalmente un programa es una secuencia de instrucciones, la cual puede ser ya sea una instrucción simple o cualquiera de las otras dos estructuras.

DECISIÓN

Algunos problemas no pueden resolverse sólo con una secuencia de instrucciones simples. En ocasiones usted necesita probar una condición. Si el resultado de la prueba es verdadero, usted sigue una secuencia de instrucciones; si es falso, usted sigue una secuencia de instrucciones diferente. A esto se le llama estructura de decisión (selección).

REPETICIÓN

En algunos problemas, debe repetirse la misma secuencia de instrucciones. Esto se maneja con la estructura de repetición. Encontrar el número mayor entre una serie de números (que se vio al principio del capítulo) es una estructura de este tipo.

8.3 REPRESENTACIÓN DE ALGORITMOS

Hasta ahora, hemos utilizado figuras para expresar el concepto de un algoritmo. Durante las últimas décadas, se han diseñado herramientas para este propósito. Dos de estas herramientas, el diagrama de flujo y el pseudocódigo, se presentan aquí.

DIAGRAMA DE FLUJO

Un **diagrama de flujo** es una representación pictórica de un algoritmo. Oculta todos los detalles de un algoritmo en un intento por dar la idea global; muestra cómo el algoritmo fluye de principio a fin. Los diagramas de flujo se cubren con detalle en el Apéndice C. Aquí sólo mostramos cómo se representan las tres estructuras en diagramas de flujo (figura 8.7).

PSEUDOCÓDIGO

El **pseudocódigo** es una representación de un algoritmo parecida al inglés. No hay un estándar para pseudocódigo; algunas personas utilizan muchos detalles y otras usan menos. Algunos utilizan un tipo de código que se parece al inglés y otros utilizan una sintaxis como el lenguaje Pascal. El pseudocódigo se cubre con detalle en el Apéndice D. Aquí sólo mostramos cómo pueden representarse las tres sentencias mediante pseudocódigo (figura 8.8).

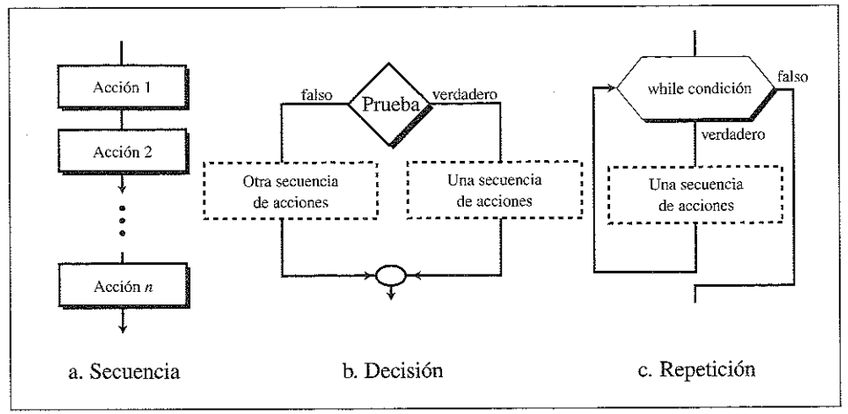


Figura 8.7 Diagrama de flujo para las tres estructuras de control

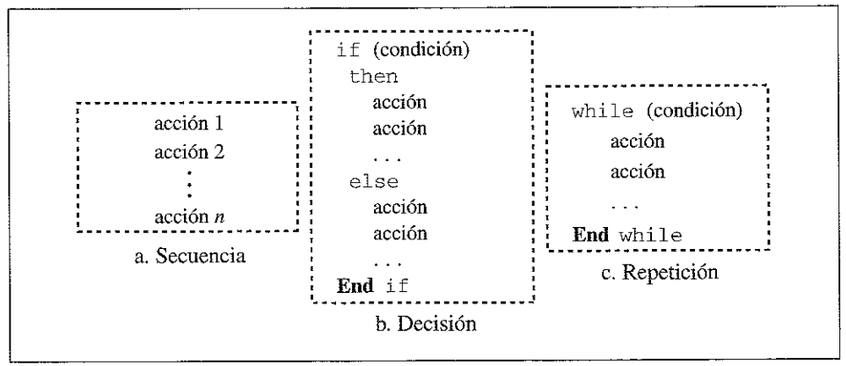


Figura 8.8 Pseudocódigo para las tres estructuras de control

EJEMPLO 1

Escriba un algoritmo en pseudocódigo que encuentre la estructura de dos números.

SOLUCIÓN

Éste es un problema simple que puede resolverse usando sólo la estructura de secuencia. Observe que las instrucciones se numeraron para una referencia fácil (algoritmo 8.1). Observe también que se nombró el algoritmo, se definió la entrada del algoritmo y al final se mostró el resultado usando una instrucción de retorno.

Algoritmo 8.1 Promedio de dos

PromedioDeDos
Entrada: Dos números

1. Suma los dos números
2. Divide el resultado entre 2
3. Devuelve el resultado del paso 2

End

■

EJEMPLO 2

Escriba un algoritmo para traducir una calificación numérica a una calificación aprobado/reprobado.

SOLUCIÓN

Este problema no puede resolverse sólo con la estructura de secuencia. También se necesita la estructura de decisión. Se da un número a la computadora entre 0 y 100. Devuelve "aprobado" si el número es mayor o igual que 70; devuelve "reprobado" si el número es menor que 70. El algoritmo 8.2 muestra el pseudocódigo para este algoritmo. Observe la numeración. La instrucción de decisión tiene el número 1 y su subinstrucción tiene el número 1.1 (la primera subinstrucción dentro de la primera instrucción).

Algoritmo 8.2 Traducción a aprobado/reprobado

```
   Aprobado/reprobado
   Entrada: Un número
1. if (el número es mayor o igual que 70)
      then
         1.1  Establece la calificación a "aprobado"
   else
         1.2  Establece la calificación a "reprobado"
   End if
2. Devuelve la calificación
   End
```

■

EJEMPLO 3

Escriba un algoritmo para cambiar una calificación numérica a una calificación de letra.

SOLUCIÓN

Este problema necesita más de una decisión. El pseudocódigo en el algoritmo 8.3 muestra una forma (no la mejor, sino una fácil de comprender) para resolver el problema. De nuevo, se da un número entre 0 y 100 y usted quiere cambiarlo a una calificación de letra (A, B, C, D o F).

Algoritmo 8.3 Calificación con letra

```
   CalificaciónLetra
   Entrada: Un número
1. if (el número está entre 90 y 100, inclusive)
      then
         1.1  Establece la calificación a "A"
   End if
2. if (el número está entre 80 y 89, inclusive)
      then
         2.1  Establece la calificación a "B"
   End if
3. if (el número está entre 70 y 79, inclusive)
      then
         3.1  Establece la calificación a "C"
   End if
```



```
4. if (el número está entre 60 y 69, inclusive)
      then
         4.1  Establece la calificación a "D".
   End if
5. if (el número es menor que 60, inclusive)
      then
         5.1  Establece la calificación a "F"
   End if
6. Devuelve la calificación
   End
```

■

Observe que las instrucciones `if` no necesitan `else` porque no ocurre nada si la condición es falsa.

EJEMPLO 4

Escriba un algoritmo para encontrar el número mayor de una serie de enteros. Usted no conoce el número de enteros.

SOLUCIÓN

Utilice el concepto de la figura 8.5 para escribir un algoritmo para este problema (algoritmo 8.4).

Algoritmo 8.4 Encontrar el mayor

```
   EncontrarMayor
   Entrada: Una lista de enteros positivos
1. Establece el Mayor en 0
2. while (más enteros)
      2.1 if (el entero es más grande que el Mayor)
            then
               2.1.1 Establece el valor del entero como el Mayor
         End if
   End while
3. Devuelve el Mayor
   End
```

■

EJEMPLO 5

Escriba un algoritmo para encontrar el número mayor entre 1000 números.

SOLUCIÓN

En este caso se necesita un contador para contar el número de enteros. Inicialice el contador en 0 y auméntelo en cada repetición. Cuando el contador sea 1000, salga del ciclo (*loop*) (algoritmo 8.5).

Algoritmo 8.5 Encontrar el número mayor entre 1000 números

```
   EncontrarMayor
   Entrada: 1000 enteros positivos
1. Establece el Mayor en 0
2. Establece el Contador en 0
```

```
3. while (el Contador es menor que 1000)
        3.1 if (el entero es más grande que el Mayor)
              then
                  3.1.1 Establece el valor del entero como el Mayor
            End if
        3.2 Incrementa el contador
   End while
4. Devuelve el Mayor
   End
```

■

8.4 DEFINICIÓN MÁS FORMAL

Ahora que hemos estudiado el concepto de un algoritmo y hemos mostrado su representación, se presenta una definición más formal:

> **Algoritmo:** una serie ordenada de pasos precisos que produce un resultado y termina en un tiempo finito.

Expliquemos con mayor detalle esta definición.

SERIE ORDENADA

Un algoritmo debe ser una serie de instrucciones ordenadas y bien definidas.

PASOS PRECISOS

Cada paso en un algoritmo debe definirse de una manera clara e inequívoca. Si un paso es para *sumar dos números* se deben definir ambos números, así como la operación de suma. No puede utilizarse el mismo símbolo para referenciar suma en una parte y multiplicación en algún otro lado.

PRODUCE UN RESULTADO

Un algoritmo debe producir un resultado; de lo contrario es inútil. El resultado pueden ser datos que se devuelven al algoritmo que hace la llamada o algún otro algoritmo (por ejemplo, imprimir).

TERMINAR EN UN TIEMPO FINITO

Un algoritmo debe terminar (detenerse). Si no lo hace (por ejemplo que tenga un ciclo infinito), entonces no se ha creado un algoritmo. En el capítulo 17, estudiaremos problemas solubles e insolubles, y usted verá que un problema soluble tiene una solución en la forma de un algoritmo que termina.

8.5 SUBALGORITMOS

Con las tres estructuras que se han escrito se puede crear un algoritmo para cualquier problema soluble. No obstante, los principios de la programación estructurada requieren que un algoritmo se divida en pequeñas unidades llamadas **subalgoritmos** (los términos **subprogramas, subrutinas, procedimientos, funciones, métodos** y **módulos** también se utilizan). Cada subalgoritmo a su vez se divide en subalgoritmos más pequeños. El proceso continúa hasta que los subalgoritmos se vuelven intrínsecos (comprensibles de inmediato).

Usted puede dividir el algoritmo 8.4 en un algoritmo principal y en un subalgoritmo. Este algoritmo es un buen candidato para este propósito debido a que repite una tarea muchas veces. En cada **iteración**, el algoritmo encuentra el número mayor entre dos enteros. Usted puede separar esta parte de la tarea y crear una pequeña subtarea fuera de ella (figura 8.9).

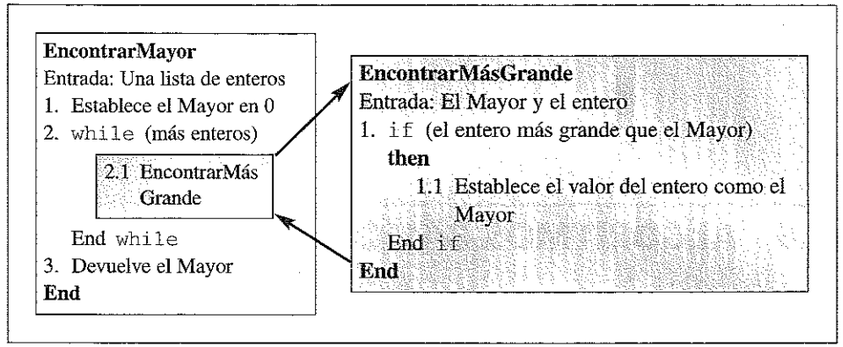


Figura 8.9 Concepto de subalgoritmo

El algoritmo EncontrarMayor ejecuta cada instrucción hasta que encuentra el nombre de otro algoritmo, EncontrarMásGrande, en la línea 2.1. En esta línea, EncontrarMayor se suspende y EncontrarMásGrande comienza. EncontrarMásGrande encuentra el número más grande entre el valor del Mayor y el valor del entero actual. Esto se conoce como una llamada de función. Observe que en cada iteración EncontrarMásGrande se llama una vez. Usted puede pensar que este diseño es más grande que el diseño original. En este caso es verdad. Pero el uso de subalgoritmos tiene ventajas que hacen algo más que compensar el esfuerzo adicional de escribir unas cuantas líneas más de código:

- Es más comprensible. Al estudiar el algoritmo EncontrarMayor, usted puede ver de inmediato que una tarea (encontrar el más grande entre dos números) se repite.
- Un subalgoritmo puede llamarse muchas veces en diferentes partes del algoritmo principal sin que se vuelva a escribir.

El algoritmo 8.6 muestra en pseudocódigo cómo utilizar un subalgoritmo dentro de un algoritmo.

Algoritmo 8.6 Encontrar el número mayor

```
EncontrarMayor
Entrada: Una lista de enteros positivos
1. Establece el Mayor en 0
2. while (más enteros)
      2.1 EncontrarMayor
   End while
3. Devuelve el Mayor
   End
```

```
EncontrarMásGrande
Entrada: El número mayor y el entero actual
1. if (el entero es más grande que el Mayor)
      then
          1.1 Establece el valor del entero como el Mayor
      End if
   End
```

■

CARTA ESTRUCTURADA

Otra herramienta que los programadores utilizan es la **carta estructurada,** la cual es una herramienta de alto nivel que muestra la relación entre diferentes módulos en un algoritmo. Se utiliza principalmente en el nivel de diseño y no en el nivel de programación. En el apéndice E se estudia la carta estructurada.

8.6 ALGORITMOS BÁSICOS

Varios algoritmos se utilizan en la ciencia de la computación con tanta frecuencia que se les considera básicos. En esta sección estudiamos los más comunes. El análisis es muy general, ya que su implementación depende del lenguaje.

SUMATORIA

Uno de los algoritmos más comunes en las ciencias de la computación es la **sumatoria**. Usted puede sumar dos o tres números muy fácilmente, pero ¿cómo puede sumar muchos números o una serie variable de números? La solución es simple: Utilice el operador de suma en un ciclo (figura 8.10). Un algoritmo que suma tiene tres partes lógicas:

1. Inicialización de la suma al principio.
2. El ciclo, el cual añade en cada iteración un nuevo número a la suma.
3. Devolver el resultado después de salir del ciclo.

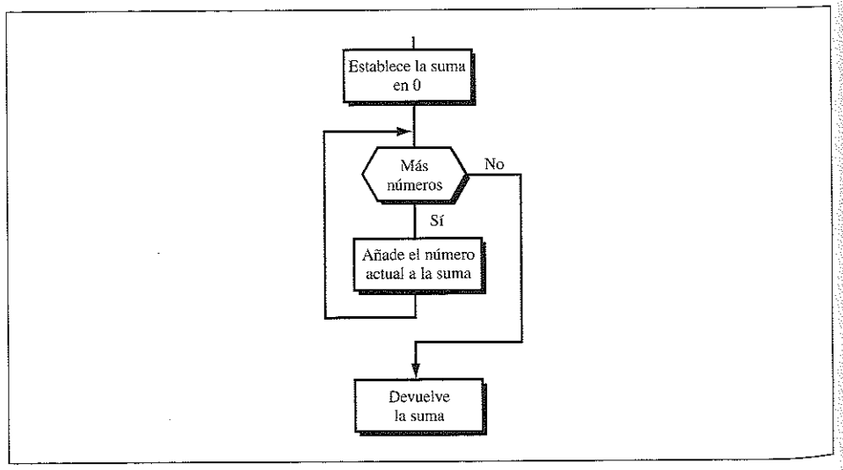


Figura 8.10 Sumatoria

MULTIPLICATORIA

Otro algoritmo común es hallar el producto de una lista de números. La solución es simple: Use el operador de multiplicación en un ciclo (figura 8.11). Un algoritmo de producto tiene tres partes lógicas:

1. Inicialización del producto al principio.
2. El ciclo, el cual multiplica en cada iteración un nuevo número por el producto.
3. Devuelve el resultado después de salir del ciclo.

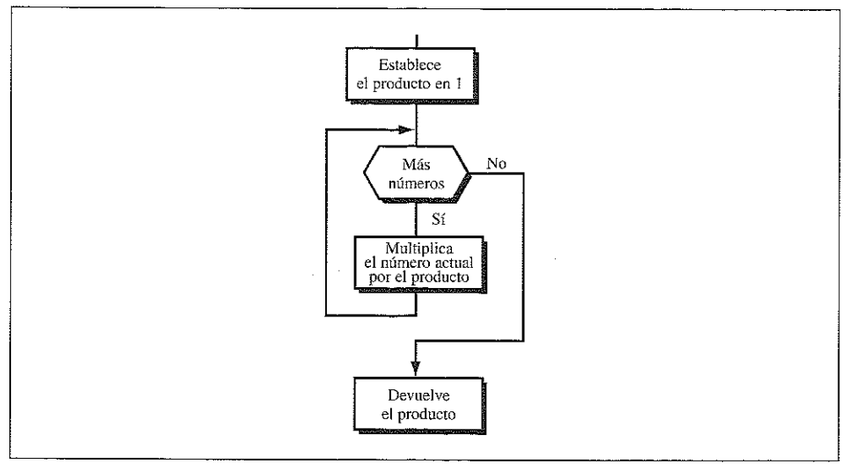


Figura 8.11 Multiplicatoria

Por ejemplo, el algoritmo anterior puede utilizarse para calcular x^n usando una modificación menor (se deja como ejercicio). Como otro ejemplo, el mismo algoritmo puede usarse para calcular el factorial de un número (se deja como ejercicio).

MENOR Y MAYOR

Estudiamos el algoritmo para encontrar el número mayor entre una lista de números al principio del capítulo. La idea era escribir una estructura de decisión para encontrar el más grande de dos números. Si usted pone esta estructura en un ciclo, puede encontrar el número mayor de una lista de números.

Determinar el número menor entre una lista de números es similar con dos diferencias menores. Primero, se utiliza una estructura de decisión para encontrar el menor de dos números. Segundo, se inicializa con un número muy grande en lugar de uno muy pequeño.

ORDENACIÓN

Una de las aplicaciones más comunes en las ciencias de la computación es la **ordenación**, la cual es el proceso mediante el cual los datos se acomodan de acuerdo con sus valores. La gente está rodeada de datos. Si los datos no estuvieran ordenados, tomaría horas y horas encontrar una sola pieza de información. Imagine la dificultad de encontrar el número telefónico de alguien en un directorio telefónico que no estuviera ordenado.

En esta sección presentamos tres algoritmos de ordenación: ordenación de selección, ordenación por burbuja y ordenación por inserción. Estos tres algoritmos de ordenación son la base de los ordenamientos más rápidos utilizados actualmente en las ciencias de la computación.

Ordenación por selección

En la **ordenación por selección**, la lista se separa en dos sublistas –ordenada y sin ordenar– las cuales se dividen por una pared imaginaria. Usted encuentra el elemento más pequeño de la sublista sin ordenar y lo intercambia por el elemento al principio de los datos sin ordenar. Después de cada selección e intercambio, la pared imaginaria entre las dos sublistas mueve un elemento hacia delante, aumentando el número de elementos ordenados y disminuyendo el número de aquellos sin ordenar. Cada vez que usted mueve un elemento de la sublista sin ordenar a la lista ordenada, usted ha completado un **paso de ordenación**. Una lista de n elementos requiere $n - 1$ pasos para reacomodar los datos completamente. El orden de selección se presenta gráficamente en la figura 8.12.

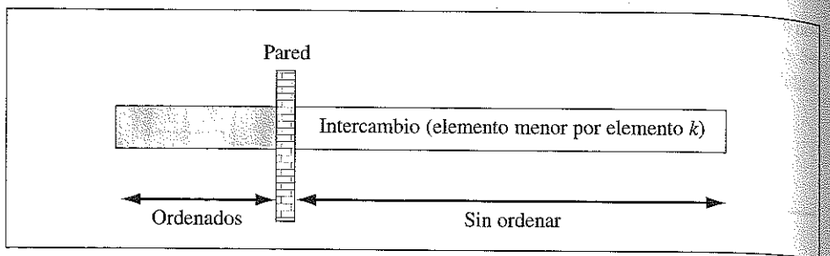


Figura 8.12 Ordenación por selección

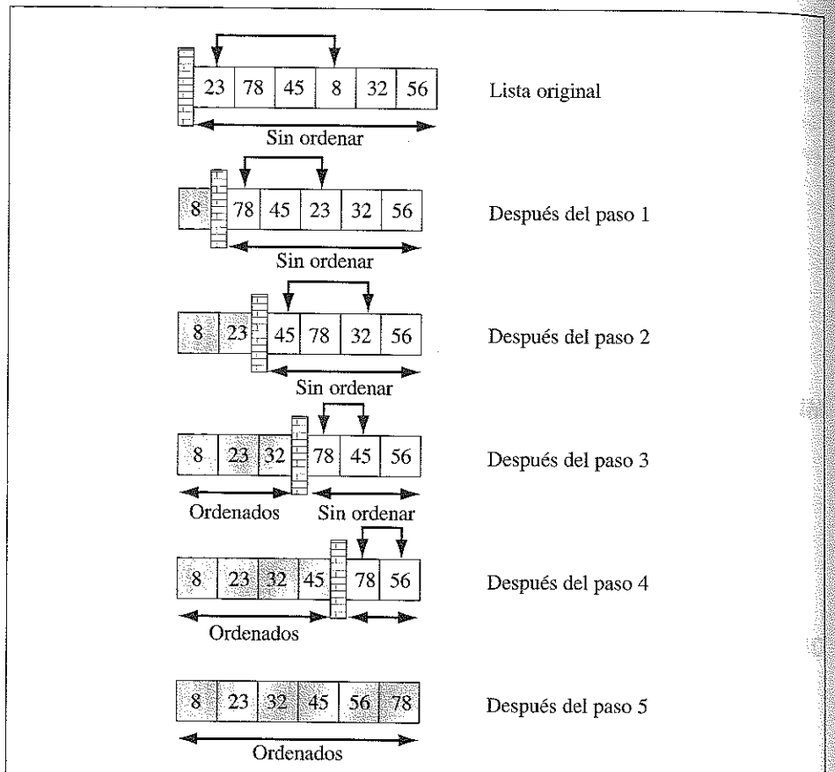


Figura 8.13 Ejemplo de ordenación por selección

La figura 8.13 ubica un conjunto de seis enteros y como son ordenados. Muestra cómo la pared entre las sublistas ordenada y sin ordenar se mueve en cada paso. A medida que estudie la figura se dará cuenta de que la lista está ordenada después de cinco pasos, lo cual es uno menos que el número de elementos en la lista. De esta manera, si usted utiliza un ciclo para controlar el ordenamiento, el ciclo tendrá una iteración menos que el número de elementos en la lista.

Algoritmo de la ordenación por selección

El algoritmo utiliza dos ciclos, uno dentro del otro. El ciclo exterior se itera para cada paso; el ciclo interior encuentra el elemento más pequeño en la lista sin ordenar. La figura 8.14 muestra el diagrama de flujo para el algoritmo de ordenación por selección. Dejamos el pseudocódigo como un ejercicio.

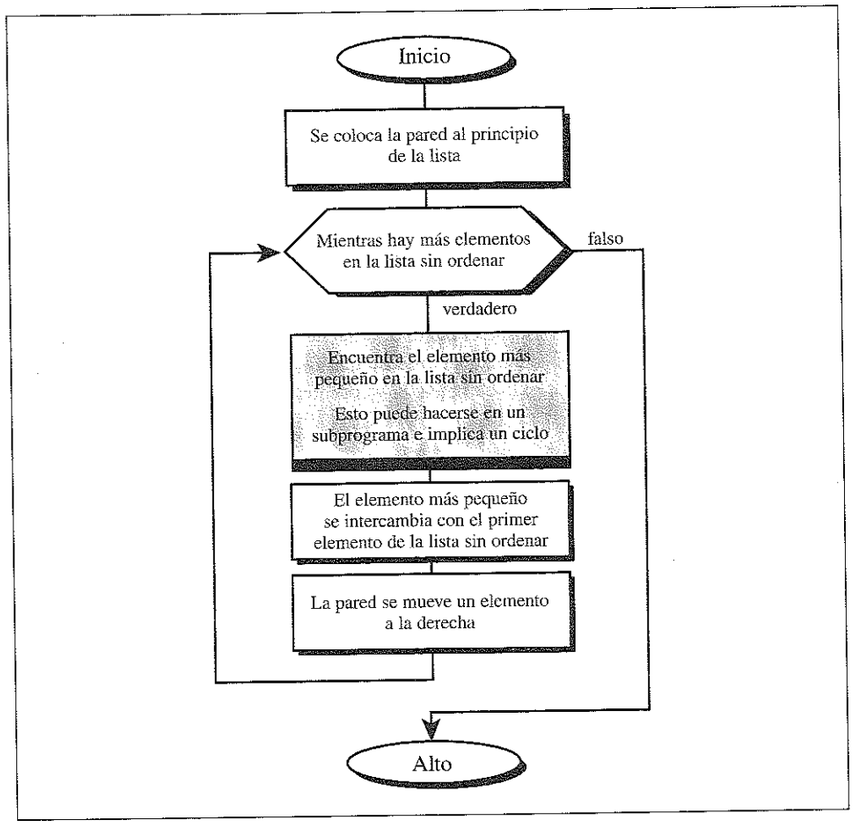


Figura 8.14 Algoritmo de ordenación por selección

Ordenación por burbuja

En el método de **ordenación por burbuja**, la lista se divide en dos sublistas: ordenada y sin ordenar. El elemento más pequeño es una burbuja de la sublista sin ordenar y se mueve a la lista ordenada. Después de que el elemento menor se ha movido a la lista ordenada, la pared se mueve un elemento hacia delante, aumentando el número de elementos ordenados y disminuyendo el número de aquellos sin ordenar. Cada vez que un elemento se mueve de la sublista sin ordenar a la sublista ordenada, un paso de ordenación se completa (figura 8.15). Dada una lista de *n elementos,* la ordenación por burbuja requiere hasta $n - 1$ pasos para ordenar los datos.

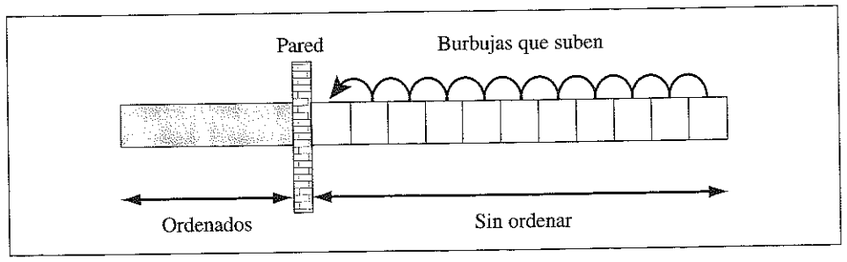


Figura 8.15 Ordenación por burbuja

La figura 8.16 muestra cómo la pared se mueve un elemento en cada paso. Al observar el primer paso, usted comienza con 56 y lo compara con 32. Puesto que 56 no es menor que 32, éste no se mueve y usted se mueve un elemento hacia abajo. No ocurre ningún intercambio hasta que se compara 45 con 8. Puesto que 8 es menor que 45, se intercambian los dos elementos y usted se mueve hacia abajo un elemento. Debido a que 8 se movió hacia abajo, ahora se compara con 78 y estos dos elementos se intercambian. Finalmente, el 8 se compara con el 23 y se intercambian. Esta serie de intercambios coloca al 8 en la primera ubicación y la pared se mueve una posición hacia arriba.

El orden de burbuja se escribió originalmente para "burbujear hacia abajo" el elemento en la posición superior de la lista. Desde el punto de vista de la eficiencia, no hay diferencia entre que sea el elemento en la posición superior el que burbujea hacia abajo o sea el elemento inferior. Desde el punto de vista de la consistencia, sin embargo, hace que las comparaciones entre los ordenamientos sean más fáciles si los tres trabajan del mismo modo. Por esa razón, hemos elegido burbujear el valor inferior en cada paso.

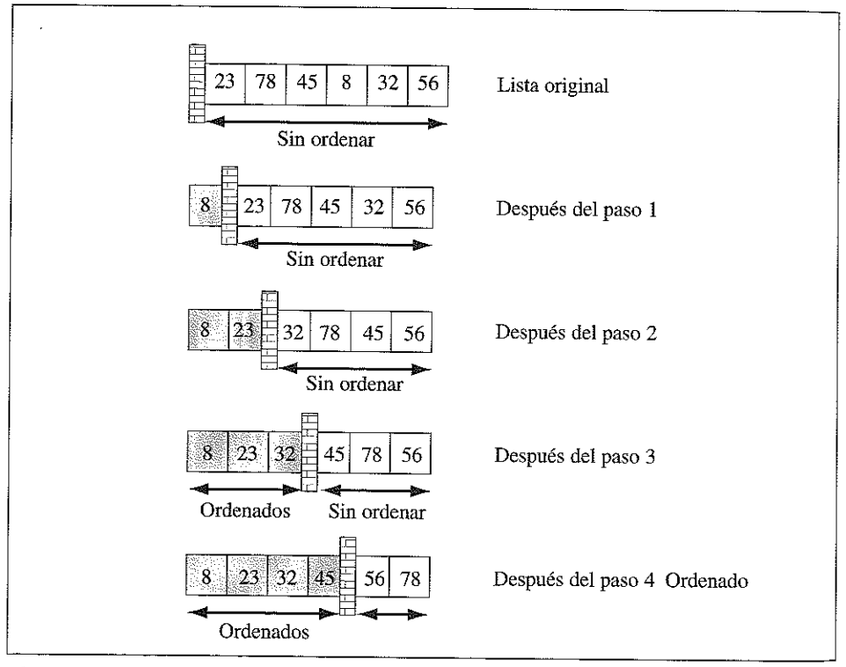


Figura 8.16 Ejemplo de ordenación por burbuja

Algoritmo de ordenación por burbuja La ordenación por burbuja también utiliza dos ciclos, uno dentro del otro. El ciclo exterior se itera para cada paso; cada iteración del ciclo interior trata de burbujear un elemento hacia la parte superior (izquierda). Dejamos el diagrama de flujo y el pseudocódigo como ejercicios.

Ordenación por inserción

El algoritmo de la **ordenación por inserción** es una de las técnicas de ordenamiento más comunes y a menudo la utilizan los jugadores de cartas. Cada carta que un jugador escoge se inserta en el lugar apropiado en su mano para mantener una secuencia particular. (El ordenamiento de las cartas es un ejemplo de ordenación que usa dos criterios para ordenar: seguir el mismo palo y clasificar.)

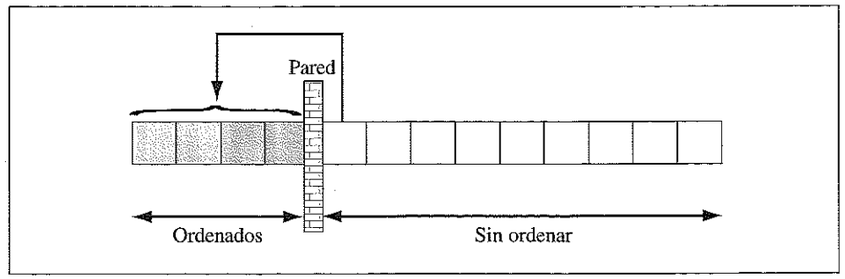


Figura 8.17 Ordenación por inserción

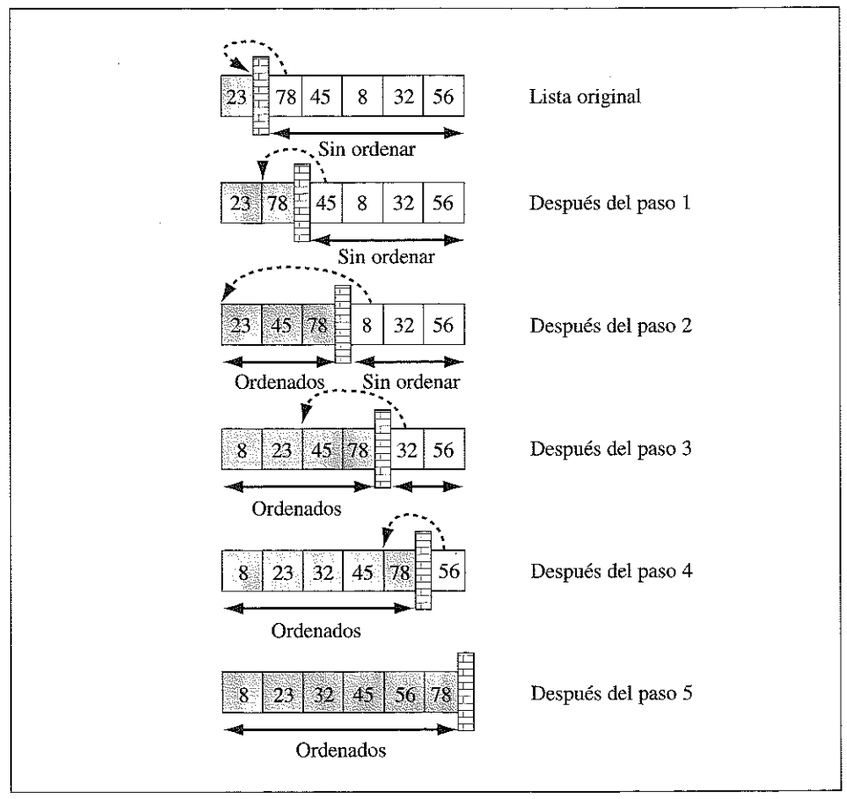


Figura 8.18 Ejemplo de ordenación por inserción

En la ordenación por inserción, al igual que en los otros dos algoritmos de ordenación analizados en este capítulo, la lista se divide en dos partes: ordenada y sin ordenar. En cada paso, se escoge el primer elemento de la sublista sin ordenar, se transfiere a la sublista ordenada y se inserta en el lugar apropiado (figura 8.17). Observe que una lista de n elementos tomará cuando más $n - 1$ pasos para ordenar los datos.

La figura 8.18 sigue la ordenación por inserción a través de nuestra lista de seis números. La pared se mueve con cada paso conforme un elemento se quita de la sublista sin ordenar y se inserta en la sublista ordenada.

Algoritmo de ordenación por inserción El diseño de la ordenación por inserción sigue el mismo patrón visto tanto en la ordenación por selección como en la ordenación por burbuja. El ciclo exterior se itera para cada paso y el ciclo interior encuentra la posición de inserción. Dejamos el diagrama de flujo y el pseudocódigo como ejercicios.

Otros algoritmos de ordenación

Existen otros algoritmos de ordenación: ordenación rápida, ordenación de pila, ordenación de Shell, ordenación por balde, ordenación por fusión, etc. La mayoría de estos algoritmos de ordenación avanzados se analiza en libros sobre estructuras de datos.[1]

Tal vez se pregunte por qué hay tantos algoritmos de ordenación. La razón es el tipo de datos que necesitan ser clasificados. Un algoritmo es más eficiente para una lista que se ordena más frecuentemente, mientras que otro algoritmo es más eficiente para una lista que está completamente desordenada. Para decidir cuál algoritmo es adecuado para una aplicación en particular, se necesita una medición llamada *complejidad de algoritmos*. Este tema se estudia en el capítulo 17, pero una comprensión a fondo requiere tomar cursos adicionales en programación y estructuras de datos.

BÚSQUEDA

Otro algoritmo común en las ciencias de la computación es la **búsqueda**, la cual es el proceso de encontrar la ubicación de un objetivo entre una lista de objetos. En el caso de una lista, la búsqueda significa que dado un valor, usted desea encontrar la ubicación (índice) del primer elemento en la lista que contiene ese valor. El concepto de búsqueda se ilustra en la figura 8.19.

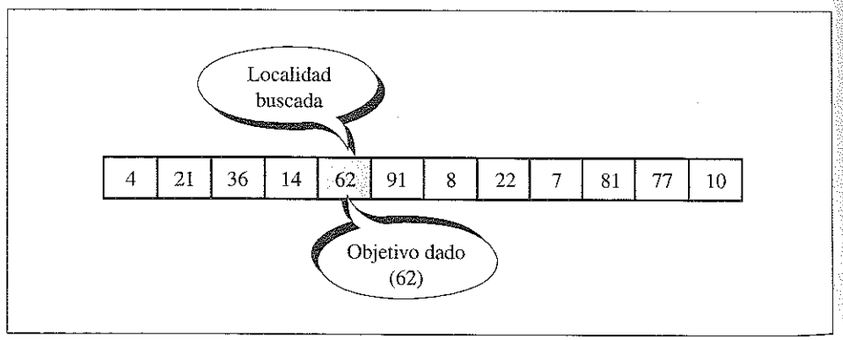


Figura 8.19 Concepto de búsqueda

Hay dos búsquedas básicas para listas: búsqueda secuencial y búsqueda binaria. La búsqueda secuencial puede utilizarse para localizar un elemento en cualquier lista, mientras que la búsqueda binaria requiere que la lista se ordene.

Búsqueda secuencial

La **búsqueda secuencial** se utiliza si la lista en la que se está realizando la búsqueda no está ordenada. Por lo general usted utiliza esta técnica sólo para listas pequeñas o listas que no se consultan a menudo. En otro casos, es mejor primero ordenar la lista y luego realizar la búsqueda usando la búsqueda binaria que se estudia más adelante.

En una búsqueda secuencial usted comienza buscando el objetivo desde el principio de la lista y continúa hasta que encuentre el objetivo o esté seguro de que éste no está en la lista (debido a que usted ha llegado al final de la lista). La figura 8.20 esboza los pasos para encontrar el valor 62.

[1] Véase Richard F. Gilberg y Behrouz A. Forouzan, *Data Structures: A Pseudocode Approach with C++*, (Pacfic Grove, California, Brooks/Cole, 2001), pp. 502-559.

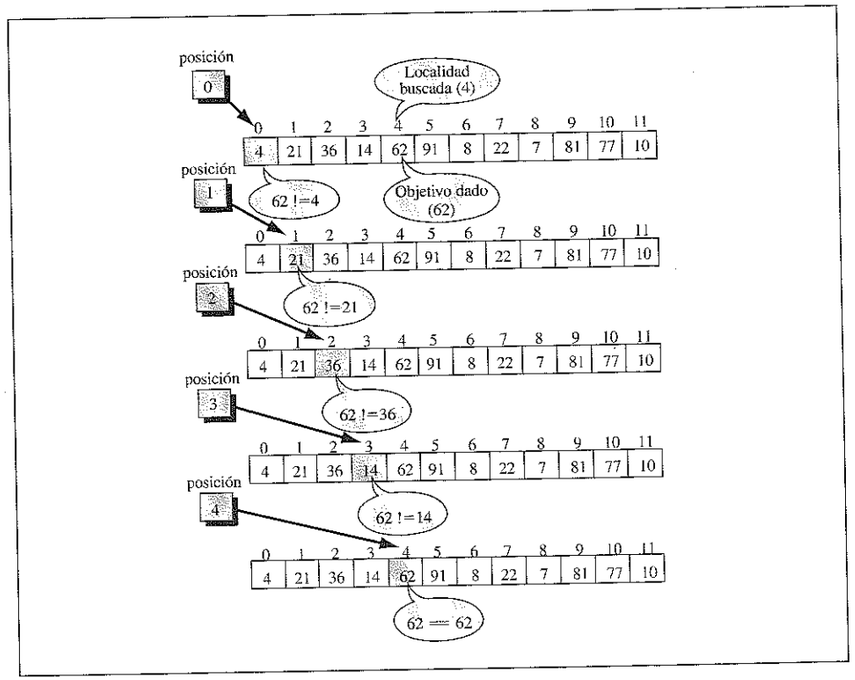


Figura 8.20 Ejemplo de una búsqueda secuencial

Búsqueda binaria

El algoritmo de búsqueda secuencial es muy lento. Si usted tiene una lista de 1 millón de elementos, debe hacer un millón de comparaciones en el peor de los casos. Si la lista no está ordenada, ésta es la única solución. Sin embargo, si la lista está ordenada puede usar un algoritmo más eficiente llamado **búsqueda binaria.** En términos generales, los programadores utilizan una búsqueda binaria cuando una lista es grande.

Una búsqueda binaria comienza al probar los datos en el elemento que está a mitad de la lista. Esto determina si el objetivo está en la primera o la segunda mitad de la lista. Si está en la primera mitad no hay necesidad de hacer una revisión posterior en la segunda mitad. Si está en la segunda mitad, no hay necesidad de revisar la primera. En otras palabras, usted descarta el estudio de la mitad de la lista en cada paso. Repita este proceso hasta que se encuentre el objetivo o compruebe que éste no está en la lista. La figura 8.21 muestra cómo encontrar el objetivo, 22, en la lista usando tres referencias: *primero*, *medio* y *último*.

1. Al principio, el *primero* muestra 0 y el *último* muestra 11. Sea el *medio* la posición de en medio, (0 + 11)/2, o 5. Ahora se compara el objetivo (22) con los datos en la posición 5(21). El objetivo es mayor que este valor, así que se ignora la primera mitad.

2. El *primero* se mueve después del *medio* a la posición 6. Sea el *medio* la parte media de la segunda mitad, (6 + 11)/2, u 8. Ahora se compara el objetivo (22) con los datos de la posición 8 (62). El objetivo es menor que este valor, así que los números de este valor (62) se ignoran hasta el final.

3. El *último* se mueve antes del *medio* a la posición 7. El *medio* se calcula de nuevo, (7 + 6)/2, o 6. El objetivo (22) se compara con el valor de esta posición (22). Usted ha encontrado el objetivo y puede salir.

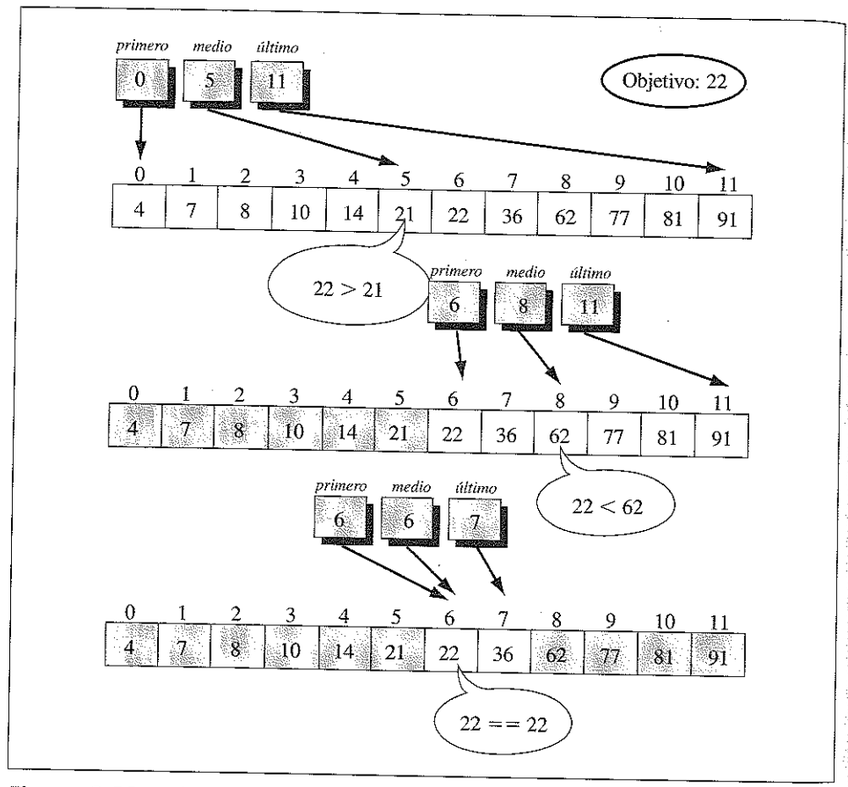


Figura 8.21 Ejemplo de una búsqueda binaria

8.7 RECURSIVIDAD

En general, hay dos métodos para escribir algoritmos que resuelvan un problema. Uno utiliza iteraciones y el otro usa la recursividad. La **recursividad** es un proceso mediante el cual un algoritmo se llama a sí mismo.

DEFINICIÓN ITERATIVA

Para estudiar un ejemplo simple, considere el cálculo de un factorial. El factorial de un número es el resultado de los valores absolutos desde 1 hasta el número. La definición es iterativa (figura 8.22). Un algoritmo es iterativo siempre que la definición no involucra al algoritmo mismo.

$$\text{Factorial } (n) = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 & \text{si } n > 0 \end{cases}$$

Figura 8.22 Definición iterativa de factorial

$$\text{Factorial } (n) = \begin{cases} 1 & \text{si } n = 0 \\ n \times \text{Factorial } (n-1) & \text{si } n > 0 \end{cases}$$

Figura 8.23 Definición recursiva de factorial

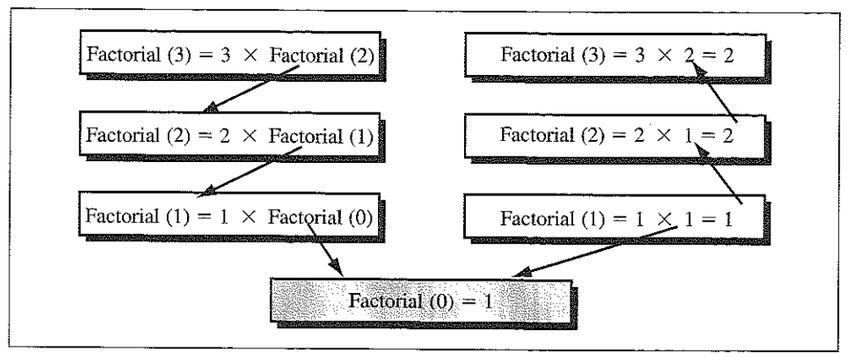


Figura 8.24 Trazado de la solución recursiva del problema factorial

DEFINICIÓN RECURSIVA

Un algoritmo se define de manera recursiva siempre que el algoritmo aparece dentro de la definición misma. Por ejemplo, la función factorial puede definirse de manera recursiva como en la figura 8.23.

La descomposición de factorial (3), usando la recursividad, se muestra en la figura 8.24. Si usted estudia la figura con cuidado, notará que la solución recursiva para un problema involucra un proceso de dos sentidos. Primero se descompone el problema de arriba abajo y luego se resuelve de abajo para arriba.

A juzgar por este ejemplo, parece como si el cálculo recursivo es mucho más grande y más difícil. Así que ¿por qué querría usar el método recursivo? Aunque el método recursivo parece más difícil cuando se usa papel y pluma, a menudo es una solución mucho más fácil y más elegante cuando se usan computadoras. Asimismo, ofrece una simplicidad conceptual para el creador y el lector.

Solución iterativa

Escribamos un algoritmo para resolver el problema factorial de modo iterativo. Esta solución por lo general implica un ciclo como el que aparece en el algoritmo 8.7.

Algoritmo 8.7 Factorial iterativo

```
Factorial
Entrada: Un entero positivo num
1. Establece FactN en 1
2. Establece i en 1
3. while (i menor o igual que num)
        3.1 Establece FactN a FactN × i
        3.2 Incrementa i
   End while
4. Devuelve FactN
   End
```

Solución recursiva

La solución recursiva para un factorial se ve en el algoritmo 8.8. No se necesita un ciclo; el concepto por sí mismo implica repetición. En la versión recursiva, usted permite que el algoritmo *factorial* se llame a sí mismo.

Algoritmo 8.8 Factorial recursivo

```
Factorial
Entrada: Un entero positivo num
1. if (num es igual a 0)
   then
   1.1 devuelve 1
   else
   1.2 devuelve num × Factorial (num − 1)
   End if
   End
```

8.8 TÉRMINOS CLAVE

algoritmo
búsqueda
búsqueda binaria
búsqueda secuencial
carta estructurada
datos de entrada
datos de salida
diagrama de flujo
función
iteración
módulo
ordenación
ordenación por burbuja
ordenación por inserción
ordenación por selección
paso de ordenación
procedimiento
pseudocódigo
recursividad
subalgoritmo
subprograma
subrutina
sumatoria

8.9 RESUMEN

- De manera informal, un algoritmo es un método paso a paso para resolver un problema o realizar una tarea.
- Un algoritmo acepta una lista de entrada de datos y crea una lista de salida de datos.
- Un programa es una combinación de estructuras de secuencia, estructuras de decisión y estructuras de repetición.
- Un diagrama de flujo es una representación pictórica de un algoritmo.
- El pseudocódigo es una representación de un algoritmo parecida al inglés.
- Formalmente, un algoritmo es una serie ordenada de pasos precisos que produce un resultado y termina en un tiempo finito.
- Un algoritmo puede dividirse en unidades más pequeñas llamadas subalgoritmos.
- Una carta estructurada es una herramienta de diseño de alto nivel que muestra la relación entre diferentes módulos de un programa.
- La sumatoria es un algoritmo básico en el cual los números se suman.
- La multiplicación es un algoritmo básico en el cual los números se multiplican.
- Encontrar el mínimo o el máximo en una lista de números es un algoritmo básico.
- La ordenación, un proceso para acomodar los datos en orden, es un algoritmo básico.
- La ordenación por selección, la ordenación por burbuja y la ordenación por inserción son algoritmos de ordenación de uso común.
- La búsqueda, un proceso para localizar un objetivo en una lista de datos, es un algoritmo básico.
- La búsqueda secuencial se usa para listas sin ordenar.
- La búsqueda binaria se usa para listas ordenadas.
- Un algoritmo iterativo implica sólo los parámetros y no el algoritmo mismo.
- Un algoritmo recursivo implica el algoritmo mismo.

8.10 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuál es la definición formal de algoritmo?
2. Defina las tres estructuras utilizadas en la programación estructurada.
3. ¿Cómo se relaciona un diagrama de flujo con un algoritmo?
4. ¿Cómo se relaciona el pseudocódigo con un algoritmo?
5. ¿Cuáles son los otros términos para las unidades que conforman un algoritmo?
6. ¿Cómo utilizan los programadores la carta estructurada?
7. ¿Cuál es el propósito del algoritmo de sumatoria?
8. ¿Cuál es el propósito del algoritmo de multiplicatoria?
9. ¿Cuál es el propósito de un algoritmo de ordenación?
10. ¿Cuáles son los tres tipos de algoritmos de ordenación?
11. ¿En qué se parecen los tres algoritmos de ordenación? ¿En qué difieren?
12. ¿Cuál es el propósito de un algoritmo de búsqueda?
13. ¿Cuáles son los dos tipos de búsqueda principales? ¿Cuál es la diferencia entre ellos?
14. Dé una definición y un ejemplo de un proceso iterativo.
15. Dé una definición y un ejemplo de un proceso recursivo.

PREGUNTAS DE OPCIÓN MÚLTIPLE

16. _____ es un método paso a paso para resolver un problema o realizar una tarea.
    a. Una estructura
    b. Una recursividad
    c. Una iteración
    d. Un algoritmo

17. Para establecer el valor de una variable antes de que ocurra cualquier procesamiento, usted _____ la variable.
    a. estructura
    b. itera
    c. inicializa
    d. incrementa

18. En las ciencias de la computación hay _____ estructuras básicas.
    a. una
    b. dos
    c. tres
    d. cuatro

19. La estructura _____ prueba una condición.
    a. de secuencia
    b. de decisión
    c. de repetición
    d. lógica

20. La estructura _____ es cualquier acción.
    a. de secuencia
    b. de decisión
    c. de repetición
    d. lógica

21. La estructura _____ maneja acciones repetitivas.
    a. de secuencia
    b. de decisión
    c. de repetición
    d. lógica

22. _____ es una representación pictórica de un algoritmo.
    a. Un diagrama de flujo
    b. Una carta estructurada
    c. El pseudocódigo
    d. Un algoritmo

23. _____ es una representación del código parecida al inglés.
    a. Un diagrama de flujo
    b. Una carta estructurada
    c. El pseudocódigo
    d. Un algoritmo

24. _____ es una herramienta de alto nivel que muestra la relación entre diferentes módulos de un programa.
    a. Un diagrama de flujo
    b. Una carta estructurada
    c. El pseudocódigo
    d. Un algoritmo

25. Un subalgoritmo también se conoce como _____.
- **a.** función
- **b.** subrutina
- **c.** módulo
- **d.** todos los anteriores

26. _____ es un algoritmo básico que encuentra el producto de una lista de números.
- **a.** La sumatoria
- **b.** La multiplicatoria
- **c.** El Menor
- **d.** El Mayor

27. La _____ es un algoritmo básico que acomoda los datos de acuerdo con sus valores.
- **a.** solicitud de información
- **b.** ordenación
- **c.** búsqueda
- **d.** recursividad

28. _____ es un algoritmo básico que suma una lista de números.
- **a.** La sumatoria
- **b.** La multiplicatoria
- **c.** El Menor
- **d.** El Mayor

29. _____ es un algoritmo básico que encuentra el número menor de una lista de números.
- **a.** La sumatoria
- **b.** La multiplicatoria
- **c.** El Menor
- **d.** El Mayor

30. En la ordenación por _____, los elementos se dividen en dos listas: ordenados y sin ordenar.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

31. Para la ordenación por _____, se necesitan $n - 1$ pasos para ordenar los datos.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

32. Para la ordenación por _____, se necesitan dos ciclos.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

33. En la ordenación por _____, el elemento que va a la lista ordenada siempre es el primer elemento en la lista sin ordenar.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

34. En la ordenación por _____, el elemento más pequeño de la lista sin ordenar se intercambia con el elemento al principio de la lista sin ordenar.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

35. En la ordenación por _____, el elemento más pequeño se mueve al principio de la lista sin ordenar. No hay intercambio uno a uno.
- **a.** selección
- **b.** burbuja
- **c.** inserción
- **d.** todos los anteriores

36. _____ es un algoritmo básico en el cual usted quiere encontrar la localidad de un objetivo en una lista de elementos.
- **a.** La ordenación
- **b.** La búsqueda
- **c.** La multiplicatoria
- **d.** La sumatoria

37. Utilice una búsqueda _____ para una lista sin ordenar.
- **a.** secuencial
- **b.** binaria
- **c.** por burbuja
- **d.** por inserción

38. Utilice una búsqueda _____ para una lista ordenada.
- **a.** secuencial
- **b.** binaria
- **c.** por burbuja
- **d.** por inserción

39. La _____ es un proceso en el cual un algoritmo se llama a sí mismo.
- **a.** inserción
- **b.** búsqueda
- **c.** recursividad
- **d.** iteración

EJERCICIOS

40. Escriba un pseudocódigo para el algoritmo de sumatoria definido en la figura 8.10.

41. Escriba un pseudocódigo para el algoritmo de multiplicatoria definido en la figura 8.11.

42. Haga un diagrama de flujo para un algoritmo que encuentre el número menor entre N números.

43. Escriba un pseudocódigo para el algoritmo del número menor del ejercicio 42.

44. Haga un diagrama de flujo para un algoritmo que encuentre el número mayor entre N números.

45. Escriba un pseudocódigo para el algoritmo del número mayor del ejercicio 44.

46. Escriba un pseudocódigo para el algoritmo de ordenación por selección.

47. Haga un diagrama de flujo y escriba un pseudocódigo para el algoritmo de ordenación por burbuja.

48. Dibuje un diagrama de flujo y escriba un pseudocódigo para el algoritmo de ordenación por inserción.

49. Usando el algoritmo de ordenación por selección, ordene la siguiente lista en forma manual y muestre su trabajo en cada paso:

14 7 23 31 40 56 78 9 2

50. Usando el algoritmo de ordenación por burbuja, ordene la siguiente lista en forma manual y muestre su trabajo en cada paso:

14 7 23 31 40 56 78 9 2

51. Usando el algoritmo de ordenación por inserción, ordene la siguiente lista en forma manual y muestre su trabajo en cada paso:

14 7 23 31 40 56 78 9 2

52. Una lista contiene los siguientes elementos. Los primeros dos elementos se han ordenado usando el algoritmo de ordenación por selección. ¿Cuál es el valor de los elementos en la lista después de tres pasos más de la ordenación por selección?

7 8 26 44 13 23 98 57

53. Una lista contiene los siguientes elementos. Los primeros dos elementos se han ordenado usando el algoritmo de ordenación por burbuja. ¿Cuál es el valor de los elementos en la lista después de tres pasos más de la ordenación por burbuja?

7 8 26 44 13 23 57 98

54. Una lista contiene los siguientes elementos. Los primeros dos elementos se han ordenado usando el algoritmo de ordenación por inserción. ¿Cuál es el valor de los elementos en la lista después de tres pasos más de la ordenación por inserción?

3 13 7 26 44 23 98 57

55. Una lista contiene los siguientes elementos. Usando el algoritmo de búsqueda binaria, detalle los pasos que se siguieron para encontrar el número 88. En cada paso, muestre los valores del *primero, último* y *medio.*

8 13 17 26 44 56 88 97

56. Una lista contiene los siguientes elementos. Usando el algoritmo de búsqueda binaria, detalle los pasos que se siguieron para encontrar 20. En cada paso, muestre los valores de *primero, último* y *medio.*

8 13 17 26 44 56 88 97

57. Escriba un algoritmo recursivo para encontrar el máximo común divisor (mcd) de dos enteros utilizando la definición de la figura 8.25.

$$\text{mcd}(x, y) = \begin{cases} \text{mcd}(y, x) & \text{si } x < y \\ x & \text{si } y = 0 \\ \text{mcd}(y, x \bmod y) & \text{en caso contrario} \end{cases}$$

Figura 8.25 Ejercicio 57

58. Escriba un algoritmo recursivo para encontrar la combinación de n objetos tomando k a la vez utilizando la definición de la figura 8.26.

$$C(n, k) = \begin{cases} 1 & \text{si } k = 0 \text{ o } n = k \\ C(n-1, k) + C(n-1, k-1) & \text{si } n > k > 0 \end{cases}$$

Figura 8.26 Ejercicio 58

Lenguajes de programación

En este capítulo damos una visión general de los lenguajes de programación para dar a los estudiantes una idea de los distintos lenguajes y cuándo se usan. Primero estudiamos la evolución de los lenguajes. Enseguida clasificamos los lenguajes de acuerdo con su método para resolver problemas y los tipos de problemas que pueden resolver. Luego estudiamos un lenguaje popular y analizamos sus elementos y capacidades.

9.1 EVOLUCIÓN

Para escribir un programa para computadora, usted debe utilizar un lenguaje de computadora. Un **lenguaje de computadora** es una serie de palabras predefinidas que se combinan en un programa de acuerdo con reglas predefinidas (**sintaxis**).

Con los años, los lenguajes de computadora han evolucionado desde lenguaje de máquina a lenguajes naturales. Una línea de tiempo para lenguajes de computadora se presenta en la figura 9.1.

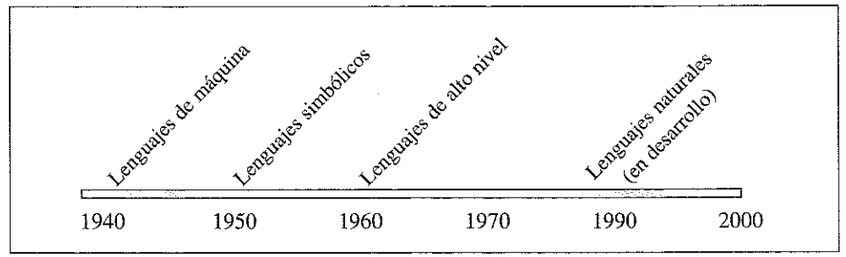


Figura 9.1 Evolución de los lenguajes de computadora

LENGUAJES DE MÁQUINA

En los primeros días de las computadoras, los únicos lenguajes de programación disponibles eran los **lenguajes de máquina**. Cada computadora tenía su propio lenguaje de máquina, el cual está formado por cadenas de 0 y 1. El programa 9.1 muestra un ejemplo de un programa en lenguaje de máquina. Este programa multiplica dos números e imprime el resultado.

```
 1. 00000000  00000100  0000000000000000
 2. 01011110  00001100  11000010  0000000000000010
 3.           11101111  00010110  0000000000000101
 4.           11101111  10011110  0000000000001011
 5. 11111000  10101101  11011111  0000000000010010
 6.           01100010  11011111  0000000000010101
 7. 11101111  00000010  11111011  0000000000010111
 8. 11110100  10101101  11011111  0000000000011110
 9. 00000011  10100010  11011111  0000000000100001
10. 11101111  00000010  11111011  0000000000100100
11. 01111110  11110100  10101101
12. 11111000  10101110  11000101  0000000000101011
13. 00000110  11111011  11000101  0000000000110001
14. 11101111  00000010  11111011  0000000000110100
15.                     00000100  0000000000111101
16.                     00000100  0000000000111101
```

Programa 9.1 Programa en lenguaje de máquina

Las instrucciones en lenguaje de máquina deben estar en cadenas de 0 y 1 debido a que el circuito interno de una computadora está formado por interruptores transistores y otros dispositivos electrónicos que pueden estar en uno de dos estados: encendido o apagado. El estado apagado se representa por 0; el estado encendido, mediante 1.

El único lenguaje comprendido por una computadora es el lenguaje de máquina.

LENGUAJES SIMBÓLICOS

Se vuelve evidente que no se escribirían muchos programas si los programadores continuaran trabajando en lenguaje de máquina. A principios de la década de 1950, Grace Hooper, un matemático y miembro de la Armada de Estados Unidos, desarrolló el concepto de un lenguaje que simplemente reflejaba los lenguajes de máquina utilizando símbolos, o ayuda nemotécnica, para representar las diversas instrucciones del lenguaje de máquina. Dado que utilizaban símbolos, estos lenguajes se conocían como **lenguajes simbólicos**. El programa 9.2 muestra el programa de multiplicación en un lenguaje simbólico.

```
1.  entry main,^m<r2>
2.  sub12 #12,sp
3.  jsb   C$MAIN_ARGS
4.  movab $CHAR_STRING_CON
5.
6.  pushal  -8(fp)
7.  pushal  (r2)
8.  calls   #2,read
9.  pushal  -12(fp)
10. pushal  3(r2)
11. calls   #2,read
12. mull3   -8(fp),-12(fp),-
13. pusha   6(r2)
14. calls   #2,print
15. clrl            r0
16. ret
```

Programa 9.2 Programa en lenguaje simbólico

Un programa especial llamado **ensamblador** se usa para traducir el código simbólico en lenguaje de máquina. Como los lenguajes simbólicos debían ensamblarse en lenguaje de máquina, pronto se conocieron como **lenguajes ensambladores**. Este nombre aún se utiliza en la actualidad para los lenguajes simbólicos que representan rigurosamente el lenguaje de máquina.

LENGUAJES DE ALTO NIVEL

Aunque los lenguajes simbólicos mejoraron en gran medida la eficiencia de la programación, aún requerían que los programadores se concentraran en el hardware que estaban usando.Trabajar con lenguajes simbólicos también era muy tedioso debido a que cada instrucción de máquina debía codificarse individualmente. El deseo de mejorar la eficiencia del programador y cambiar el centro de atención de la computadora al problema que se estaba resolviendo condujo al desarrollo de los **lenguajes de alto nivel**.

Los lenguajes de alto nivel se pueden portar a muchas computadoras diferentes, permitiendo al programador concentrarse en la aplicación en lugar de las complejidades de la computadora. Se diseñaron para liberar al programador de los detalles del lenguaje ensamblador. Los lenguajes de alto nivel comparten una característica con los lenguajes simbólicos: deben convertirse a lenguaje de máquina. Este proceso se llama *compilación*.

Con los años, se desarrollaron varios lenguajes, en particular BASIC, COBOL, Pascal, Ada, C, C++ y Java. El programa 9.3 muestra el programa de multiplicación del programa 9.2 como si estuviera en lenguaje C++.


```
1.  /* Este programa lee dos números enteros desde el
2.  teclado e imprime su producto.
3.  Escrito por:
4.  Fecha:
5.  */
6.  #include <iostream.h>
7.
8.  int main (void)
9.  {
10. // Declaraciones locales
11.         int number1;
12.         int number2;
13.         int result;
14.
15. // Instrucciones
16.         cinn >> number1;
17.         cinn >> number2;
18.         result = number1 * number2;
19.         cout << result;
20.         return 0;
21. }       // principal
```

Programa 9.3 Programa en lenguaje C++

LENGUAJES NATURALES

Idealmente, usted podría usar su **lenguaje natural** (por ejemplo, el idioma inglés, francés o chino) y la computadora lo comprendería y ejecutaría sus solicitudes de inmediato. Aunque esto puede parecer ciencia ficción, actualmente se está realizando un trabajo considerable sobre los lenguajes naturales en los laboratorios. Hasta el momento, su uso en la industria todavía es bastante limitado.

9.2 ESCRIBIR UN PROGRAMA

Como se vio en la sección anterior, una computadora comprende un programa sólo si el programa se traduce a su lenguaje de máquina. En esta sección, explicamos el procedimiento para convertir un programa en un lenguaje de alto nivel a un lenguaje de máquina.

Es tarea del programador escribir el programa y luego convertirlo en un **archivo ejecutable** (lenguaje de máquina). Hay tres pasos en este proceso:

1. Escribir y editar el programa.
2. Compilar el programa.
3. Vincular el programa con los módulos de biblioteca requeridos.

La figura 9.2 muestra estos tres procesos.

ESCRITURA Y EDICIÓN DE PROGRAMAS

El software utilizado para escribir programas se conoce como **editor de texto**. Un editor de texto ayuda a introducir, cambiar y almacenar datos de caracteres. Dependiendo del editor en su sistema, usted podría usarlo para escribir cartas, crear informes o escribir programas. La gran diferencia entre las otras formas de procesamiento de texto y escribir programas es que

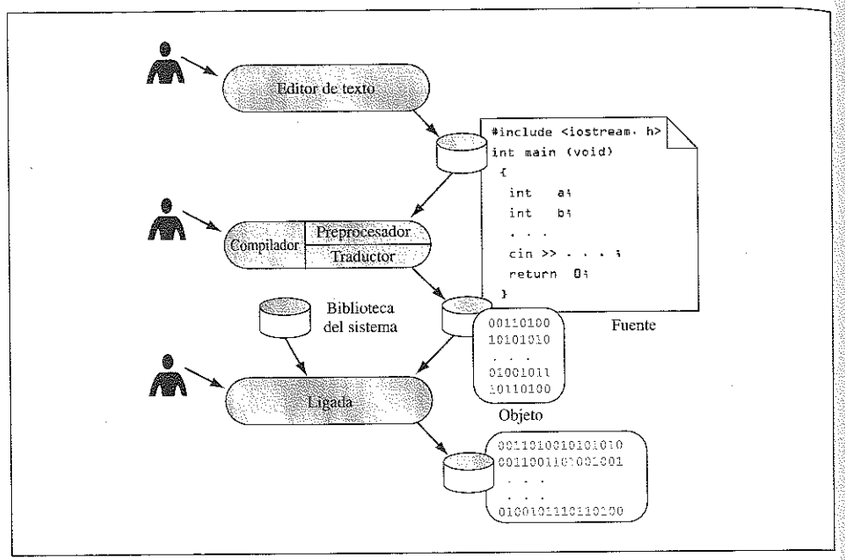


Figura 9.2 Construcción de un programa

los programas se orientan en torno a líneas de código mientras que la mayor parte del procesamiento de texto se orienta en torno a caracteres y líneas. Después de completar un programa, usted guarda su archivo en el disco. Este archivo se meterá al compilador; a este archivo se le conoce como **archivo fuente.**

COMPILACIÓN DE PROGRAMAS

La información en un archivo fuente almacenada en el disco debe traducirse a lenguaje de máquina de modo que la computadora pueda comprenderlo. La mayoría de los lenguajes de alto nivel utiliza un compilador para hacerlo. El compilador en realidad son dos programas separados: el **preprocesador** y el **traductor**.

El preprocesador lee el código fuente y lo prepara para el traductor. Cuando prepara el código, busca comandos especiales conocidos como **directivas de preprocesador**. Estas directivas indican al preprocesador que busque bibliotecas de código especiales, haga sustituciones en el código y prepare de algunas otras maneras el código para su traducción a lenguaje de máquina. El resultado del preprocesamiento se llama **unidad de traducción**.

Después de que el preprocesador ha preparado el código para su compilación, el traductor realiza el trabajo de convertir la unidad de traducción a lenguaje de máquina. El traductor lee la unidad de traducción y escribe el **módulo de objeto** resultante en un archivo que luego puede combinarse con otras unidades precompiladas para formar el programa final. Un módulo de objeto es el código en lenguaje de máquina. Aun cuando la salida del compilador está en lenguaje de máquina no está listo para ejecutarse; es decir, aún no es ejecutable porque no tiene todas las partes requeridas.

LIGADOR DE PROGRAMAS

Un lenguaje de alto nivel tiene muchos **subprogramas**, algunos de los cuales están escritos por usted y son parte de su programa fuente. Sin embargo, existen otros subprogramas como los procesos de entrada/salida y subrutinas de biblioteca matemáticas que existen en algunas otras partes y deben ser adjuntadas a su programa. El **ligador** ensambla todas estas funciones, las de usted y las del sistema, en su programa ejecutable final.

9.3 EJECUCIÓN DE PROGRAMAS

Una vez que su programa se ha ligado, está listo para su ejecución. Para ejecutar su programa usted utiliza un comando del sistema operativo, como *run*, para cargar su programa en la memoria principal y ejecutarlo. Meter el programa en la memoria es la función de un programa de sistema operativo conocido como el **cargador**, el cual localiza el programa ejecutable y lo lee en la memoria. Cuando todo está listo se da el control al programa y comienza la ejecución.

En la ejecución de un programa típico, el programa lee datos para su procesamiento ya sea desde el usuario o desde un archivo. Después de que el programa procesa los datos, prepara la salida. La salida de los datos puede ser hacia el monitor del usuario o a un archivo. Cuando el programa está terminado, se lo indica al sistema operativo, el cual elimina el programa de la memoria. La ejecución de un programa en un entorno de computadora personal aparece esquematizada en la figura 9.3.

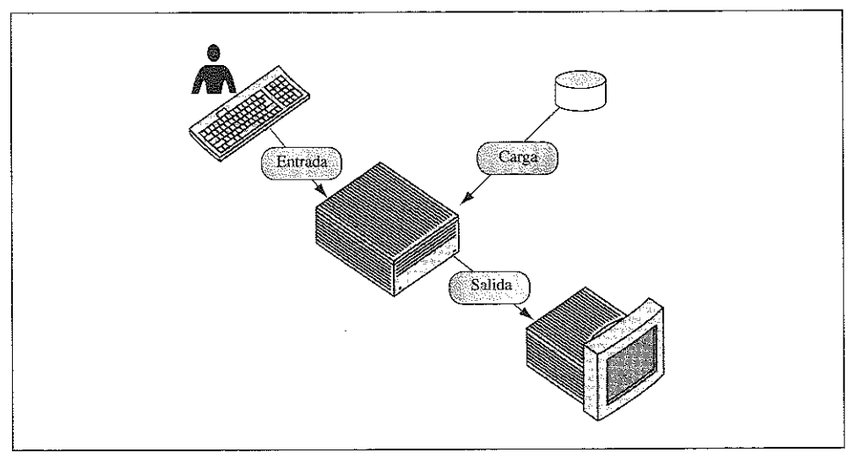


Figura 9.3 Ejecución de un programa

9.4 CATEGORÍAS DE LENGUAJES

Actualmente, los lenguajes de computadora se clasifican de acuerdo con el método que utilizan para resolver un problema y la categoría de problemas que resuelven. Dividimos los lenguajes de computadora en cinco categorías: lenguajes procedurales (imperativos), lenguajes orientados a objetos, lenguajes funcionales, lenguajes declarativos y lenguajes especiales (figura 9.4).

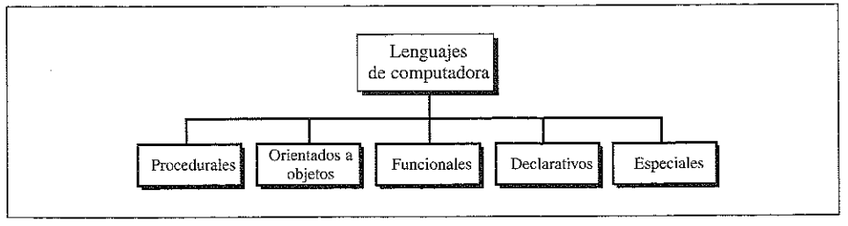


Figura 9.4 Categorías de lenguajes

LENGUAJES PROCEDURALES (IMPERATIVOS)

Un lenguaje procedural, o imperativo, utiliza el método tradicional de programación. Sigue el mismo método empleado por el hardware de computadora para ejecutar un programa (buscar y traer, decodificar, ejecutar). Un **lenguaje procedural** es una serie de instrucciones que se ejecutan una por una de principio a fin a menos que una instrucción mande el control a alguna otra parte. Incluso en este caso, el programa todavía requiere una serie de instrucciones que se ejecutan una después de otra, aunque algunas podrían ejecutarse más de una vez o algunas podrían saltarse.

Cuando los programadores necesitan resolver un problema usando uno de los lenguajes procedurales, deben saber cuál es el *procedimiento* a seguir. En otras palabras, para cada problema el programador debe diseñar un algoritmo cuidadosamente, y el algoritmo debe traducirse con cuidado a instrucciones.

Cada instrucción en un lenguaje procedural manipula los elementos de datos (cambia los valores almacenados en localidades de la memoria o los mueve a alguna otra parte) o es una instrucción de control que localiza la siguiente instrucción a ser ejecutada. Por esta razón, un lenguaje procedural a veces se llama **lenguaje imperativo**: cada instrucción es un comando para que el sistema de computación realice alguna tarea específica.

Se han desarrollado varios lenguajes procedurales de alto nivel durante las últimas décadas: FORTRAN, COBOL, Pascal, C y Ada. En esta sección presentamos brevemente características de cada lenguaje. Más adelante en este capítulo analizamos los elementos de C con más detalle.

FORTRAN

FORTRAN (*FORmula TRANslation*: traducción de fórmulas), diseñado por un grupo de ingenieros de IBM bajo la supervisión de Jack Backus, estaba disponible comercialmente en 1957. FORTRAN fue el primer lenguaje de alto nivel. Tiene algunas características que, después de cuatro décadas, aún lo hacen un lenguaje ideal para aplicaciones científicas y de ingeniería. Estas características pueden resumirse como sigue:

- Aritmética de alta precisión
- Capacidad de manejo de números complejos
- Cálculo de exponentes (a^b)

Durante los últimos 40 años, FORTRAN ha pasado por varias versiones:

1. FORTRAN
2. FORTRAN II
3. FORTRAN IV
4. FORTRAN 77
5. FORTRAN 99
6. HPF (*High Performance* FORTRAN: FORTRAN de alto rendimiento)

La versión más reciente (HPF) se utiliza en computadoras con multiprocesadores de alta velocidad.

COBOL

COBOL (*COmmon Business-Oriented Language*: Lenguaje común orientado a negocios) fue diseñado por un grupo de científicos de computación dirigido por Grace Hopper de la Armada de Estados Unidos. COBOL tiene una meta de diseño específica: servir como un lenguaje de programación para negocios. Los problemas a resolver en el medio empresarial son completamente diferentes de aquellos en un entorno de ingeniería. Los programas para negocios no necesitan los cálculos precisos que requieren los programas de ingeniería. Las necesidades de programación del mundo de los negocios pueden resumirse como sigue:

- Acceso rápido a archivos y bases de datos
- Actualización rápida de archivos y bases de datos
- Generación de una gran cantidad de informes
- Salida con un formato comprensible para el usuario

COBOL se diseñó para satisfacer todas estas necesidades.

Pascal

Niklaus Wirth inventó **Pascal** en 1971 en Zurich, Suiza. Recibió su nombre en honor a Blaise Pascal, el matemático y filósofo francés del siglo XVII que inventó la calculadora Pascalina.

Pascal se diseñó con un objetivo específico en mente: enseñar programación a los novatos al enfatizar el método de programación estructurada. Aun cuando Pascal se convirtió en el lenguaje más popular en la academia, nunca logró la misma popularidad en la industria. Los lenguajes de procedimiento actuales le deben mucho a este lenguaje.

C

El **lenguaje C** Se desarrolló a principios de la década de 1970 por Dennis Ritchie en Bell Laboratories. Originalmente estaba pensado para escribir sistemas operativos y software del sistema (la mayor parte del sistema operativo UNIX está escrito en C). Posteriormente, se volvió popular entre los programadores por varias razones:

- C tiene todas las instrucciones de alto nivel que un lenguaje de programación estructurada de alto nivel debe tener; oculta los detalles del hardware al programador.
- C también tiene algunas de las instrucciones de bajo nivel que permiten al programador tener acceso al hardware de manera directa y rápida; C está más cerca de ser un lenguaje ensamblador que cualquier otro lenguaje. Esto lo vuelve un buen lenguaje para los programadores de sistemas.
- C es un lenguaje muy eficiente; sus instrucciones son breves. Algunas veces un símbolo en C puede realizar la misma tarea que una palabra larga en un lenguaje como COBOL. Esta concisión atrae a los programadores que quieren escribir programas cortos.
- C ha sido estandarizado por el ANSI y la ISO.

Algunos elementos del lenguaje C se estudian con más detalle al final de este capítulo.

Ada

Ada recibió su nombre por Augusta Ada Byron, la hija de Lord Byron y asistente de Charles Babbage (el inventor de la máquina analítica). El Departamento de Defensa (DoD) de Estados Unidos creó este lenguaje para que fuera el lenguaje uniforme usado por todos los contratistas del DoD.

Ada tiene tres características que lo vuelven un lenguaje muy utilizado por el DoD, las computadoras jumbo y la industria:

- Tiene instrucciones de alto nivel como otros lenguajes procedurales.
- Tiene instrucciones que permiten el procesamiento en tiempo real. Esto lo vuelve adecuado para el control de procesos.
- Tiene capacidades de procesamiento paralelo. Puede ejecutarse en computadoras mainframe con múltiples procesadores.

LENGUAJES ORIENTADOS A OBJETOS

La programación orientada a objetos utiliza un método para la solución de problemas totalmente diferente de la programación de procedimientos. Usted puede pensar en un elemento de datos en ambos lenguajes como un objeto. Un programa puede imaginarse como una serie de operaciones que desea realizar en el objeto.

En la programación procedural, los objetos están completamente separados y son totalmente independientes de las operaciones. Los objetos pueden almacenarse en una computadora y diferentes programas que utilizan aplicaciones distintas se aplican a ellos. En otras palabras, en la programación procedural los objetos son *pasivos*. No tienen ninguna operación definida para ellos. Los programadores definen las operaciones y las aplican a los objetos.

En la programación orientada a objetos, los objetos y las operaciones que se van a aplicar en ellos están ligados. El programador primero define un objeto y los tipos de operaciones que pueden aplicarse a este objeto. El programador (u otro programador) puede entonces usar esta combinación e invocar algunas o todas las operaciones definidas para resolver un problema. En otras palabras, los objetos en la programación orientada a objetos están *activos*.

La idea es similar para los objetos en la vida real. Existen objetos que son pasivos, como una piedra. No hay operación incluida con la piedra. Si alguien quiere realizar una operación en una piedra, él o ella define la operación y la realiza. Por otro lado, un automóvil es un objeto activo. Hay varias operaciones definidas para un automóvil. El conductor sólo necesita invocar a una de estas operaciones.

Varios **lenguajes orientados a objetos** se han desarrollado. Estudiamos brevemente las características de dos: C++ y Java.

C++

El **lenguaje C++** fue desarrollado por Bjarne Stroustrup en Bell Laboratories como un lenguaje C mejor. Utiliza **clases** para definir las características generales de objetos similares y las operaciones que pueden aplicarse a ellos. Por ejemplo, un programador puede definir una clase `Formas_geométricas` y todas las características comunes para dos formas geométricas bidimensionales tales como el centro, el número de lados y así por el estilo. La clase también puede definir operaciones (funciones o métodos) que pueden aplicarse a una forma geométrica tales como el cálculo y la impresión del área, el cálculo y la impresión del perímetro, la impresión de las coordenadas del punto central, etc. Por tanto, se puede escribir un programa para crear diferentes objetos del tipo `Formas_geométricas`. Cada objeto puede tener un centro localizado en un punto distinto y un número diferente de lados. El programa puede entonces calcular e imprimir el área, el perímetro y la ubicación del centro para cada objeto.

En el diseño del lenguaje C++ se utilizaron tres principios: encapsulamiento, herencia y polimorfismo.

Encapsulamiento El **encapsulamiento** es la idea de ocultar los datos y algunas operaciones que pueden realizarse en los datos dentro del objeto. El usuario de objetos normalmente no tiene acceso directo a los datos. En vez de ello, se accede a los datos a través de una interfaz (una llamada a una serie particular de operaciones). En otras palabras, el usuario sabe qué se hace con los datos sin saber cómo se hace.

Herencia En C++, como en la naturaleza, un objeto puede heredar algo de otro objeto. A este concepto se le conoce como **herencia**. Cuando se define una clase general, usted puede definir una clase más específica que hereda algunas de las características de la clase general, pero también tiene nuevas características. Por ejemplo, cuando un objeto de tipo `Formas_geométricas` se define, usted puede definir una clase llamada `Rectángulos`. Los rectángulos son formas geométricas con características adicionales. En C++ se permite herencia múltiple. Una clase puede heredar algo de más de una clase.

Polimorfismo Polimorfismo significa "muchas formas". El **polimorfismo** en C significa que usted puede definir varias operaciones con el mismo nombre que pueden realizar cosas distintas en clases afines. Por ejemplo, suponga que usted define dos clases, `Rectángulos` y `Círculos`, ambas heredadas de la clase `Formas_geométricas`. Usted define dos operaciones ambas llamadas *área*, una en `Rectángulos` y otra en `Círculos`, que calculan el área de un rectángulo o un círculo. Las dos operaciones tienen el mismo nombre pero hacen cosas distintas; el cálculo del área de un rectángulo y del área de un círculo necesitan diferentes operandos y operaciones.

Java

Java se desarrolló en Sun Microsystems, Inc. Se basa en C y en C++, pero algunas características de C++, como la herencia múltiple, se quitaron para hacer el lenguaje más robusto. Además el lenguaje está totalmente orientado a clases. En C++ uno puede resolver un problema incluso sin definir una clase. En Java, cada elemento de datos pertenece a una clase.

Un programa en Java puede ya sea ser una aplicación o un applet. Una aplicación es un programa independiente completo que puede ejecutarse de manera independiente (como un programa C++). Un applet, por otro lado, se incrusta en lenguaje HTM, se almacena en un servidor y se ejecuta por un explorador. El explorador puede descargar el applet y ejecutarlo localmente.

En Java, un programa de aplicación (o un applet) es una colección de clases e instancias de esas clases. Una característica interesante de Java es la biblioteca de clases, una colección de clases. Aunque C++ también ofrece una biblioteca de clases, en Java el usuario puede construir nuevas clases con base en aquellas provistas por la biblioteca.

La ejecución de un programa en Java también es única. Usted crea una clase y la pasa al intérprete que llama a los métodos de clase.

Otra característica interesante en Java es la ilación múltiple (*multithreading*). Un hilo (*thread*) es una secuencia de acciones ejecutadas una después de otra. C++ sólo permite una ilación individual (todo el programa se ejecuta como un solo hilo), pero Java permite la ilación múltiple (ejecución concurrente de varias líneas de código).

LENGUAJES FUNCIONALES

En la programación funcional, un programa se considera una función matemática. En este contexto, una **función** es una caja negra que correlaciona una lista de entradas con una lista de salidas (figura 9.5).

Por ejemplo, la *suma* puede considerarse una función con *n* entradas y sólo una salida. La función toma las *n* entradas, las suma y crea el resultado de la suma. Un **lenguaje funcional** realiza lo siguiente:

1. Predefine una serie de funciones primitivas (atómicas) que puede usar cualquier programador.
2. Permite al programador combinar funciones primitivas para crear funciones nuevas.

Por ejemplo, usted puede definir una función primitiva llamada **primero** que extrae el primer elemento de una lista. Puede tener también una función llamada **resto** que extraiga todos los elementos excepto el primero. Un programa puede definir una función que extrae el tercer elemento de una lista al combinar estas dos funciones como se muestra en la figura 9.6.

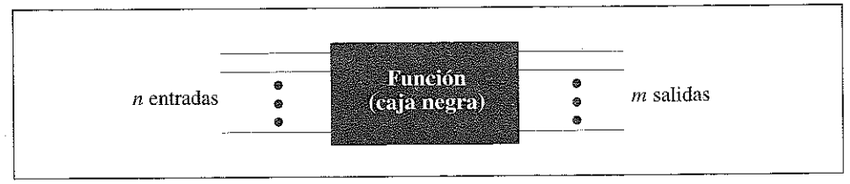


Figura 9.5 Función en un lenguaje funcional

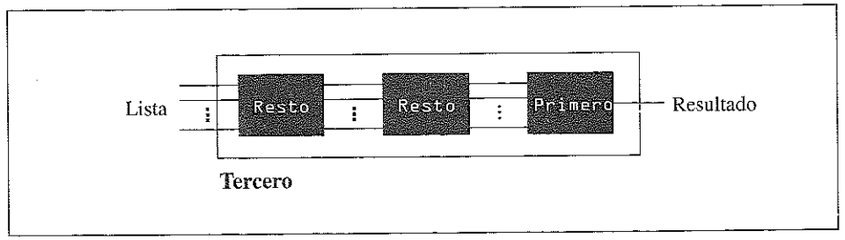


Figura 9.6 Extracción del tercer elemento de una lista

Un lenguaje funcional tiene dos ventajas sobre un lenguaje de procedimiento: fomenta la programación modular y permite al programador hacer nuevas funciones además de las existentes. Estos dos factores ayudan a un programador a crear programas grandes y menos propensos a errores a partir de programas ya probados.

Estudiamos brevemente LISP y Scheme como ejemplos de lenguajes funcionales.

LISP

LISP (*LISt Programming*) fue diseñado por un equipo de investigadores del MIT a principios de la década de 1960. Es un lenguaje de programación de procesamiento de listas en el cual todo se considera una lista.

Scheme

El lenguaje LISP sufrió de una carencia de estandarización. Después de un tiempo, había diferentes versiones de LISP por todas partes; el estándar de facto es el desarrollado por el MIT a principios de la década de 1970, llamado **Scheme**.

El lenguaje Scheme define un conjunto de funciones primitivas que resuelve problemas. El nombre de la función y la lista de entradas a la función se colocan entre paréntesis. El resultado es una lista de salida que puede usarse como la lista de entrada a otra función. Por ejemplo, existe una función, `car`, que extrae el primer elemento de una lista. Hay una función, llamada `cdr`, que extrae el resto de los elementos de una lista excepto el primero. En otras palabras usted tiene:

```
(car  2 3 7 8 11 17 20)   ====>  2
(cdr  2 3 7 8 11 17 20)   ====>  3 7 8 11 17 20
```

Ahora puede combinar estas dos funciones para extraer el tercer elemento de cualquier lista.

```
(car (cdr (cdr list)))
```

Si usted aplica la función anterior a `2 3 7 8 11 17 20`, ésta extrae el 7 debido a que el resultado del paréntesis más interior es `3 7 8 11 17 20`. Esto se convierte en la entrada para el paréntesis medio con el resultado `7 8 11 17 20`. Esta lista ahora se vuelve la entrada para la función `car` la cual extrae el primer elemento, `7`.

LENGUAJES DECLARATIVOS (LÓGICOS)

Un **lenguaje declarativo** utiliza el principio del razonamiento lógico para responder a las consultas. Se basa en la lógica formal definida por los matemáticos griegos y desarrollada posteriormente en lo que se llama *cálculo de predicados de primer orden*.

El razonamiento lógico se basa en la deducción. Se dan algunas instrucciones (hechos) que se supone son verdaderas; la lógica utiliza reglas rigurosas de razonamiento lógico para deducir nuevas instrucciones (hechos).

Por ejemplo, la famosa regla de deducción en lógica es:

```
Si (A es igual a B) y (B es igual a C), entonces (A es
igual a C)
```

Usando esta regla y los dos hechos siguientes,

```
Hecho 1: Sócrates es un ser humano   ⟶  A es igual a B
Hecho 2: Un humano es mortal         ⟶  B es igual a C
```

podemos deducir un nuevo hecho:

```
Hecho 3: Sócrates es mortal          ⟶  A es igual a C
```

Los programadores deben estudiar el dominio de su tema (conocer todos los hechos en el dominio) u obtener los hechos de expertos en el campo. Además los programadores deben ser expertos en lógica para definir cuidadosamente las reglas. Luego el programa puede deducir y crear nuevos hechos.

Un problema asociado con los lenguajes declarativos es que un programa es específico para un dominio particular porque la recopilación de todos los hechos en un programa hace que el programa se vuelva enorme. Ésta es la razón por la cual la programación declarativa está limitada hasta ahora sólo a campos específicos como la inteligencia artificial.

Prolog

Uno de los lenguajes declarativos famosos es **Prolog** (*PROgramming in LOGic*: programación en lógica), desarrollado por A. Colmerauer en Francia en 1972. Un programa en Prolog está formado por hechos y reglas. Por ejemplo, los hechos anteriores sobre los seres humanos pueden exponerse como sigue:

```
humano (John)
mortal (humano)
```

El usuario puede entonces preguntar:

```
? - mortal (John)
```

y el programa le responderá que *sí*.

LENGUAJES ESPECIALES

Durante las últimas décadas, han aparecido algunos lenguajes nuevos que no pueden colocarse en las cuatro categorías que acabamos de analizar. Algunos son una combinación de dos o más modelos y otros pertenecen a una tarea específica. Los clasificamos bajo el rubro de lenguajes especiales.

HTML

El **HTML** (lenguaje para marcado de hipertexto) es un pseudolenguaje que incluye marcas que sirven como sugerencias de formato y vínculos a otros archivos. Un archivo HTML está formado por texto y etiquetas. Una etiqueta aparece entre dos paréntesis angulares y por lo general viene en pares. Un archivo HTML (página) se guarda en el servidor y puede descargarse mediante un explorador. El explorador elimina las etiquetas y las interpreta como sugerencias de formato o vínculos a otros archivos. Un lenguaje de marcado como el HTML le permite incrustar instrucciones de formato en el archivo mismo. Las instrucciones se almacenan con el texto. De esta manera, cualquier explorador puede leer las instrucciones y dar formato al texto de acuerdo con la estación de trabajo que se está usando.

Uno podría preguntar: ¿Por qué no se usan las capacidades de formato de los procesadores de palabras para crear y guardar el texto con formato? La respuesta es que los distintos procesadores de palabras utilizan técnicas o procedimientos diferentes para dar formato al texto. El HTML le permite usar sólo caracteres ASCII tanto para el texto principal como para las instrucciones de formato. De esta manera, cada computadora puede recibir el documento entero como un documento ASCII. El texto principal son los datos y el explorador puede usar las instrucciones de formato para dar forma a los datos. Un programa HTML se compone de dos partes: el encabezado y el cuerpo.

Encabezado (Head) El encabezado es la primera parte que contiene el título de la página y otros parámetros que utilizará el explorador.

Cuerpo (Body) El contenido real de una página está en el cuerpo, el cual incluye el texto y las etiquetas. Mientras que el texto es la información real contenida en una página, las etiquetas definen la apariencia del documento. Cada etiqueta HTML es un nombre seguido por una lista opcional de atributos, encerrados entre paréntesis angulares (<and>).

Etiquetas (Tags) El explorador toma una decisión respecto a la estructura del texto basado en las etiquetas, las cuales son marcas que se incrustan en el texto. Una etiqueta está encerrada entre dos paréntesis angulares y usualmente viene por pares. Una etiqueta puede tener una lista de atributos, cada uno de los cuales puede estar seguido por un signo igual y un valor asociado con el atributo. La tabla 9.1 muestra algunas de las etiquetas más comunes.

Etiqueta inicial	Etiqueta final	Significado
<HTML>	</HTML>	Define un documento HTML
<HEAD>	</HEAD>	Define el encabezado del documento
<BODY>	</BODY>	Define el cuerpo del documento
<TITLE>	</TITLE>	Define el título del documento
<Hi>	</Hi>	Define diferentes encabezados (i es un entero)
<B>	</B>	Tipo negritas
<I>	</I>	Itálicas
<U>	</U>	Subrayado
_		Subíndice
[	]	Superíndice
<CENTER>	</CENTER>	Centrado
 		Salto de línea
<OL>	</OL>	Lista ordenada
<UL>	</UL>	Lista sin ordenar
<LI>	</LI>	Un elemento en una lista
<IMG>		Define una imagen
<A>	</A>	Define una dirección (hipervínculo)

Tabla 9.1 Etiquetas comunes

A continuación se muestra un ejemplo de un programa HTML (página web). Cuando éste se ejecuta, muestra la imagen vinculada al mismo.

```
<HTML>
     <HEAD>
          <TITLE> Documento de muestra </TITLE>
     </HEAD>
     <BODY>
          Ésta es la imagen de un libro:
          <IMG SRC="Imágenes/libro1.gif"    ALIGN=MIDDLE>
     </BODY>
</HTML>
```

Programa 9.4 Programa HTML

PERL

El lenguaje práctico de extracción e informes (**PERL**: *Practical Extraction and Report Language*) es un lenguaje de alto nivel con una sintaxis similar al lenguaje C pero más eficiente. La fuerza de Perl radica en su uso bien diseñado de *expresiones regulares* que permiten al programador hacer un análisis sintáctico de una cadena de caracteres en sus componentes y extraer la información requerida.

SQL

El lenguaje de consultas estructurado (**SQL**: *Structured Query Languague*) es un lenguaje utilizado para responder consultas sobre bases de datos. Veremos algunos ejemplos de SQL en el capítulo 14 cuando estudiamos las bases de datos relacionales.

9.5 UN LENGUAJE PROCEDURAL: C

En esta sección, exploramos rápidamente un lenguaje procedural común que la mayoría de los programadores considera la madre de varios lenguajes modernos como C++, Java y PERL. Analizamos la estructura del lenguaje y describimos sus elementos. Este viaje de ninguna manera es una cobertura completa de la sintaxis del lenguaje; es una mirada a vuelo de pájaro que muestra los diferentes aspectos del lenguaje para estudiantes que estudiarán los lenguajes de programación en el futuro.[1]

IDENTIFICADORES

Una característica presente en todos los lenguajes de computadora es el **identificador.** Los identificadores le permiten nombrar datos y otros objetos en el programa. Cada pieza de datos en una computadora se almacena en una dirección única. Si no hubiera identificadores para representar simbólicamente las localidades de los datos, usted tendría que conocer y usar direcciones de datos para manipularlos. En vez de ello, simplemente proporciona nombres de identificadores de datos y deja que el **compilador** siga la pista de dónde se localizan físicamente.

Los diferentes lenguajes de programación utilizan reglas distintas para formar identificadores. En C las reglas para los identificadores son muy simples. Los únicos símbolos de nombre válidos son las letras mayúsculas de la *A* a la *Z*, las letras minúsculas de la *a* a la *z*, los dígitos de 0 a 9 y el subrayado. El primer carácter del identificador no puede ser un dígito. Según la costumbre, las aplicaciones no utilizan el subrayado para el primer carácter porque muchos de los identificadores en las bibliotecas del sistema que soportan C inician con un subrayado. De este modo, usted se asegura de que sus nombres no duplican los nombres del sistema, lo cual podría volverse muy confuso. La última regla es que el nombre que usted crea no puede ser ninguno de los 50 nombres especiales, conocidos como **palabras reservadas** o **palabras clave**, que están contenidas en el lenguaje mismo.

TIPOS DE DATOS

Un **tipo de dato** define un conjunto de **valores** y una serie de **operaciones** que pueden aplicarse a aquellos valores. El conjunto de valores para cada tipo se conoce como el dominio para el tipo. Por ejemplo, un interruptor de luz puede compararse con un tipo de computadora. Tiene un conjunto de dos valores: encendido y apagado. Dado que el dominio de un interruptor de luz consiste en sólo estos dos valores, su tamaño es dos. Sólo hay dos operaciones que pueden aplicarse a un interruptor de luz: encender y apagar.

C contiene tres **tipos estándar**: `int` (abreviatura para enteros), `char` (abreviatura para caracteres), `float` (abreviatura para punto flotante). Los tipos estándar son atómicos: no pueden dividirse. También sirven como los bloques de construcción básicos para los tipos derivados. Los **tipos derivados** son estructuras complejas que se construyen usando los tipos estándar. Los tipos derivados son: *apuntador, tipo enumerado, unión, arreglo* y *estructura.*

integer

Un tipo `int` (**integer**: entero) es un número sin una parte fraccionaria. También se conoce como un número integral. C soporta tres tamaños distintos del tipo de datos entero: `short int`, `int` y `long int`. Un `short int` también puede referirse como `short` y a `long int` también se le puede llamar `long`. Ya definimos un entero en el capítulo 3; las denominaciones short y long definen el número de bytes asignados para un entero por el compilador.

char

El segundo tipo es `char` (**character**: carácter). Aunque pensamos en los caracteres como si fueran letras del alfabeto, el lenguaje C tiene otra definición. Para C, un carácter es cualquier valor que puede representarse en el alfabeto de la computadora. C utiliza el alfabeto del Código norteamericano de estándares para intercambio de información (ASCII, que se pronuncia as-qui). Los caracteres ASCII se presentan en forma de tabla en el apéndice A.

[1] Para un análisis completo, consulte Behrouz A. Forouzan y Richard G. Gilberg, *An Introduction to Computer Science: A Structured Programming Approach Using C*, 2a. ed. (Pacific Grove, California, Ed. Brooks/Cole, 2001.)

float

Un tipo `float` es un número con una parte fraccionaria, por ejemplo 43.32. Los tipos flotantes se definieron en el capítulo 3. El lenguaje C soporta tres tamaños diferentes de tipos de datos de punto flotante: `float`, `double` y `longdouble`. Al igual que en el caso de `int`, los tipos `float` se definen del menor al mayor.

VARIABLES

Las **variables** son nombres de localidades de memoria. Como se vio en el capítulo 5, cada localidad de memoria (o palabra) en una computadora tiene una dirección. Aunque la computadora utiliza las direcciones internamente es muy inconveniente para el programador usar direcciones debido a que el programador no sabe dónde se almacenan el programa y los datos en la memoria. Los nombres como un sustituto de las direcciones, liberan al programador de pensar en el nivel que se ejecuta el programa. Un programador puede usar una variable, tal como `puntaje` para almacenar el valor entero de una calificación recibida en una prueba (97). La computadora puede usar la localidad de memoria 245,876 para almacenar este valor. Cuando se hace una referencia a `puntaje` durante la ejecución del programa, la computadora sabe que esto significa la localidad 245,876. La figura 9.7 muestra el concepto de una variable.

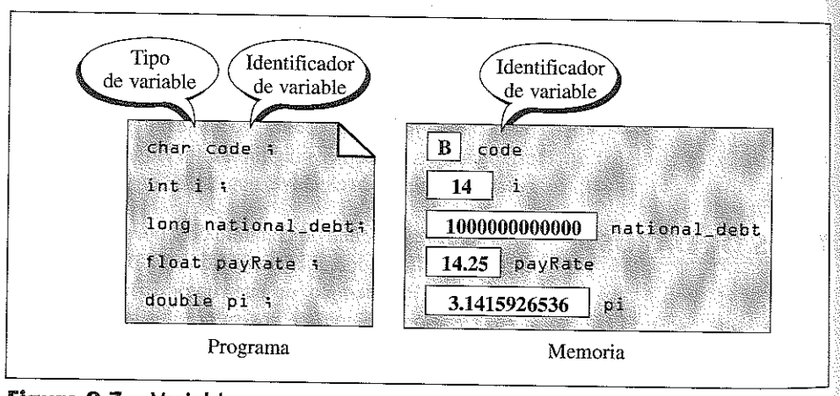


Figura 9.7 Variables

C requiere que el programador defina un tipo para la variable. La razón es que el tipo define el rango de los datos que pueden almacenarse en la variable y define el número de bytes que pueden asignarse a esa variable.

Declaración y definición de variables

C requiere que una variable se declare y defina antes de usarla. Una **declaración** se usa para nombrar una variable. Las **definiciones** se utilizan para crear la variable y asignar una localidad de memoria a ella.

Cuando usted crea una variable la declaración le da un nombre simbólico y la definición reserva memoria para ella. Una vez definidas, las variables se utilizan para alojar los datos que requiere el programa para su operación.

El tipo de variable puede ser cualquiera de los tipos de datos, como `char`, `int` o `float`. Para crear una variable se especifica el tipo y el nombre. Por ejemplo, lo siguiente puede usarse para declarar y definir una variable llamada `precio` en C. El nombre de la variable es `precio` y su tipo es `float` (número de punto flotante).

```
float precio ;
```


Después de esto, la computadora asigna algunas localidades de memoria (el número depende del sistema de cómputo) y las llama en forma colectiva `precio`. Observe que esta variable está definida, pero no hay nada almacenado todavía en ella.

Inicialización de variables

C permite que una variable se inicialice al momento que se declara y define. El **inicializador** establece el primer valor que la variable contendrá. Para inicializar una variable cuando ésta es definida el nombre es seguido por el operador de asignación (el signo igual) y luego por el inicializador. El formato de inicialización simple es:

```
float precio = 23.45 ;
```

CONSTANTES

Las **constantes** son valores de datos que no pueden cambiarse durante la ejecución de un programa. Por ejemplo, tal vez un programador necesite usar el valor de *pi* (3.14) varias veces. Es muy conveniente definir una constante al principio de un programa y usarla. En algunas ocasiones el valor puede cambiar, pero no cada vez que se ejecute el programa. Por ejemplo, el valor de gravamen puede cambiar cada año, pero se fija durante el año. Un programador puede definir una constante para el valor de gravamen y revisar cada año para ver si éste ha cambiado.

Las constantes pueden aparecer en un programa en una de tres formas: constante literal, constante nombrada o constante simbólica.

Constante literal

Una **constante literal** es un valor sin nombrar que se usa en un programa tal cual es. Por ejemplo, a continuación se muestra cómo se utiliza una constante literal en una instrucción en C. El valor 2 es una constante literal. Las variables son longitud, ancho y circunferencia.

```
circunferencia = 2 * longitud * ancho
```

Constante nombrada

Una **constante nombrada** es un valor que se almacena en la memoria; usted no quiere que el programa la cambie. Enseguida se muestra cómo el lenguaje C crea una constante nombrada:

```
const pi = 3.14 ;
```

Constante simbólica

Una **constante simbólica** es un valor que tiene sólo un nombre simbólico. El compilador puede remplazar el nombre con el valor. En la mayoría de los lenguajes de programación, el valor no se almacena en la memoria. La constante simbólica se define al principio del programa para hacerla evidente. Se utiliza para una constante que cambia su valor de manera ocasional. Lo siguiente demuestra cómo el lenguaje C define una constante simbólica:

```
#define valorGravamen 0.0825
```

Constantes y tipos

Al igual que las variables, las constantes tienen un tipo. La mayoría de los lenguajes de programación utilizan constantes enteras, constantes de punto flotante, constantes de carácter y constante de cadena. Por ejemplo, el código siguiente muestra diferentes constantes en el lenguaje C. Las literales de carácter se almacenan en comillas simples; las literales de cadena están entre comillas dobles. Observe que el texto encerrado entre /* y */ es un comentario que ignora el compilador.

```
23          /* Literal de entero          */
23.12       /* Literal de punto flotante  */
'A'         /* Literal de carácter        */
"Hola"      /* Literal de cadena          */
```

ENTRADA Y SALIDA

Casi todos los programas necesitan leer y/o escribir datos. Estas operaciones pueden ser bastante complejas, en especial cuando se leen o escriben archivos grandes. La mayoría de los lenguajes de programación utilizan una función predefinida para entrada y salida.

Entrada

La introducción de los datos se da ya sea por una instrucción o por una función predefinida. El lenguaje C tiene varias funciones de entrada. Por ejemplo, la función **scanf** lee datos desde el teclado y los almacena en una variable. Véase el siguiente ejemplo:

```
scanf ("%d", &num) ;
```

Cuando el programa encuentra esta instrucción, espera a que el usuario teclee un entero. Luego almacena el valor en la variable **num.** La **%d** indica al programa que espere un entero.

Salida

La salida de los datos se da ya sea por una instrucción o por una función predefinida. El lenguaje C tiene varias funciones de salida. Por ejemplo, la función **printf** despliega una cadena en el monitor. El programador puede incluir el valor de una variable o variables como parte de una cadena. Enseguida aparece el valor de una variable al final de una cadena literal.

```
printf ("El valor del número es: %d" , num ) ;
```

EXPRESIONES

Una **expresión** es una secuencia de operandos y operadores que se reduce a un solo valor. Por ejemplo, la siguiente expresión tiene un valor de 10.

```
2 * 5
```

Operador

Un **operador** es un elemento sintáctico de un lenguaje específico que requiere que se realice una acción. Los operadores más conocidos se tomaron de las matemáticas. Por ejemplo, multiplicar (*) es un operador. Indica que dos números se van a multiplicar. Todos los lenguajes tienen operadores y su uso se especifica rigurosamente en la sintaxis, o reglas, del lenguaje.

Operadores aritméticos El lenguaje C define varios **operadores aritméticos**, algunos de los cuales se muestran en la tabla 9.2.

Operador	Definición	Ejemplo
+	Suma	3 + 5
-	Resta	2 - 4
*	Multiplicación	Num * 5
/	División (cociente)	Sum/ Count
%	División (residuo)	Count % 4
++	Incremento (suma 1 al valor de la variable)	Count++
--	Decremento (resta 1 al valor de la variable)	Count--

Tabla 9.2 Operadores aritméticos

Operadores relacionales Los **operadores relacionales** comparan los datos para ver si un valor es mayor que, menor que o igual que otro valor. El resultado de aplicar operadores relacionales es los valores lógicos verdadero (*true*) o falso (*false*). El lenguaje C utiliza seis operadores relacionales (tabla 9.3).

Operador	Definición	Ejemplo
<	Menor que	Num1 < 5
<=	Menor o igual que	Num1 <= 5
>	Mayor que	Num2 > 3
>=	Mayor o igual que	Num2 >= 3
==	Igual que	Num1 == Num2
!=	No igual que	Num1 != Num2

Tabla 9.3 Operadores relacionales

Operadores lógicos Los **operadores lógicos** combinan valores lógicos (verdadero o falso) para obtener un nuevo valor. El lenguaje C utiliza tres operadores lógicos como se muestra en la tabla 9.4.

Operador	Definición	Ejemplo
!	NO	! (Num1 < Num2)
&&	Y	(Num1 < 5) && (Num2 > 10)
||	O	(Num1 < 5) || (Num2 > 10)

Tabla 9.4 Operadores lógicos

Operadores de asignación Un **operador de asignación** almacena un valor en una variable. El lenguaje C define varios operadores de asignación, algunos de los cuales se muestran en la tabla 9.5.

Operador	Ejemplo	Significado
=	num = 5 ;	Almacena 5 en la variable num
+=	num += 5 ;	Lo mismo que num = num + 5
-=	num -= 5 ;	Lo mismo que num = num - 5
*=	num *= 5 ;	Lo mismo que num = num * 5
/=	num /= 5 ;	Lo mismo que num = num / 5
%=	num %= 5 ;	Lo mismo que num = num % 5

Tabla 9.5 Operadores de asignación

Operando

Un **operando** recibe la acción de un operador. Para cada operador dado puede haber uno, dos o más operandos. En nuestro ejemplo aritmético, los operandos de multiplicar son el multiplicador y el multiplicando.

INSTRUCCIONES

Una **instrucción** provoca que el programa realice una acción. Ésta se traduce directamente en una o más instrucciones de computadora ejecutables. Por ejemplo, el lenguaje C define seis tipos de instrucciones: instrucción de expresión, instrucción compuesta, instrucción etiquetada, instrucción de selección, instrucción iterativa e instrucción de salto (figura 9.8). En esta sección se estudian algunas de ellas.

Instrucciones de expresión

Una expresión se convierte en instrucción al colocar un punto y coma (;) después de ella. Cuando C ve el punto y coma evalúa la expresión. Si hay una asignación involucrada (existencia de un operador de asignación, un ++ o --), el valor se almacena en la variable. Si no hay asignación el valor se descarta. A continuación se presentan algunos ejemplos de **instrucciones de expresión**:

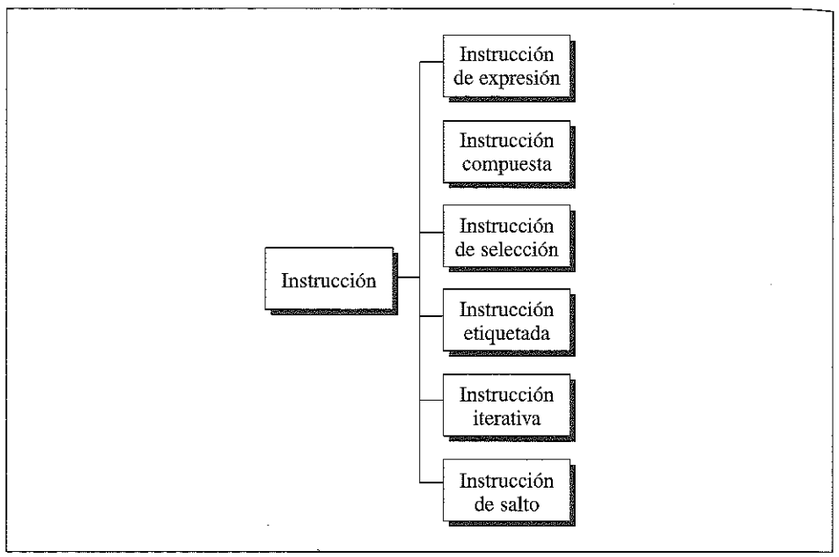


Figura 9.8 Instrucciones

```
a++ ;
b = 4 ;
c = b + c * 4 ;
```

Instrucciones compuestas

Una **instrucción compuesta** es una unidad de código que consiste en cero o más instrucciones. También se conoce como **bloque**. La instrucción compuesta permite a un grupo de instrucciones volverse una sola entidad. Una instrucción compuesta se compone de una llave de apertura y una sección de instrucciones opcional, seguida por un corchete de cierre. Enseguida se muestra la composición de una instrucción compuesta.

```
{
x = 1;
y = 20;
}
```

FUNCIONES

EN C, una subrutina se conoce como **función**. Un programa en C está formado por una o más funciones, una y sólo una de las cuales debe llamarse **main** (principal). La ejecución del programa siempre inicia y termina con **main**, pero esta función puede llamar a otras funciones para que realicen tareas especiales.

Una función en C (incluyendo a **main**) es un módulo independiente que se llama para que realice una tarea específica. La función **main** es llamada por el sistema operativo; **main** a su vez llama a otras funciones. Cuando **main** se ha completado el control regresa al sistema operativo.

En general, el propósito de una función es recibir cero o más piezas de datos, realizar operaciones en ellos y devolver a lo más una pieza de datos. Al mismo tiempo, una función puede tener un **efecto secundario**, el cual es una acción que resulta en un cambio de estado del programa. Si hay un efecto secundario, éste ocurre cuando la función se está ejecutando y antes de que regrese la función. El efecto secundario puede implicar la aceptación de datos desde fuera del programa, el envío de datos fuera del programa hacia el monitor o a un archivo, o el cambio del valor de una variable en la función que llama. El concepto de función aparece en la figura 9.9.

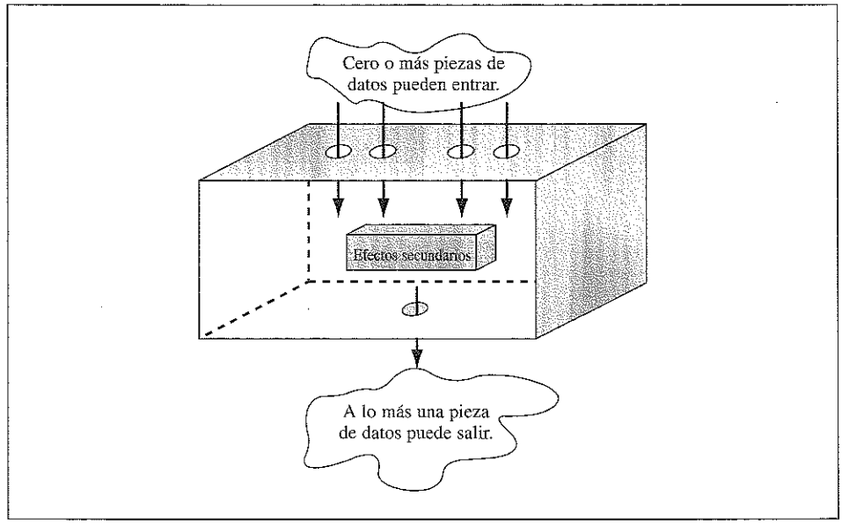


Figura 9.9 Efecto secundario de una función

Declaración de función

Las funciones en C deben tanto declararse como definirse. La **declaración de funciones** se realiza con una declaración prototipo. Usted utiliza la función al llamarla. La definición de función contiene el código requerido para completar la tarea. La figura 9.10 muestra las interrelaciones entre estos componentes de función. Observe que el nombre de función se utiliza tres veces: cuando se declara la función, cuando se llama y cuando se define.

Definición de función

La **definición de función** contiene el código para una función. La definición se compone de dos partes: el encabezado de función y el cuerpo de función, el cual es una instrucción compuesta.

Encabezado de función Un **encabezado de función** consiste en tres partes: el tipo de regreso, el nombre de función y la lista de **parámetros formales**.

Cuerpo de función El **cuerpo de función** contiene las declaraciones e instrucciones para la función. El cuerpo inicia con definiciones locales que especifican las variables requeridas por la función. Después de las declaraciones locales, se codifican las instrucciones de función, terminando con una instrucción `return`. Si un tipo de retorno de función es `void` puede escribirse sin una instrucción `return`.

Llamada de función

Una **llamada de función** se utiliza para llamar a la función. La llamada contiene los **parámetros reales**, los cuales identifican los valores que se van a enviar a la función llamada. Éstos corresponden a los parámetros formales de la función en tipo y orden en la lista de parámetros. Si hay varios parámetros actuales, éstos se separan por comas.

Paso de parámetros

En C, se puede pasar un parámetro a una función de dos maneras: paso por valor y paso por referencia.

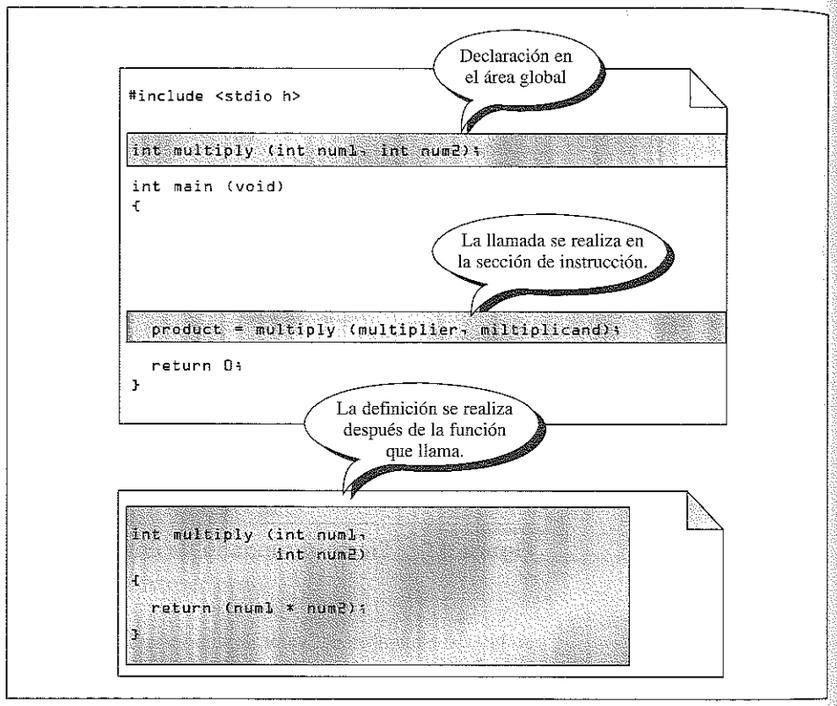


Figura 9.10 Declaración de función

Paso por valor Cuando se **pasa por valor** se crea una copia de los datos y se coloca en una variable local en la función llamada. Esta forma de pasar los datos asegura que a pesar de cómo se manipulen los datos y cambien en la función llamada, los datos originales en la función que llama están seguros y permanecen sin cambios. Debido a que el paso por valor protege a los datos, es la técnica de paso preferida.

Paso por referencia Sin embargo, hay veces que es necesario pasar por referencia. El **paso por referencia** envía la dirección de una variable a la función llamada en lugar de enviar su valor. Cuando usted quiere cambiar el contenido en una variable en la función que llama, debe pasar por referencia.

Considere el caso en el cual necesita escribir una función que procese dos valores de datos y los devuelva a la función que llama; es decir, almacena sus valores en el programa que llama. Dado que una función puede devolver sólo un valor, usted tiene un problema. La solución es el paso por referencia.

SELECCIÓN

Como se vio en el capítulo 8, para resolver un problema usando un algoritmo se necesitan instrucciones de selección.

`if-else`

Para implementar una selección de dos sentidos en C, se puede usar la **instrucción `if-else`**. La figura 9.11 muestra el flujo lógico para `if-else`. La expresión puede ser cualquier expresión C. Después de que ésta se ha evaluado, si su valor es verdadero (no 0) se ejecuta `instruccion1`; de lo contrario, se ejecuta la `instruccion2`. Es imposible que

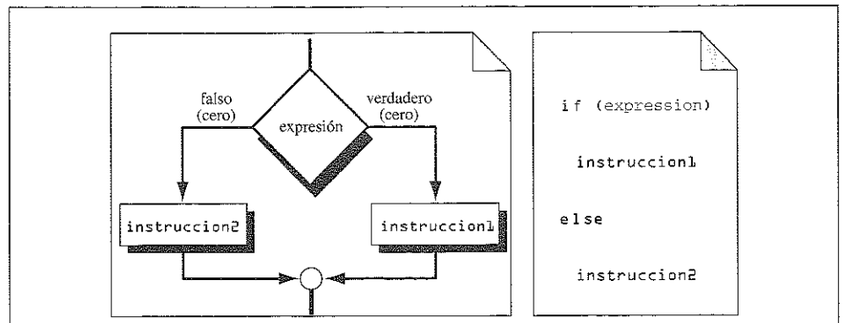


Figura 9.11 Instrucción `if-else`

```
switch  (expresión)
 {
 case constante-1 : instruccion
                    instruccion
 case constante-2 : instruccion
                    instruccion
 case constante-n : instruccion
                    instruccion
 default          : instruccion
                    instruccion
}
```

Figura 9.12 Instrucción `switch`

ambas instrucciones se ejecuten en la misma evaluación. Una instrucción `if-else` puede anidarse (una dentro de la otra) para crear una selección de múltiples sentidos.

`switch`

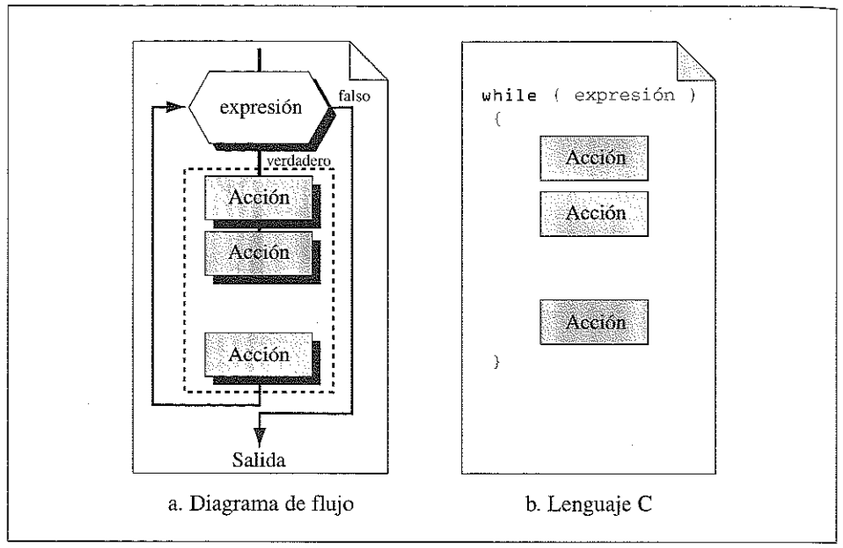
Aun cuando se puedan utilizar instrucciones `if-else` anidadas para la selección de múltiples sentidos, hay otra instrucción en C para este propósito: la **instrucción switch** (figura 9.12). La condición de selección debe ser uno de los tipos integrales de C.

REPETICIÓN

En el capítulo 8 se vio que se necesitan instrucciones de repetición. El lenguaje C define tres tipos de instrucciones de ciclo: `while`, `for` y `do-while`.

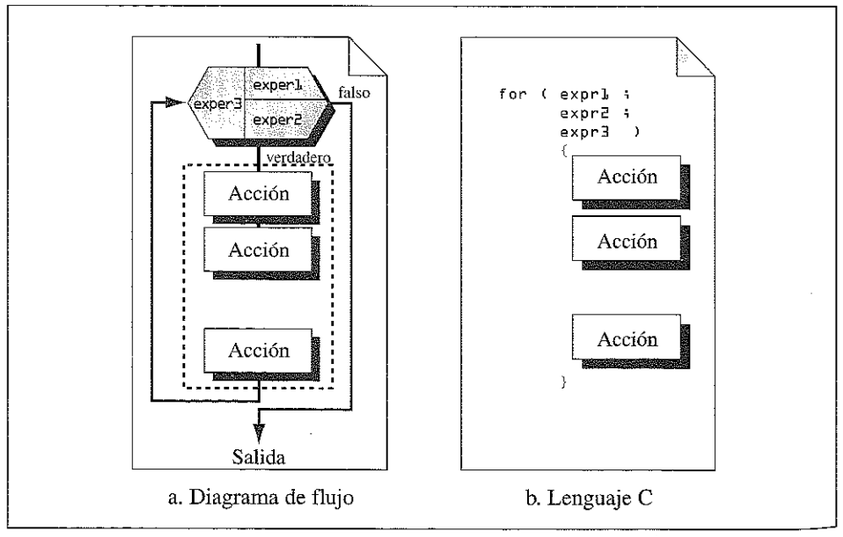
Ciclo `while`

La instrucción de repetición principal en el lenguaje C es el ciclo while (figura 9.13). Un **ciclo `while`** es un ciclo preprobado. Revisa el valor de una expresión de prueba. Si el valor es



a. Diagrama de flujo b. Lenguaje C

Figura 9.13 Ciclo `while`



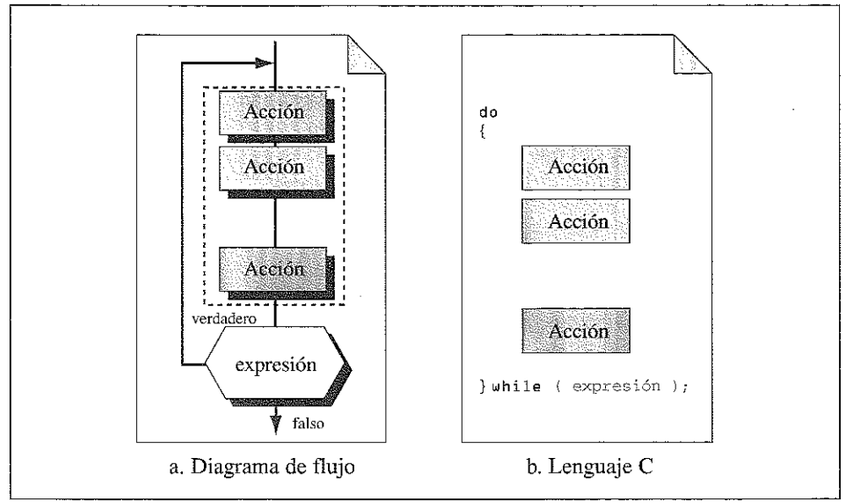
a. Diagrama de flujo b. Lenguaje C

Figura 9.14 Ciclo `for`

verdadero (no es cero), pasa por una iteración del ciclo y prueba de nuevo el valor. El ciclo `while` se considera un ciclo controlado por eventos. El ciclo continúa en iteración hasta que ocurre un evento que cambia el valor de la expresión de prueba de verdadero a falso.

Ciclo `for`

El **ciclo `for`** también es un ciclo de preprueba (figura 9.14). Sin embargo, a diferencia del ciclo `while`, es un ciclo controlado por el contador. Un contador se fija en un valor inicial y aumenta (o disminuye) en cada iteración. El ciclo se termina cuando el valor del contador coincide con un valor predeterminado.



a. Diagrama de flujo b. Lenguaje C

Figura 9.15 Ciclo `do-while`

Ciclo `do-while`

El **ciclo `do-while`** es también un ciclo controlado por eventos; no obstante, a diferencia del ciclo `while` es un ciclo post prueba (figura 9.15). El ciclo realiza una iteración y prueba el valor de una expresión. Si es falso, termina. Si es verdadero, realiza una iteración más y prueba de nuevo.

TIPOS DE DATOS DERIVADOS

Además de los datos simples estudiados anteriormente, el lenguaje C también permite tipos de datos derivados: arreglos, apuntadores, uniones y estructuras. Estudiaremos algunos de estos tipos de datos en el capítulo 11 sobre la estructura de datos.

RECURSIÓN

El lenguaje C soporta la **recursividad** (analizada en el capítulo 8). Una función en C puede llamarse a sí misma.

9.6 TÉRMINOS CLAVE

Ada
archivo ejecutable
archivo fuente
bloque
cargador
`char`
ciclo `do-while`
ciclo `for`
ciclo `while`
clase
COBOL
compilador
constante
constante literal
constante nombrada
constante simbólica
cuerpo de función
declaración
declaración de función
definición
definición de función
directiva de preprocesador
editor de textos
efecto secundario
encapsulamiento
encabezado de función
ensamblador
entero
expresión
`float`
FORTRAN
función
herencia
HTML
identificador
inicializador
instrucción
instrucción compuesta
instrucción de expresión
instrucción `if-else`
instrucción `switch`
Java
lenguaje C
lenguaje C++
lenguaje declarativo

lenguaje de alto nivel
lenguaje de computadora
lenguaje de máquina
lenguaje de programación
lenguaje ensamblador
lenguaje funcional
lenguaje imperativo
lenguaje natural
lenguaje procedural
lenguaje orientado a objetos
lenguaje simbólico
ligador
LISP
llamada a función
módulo de objetos
operación
operador
operador aritmético
operador de asignación
operador lógico
operando
operador relacional
palabra clave
palabra reservada
parámetro formal
parámetros reales
Pascal
paso por referencia
paso por valor
PERL
polimorfismo
preprocesador
Prolog
recursividad
Scheme
SQL
traductor
tipo de datos
tipos derivados
tipos estándar
sintaxis
subprograma
unidad de traducción
variable

9.7 RESUMEN

- El único lenguaje comprendido por una computadora es el lenguaje de máquina.
- Un lenguaje simbólico utiliza símbolos para representar varias instrucciones de lenguaje de máquina. Los lenguajes simbólicos también se conocen como lenguajes ensambladores.
- Un lenguaje de alto nivel se puede transportar de un tipo de computadora a otro, y libera al programador de preocupaciones relacionadas con el hardware. BASIC, Pascal, Ada, C, C++ y Java son lenguajes de alto nivel.
- Los pasos para construir un programa incluyen escritura, edición, compilación y ligado de código.
- Hay cinco categorías de lenguajes de computadora: procedurales, orientado a objetos, funcional, declarativo y especial.
- En un lenguaje procedural, un algoritmo se traduce en código. El código manipula los datos y controla la ejecución de instrucciones. FORTRAN, COBOL, Pascal, C y Ada son lenguajes procedurales.
- En un lenguaje orientado a objetos como C++ y Java, los objetos y las operaciones aplicadas a los objetos están ligados.
- C++ utiliza los conceptos de encapsulamiento, herencia y polimorfismo.
- En un lenguaje funcional, el algoritmo es de naturaleza matemática. LISP y Scheme son lenguajes funcionales.
- Un lenguaje declarativo utiliza los principios del razonamiento lógico. Prolog es un lenguaje declarativo.
- Un lenguaje especial no puede clasificarse en uno de los otros cuatro grupos. HTML, PERL y SQL son lenguajes especiales.
- C se considera la madre de lenguajes como C++, Java y PERL.
- Un identificador nombra a datos y otros objetos en un programa.
- Un tipo de datos define un conjunto de valores y una serie de operaciones que pueden aplicarse a aquellos valores.
- Los tipos de datos estándar para el lenguaje C son `int` (entero), `char` (carácter) y `float` (punto flotante).
- Una constante es un valor de datos que no puede cambiarse durante la ejecución de un programa.
- Una variable es un nombre para una localidad de memoria.
- Por lo general los datos se introducen y se da salida a los mismos mediante funciones predefinidas.
- Una expresión es una secuencia de operandos y operadores que se reduce a un solo valor.
- Una instrucción provoca que un programa realice una acción.
- Una función es un módulo independiente que realiza una tarea específica.
- Un programa en C tiene una función llamada `main` y cero o más funciones distintas.
- Una instrucción `if-else` o una instrucción `switch` puede implementar una decisión en C.
- La repetición en C puede manejarse con un ciclo `while`, un ciclo `for` o un ciclo `do-while`.

9.8 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿En qué se diferencia un lenguaje simbólico de un lenguaje de máquina?
2. ¿Por qué un lenguaje simbólico también se conoce como lenguaje ensamblador?
3. ¿Cuáles son las ventajas de un lenguaje de alto nivel sobre un lenguaje de máquina?
4. Mencione algunos de los lenguajes de alto nivel.
5. ¿Qué es un lenguaje natural?
6. ¿Qué es un archivo fuente? ¿Qué es un archivo ejecutable? ¿Cómo se relacionan los dos?
7. ¿Cuál es la función de un compilador?
8. ¿Cuál es la función de un ligador?
9. ¿Cuáles son las cinco categorías de los lenguajes de computadora? Dé un ejemplo de cada una.
10. ¿Cómo se relaciona Java con C y C++?
11. ¿Por qué C es un lenguaje popular entre los programadores?
12. ¿Cuáles son algunos de los grupos que usan Ada?
13. ¿En qué difiere la programación procedural de la programación orientada a objetos?
14. ¿Cuáles son tres conceptos intrínsecos a los programas en C++?
15. ¿Qué es un lenguaje funcional? Dé un ejemplo de una función.
16. ¿Qué es un lenguaje declarativo?
17. ¿Qué es un lenguaje especial?
18. Mencione los tres tipos de datos estándar en el lenguaje C.
19. ¿Cuál es el propósito de un inicializador en el lenguaje C?
20. ¿Cuáles son las diferencias entre una constante literal, una constante nombrada y una constante simbólica?
21. ¿Qué es una variable?
22. ¿Cuál es una manera eficiente de manejar funciones de uso frecuente como leer la introducción de datos de un archivo?
23. ¿Cuáles son los operadores relacionales en el lenguaje C?
24. ¿Cuáles son los operadores lógicos en lenguaje C?
25. ¿Cuáles son los operadores de asignación en el lenguaje C?
26. Describa la función `main` en el lenguaje C.
27. Mencione las partes de la definición de una función en C y la función de cada una de ellas.
28. ¿Cuál es el propósito de una instrucción de decisión (selección)?
29. ¿Cuál es la diferencia entre una instrucción `if-else` y una instrucción `switch`? ¿Es un subconjunto de la otra? Explique por qué sí o por qué no.
30. Comente las tres instrucciones de ciclo.

PREGUNTAS DE OPCIÓN MÚLTIPLE

31. El único lenguaje comprendido por el hardware de computadora es el lenguaje _____.
    a. de máquina
    b. simbólico
    c. de alto nivel
    d. natural
32. Los lenguajes _____ también se conocen como lenguajes ensambladores.
    a. de máquina
    b. simbólico
    c. de alto nivel
    d. natural
33. El noruego, el farsi y el ruso pueden clasificarse como idiomas (lenguajes) _____.
    a. de máquina
    b. simbólico
    c. de alto nivel
    d. natural
34. C, C++ y Java pueden clasificarse como lenguajes _____.
    a. de máquina
    b. simbólico
    c. de alto nivel
    d. natural
35. El software utilizado para escribir un programa se llama _____.
    a. preprocesador
    b. editor de texto
    c. traductor
    d. archivo fuente
36. El _____ ensambla unidades precompiladas de fuentes distintas en un programa ejecutable.
    a. preprocesador
    b. editor de texto
    c. ligador
    d. cargador

37. El compilador consiste de un _____ y un _____.
- **a.** preprocesador; cargador
- **b.** editor de texto; cargador
- **c.** preprocesador; traductor
- **d.** ligador; preprocesador

38. _____ es el código en lenguaje de máquina.
- **a.** Una unidad de traducción
- **b.** Un módulo objeto
- **c.** Un archivo fuente
- **d.** Un subprograma

39. Un programa de sistema operativo llamado _____ lleva el programa ligador en la memoria.
- **a.** cagardor
- **b.** vinculador
- **c.** traductor
- **d.** procesador

40. Un lenguaje _____ utiliza el método tradicional para programación y también se conoce como lenguaje imperativo.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

41. FORTRAN es un lenguaje _____.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

42. Pascal es un lenguaje _____.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

43. C++ es un lenguaje _____.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

44. LISP es un lenguaje _____.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

45. _____ es un lenguaje común en el mundo de los negocios.
- **a.** FORTRAN
- **b.** C++
- **c.** C
- **d.** COBOL

46. _____, el primer lenguaje de alto nivel, aún es popular en las comunidades científicas y de ingeniería.
- **a.** FORTRAN
- **b.** C++
- **c.** C
- **d.** COBOL

47. _____ es un programa diseñado para enseñar programación a los novatos que enfatiza el método de programación estructurado.
- **a.** C++
- **b.** C
- **c.** Pascal
- **d.** Scheme

48. El sistema operativo UNIX está escrito en un lenguaje llamado _____.
- **a.** C++
- **b.** C
- **c.** Pascal
- **d.** LISP

49. Un lenguaje procedural muy popular en el DoD es _____.
- **a.** Ada
- **b.** Java
- **c.** C++
- **d.** Scheme

50. Un lenguaje popular orientado a objetos es _____.
- **a.** FORTRAN
- **b.** COBOL
- **c.** C++
- **d.** LISP

51. En C++, la(el)_____ es ocultar los datos y las operaciones al usuario.
- **a.** encapsulamiento
- **b.** herencia
- **c.** polimorfismo
- **d.** modularidad

52. Un programa _____ puede ser una aplicación o un applet.
- **a.** FORTRAN
- **b.** C++
- **c.** C
- **d.** Java

53. LISP y Scheme son lenguajes _____.
- **a.** procedurales
- **b.** funcionales
- **c.** declarativos
- **d.** orientados a objetos

54. Un ejemplo de un lenguaje _____ es Prolog.
- **a.** procedural
- **b.** funcional
- **c.** declarativo
- **d.** orientado a objetos

55. HTML, PERL y SQL caen dentro de la clasificación de lenguajes _____.
- **a.** modernos
- **b.** especiales
- **c.** declarativos
- **d.** orientados a objetos

56. Un tipo de datos estándar en el lenguaje C _____.
- **a.** `int`
- **b.** `char`
- **c.** `float`
- **d.** todos los anteriores

57. El tipo de datos estándar _____ describe un número con una parte fraccionaria.
- **a.** `int`
- **b.** `char`
- **c.** `float`
- **d.** todos los anteriores

58. El tipo de datos estándar _____ describe un número sin una parte fraccionaria.
- **a.** `int`
- **b.** `char`
- **c.** `float`
- **d.** todos los anteriores

59. El tipo de datos estándar _____ describe cualquier valor que pueda representarse en el alfabeto de la computadora.
- **a.** `int`
- **b.** `char`
- **c.** `float`
- **d.** todos los anteriores

EJERCICIOS

60. Investigue y encuentre los operadores aritméticos en FORTRAN, COBOL y Pascal. Compárelos y contrástelos con los operadores aritméticos definidos para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

61. Investigue y encuentre los operadores relacionales en FORTRAN, COBOL y Pascal. Compárelos y contrástelos con los operadores aritméticos definidos para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

62. Investigue y encuentre el formato de la instrucción compuesta en FORTRAN, COBOL y Pascal. Compárelo y contrástelo con el formato definido para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

63. Investigue y encuentre el formato de las instrucciones iterativas (ciclo `for`, ciclo `while`, ciclo `do-while`, etc.) en FORTRAN, COBOL y Pascal. Compárelo y contrástelo con el formato definido para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

64. Investigue y encuentre el formato de las instrucciones de decisión (`if` y `switch`) en FORTRAN, COBOL y Pascal. Compárelo y contrástelo con el formato definido para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

65. Investigue y encuentre la instrucción de salto (por ejemplo, `goto` y `continue`) en FORTRAN, COBOL y Pascal. Compárela y contrástela con el formato definido para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

66. Investigue y encuentre cómo se manejan las funciones en FORTRAN, COBOL y Pascal. Compárelas y contrástelas con las funciones definidas para el lenguaje C en este capítulo. Utilice una tabla para la comparación.

67. Investigue y encuentre si la recursividad está soportada en FORTRAN, COBOL y Pascal.

68. Identifique cada uno de los siguientes identificadores en C como válidos o no válidos. Justifique su respuesta.
- **a.** 2AB
- **b.** A2B
- **c.** AC
- **d.** 999

69. Declare y defina las siguientes variables en el lenguaje C:
- **a.** Una variable `int` llamada i.
- **b.** Una variable `float` llamada f.
- **c.** Una variable `char` llamada c.

70. Repita el ejercicio 69 pero inicialice cada variable con un valor apropiado.

71. En el lenguaje C, muestre cómo leer dos enteros desde el teclado usando sólo una instrucción.

72. En el lenguaje C, muestre cómo imprimir el valor de dos variables de números enteros usando sólo una instrucción.

73. Escriba las siguientes instrucciones en lenguaje C:

- **a.** Multiplique el valor de la variable *x* por el valor de la variable *y* y almacene el resultado en la variable *z*.
- **b.** Aumente el valor de la variable *x*.
- **c.** Disminuya el valor de la variable *y*.
- **d.** Compare el valor de las variables *x* y *y* para probar una igualdad.

74. Escriba un ciclo `for` en C para imprimir el mensaje "¡Hola mundo!" diez veces.

75. Repita el ejercicio 74 utilizando un ciclo `while`.

76. Repita el ejercicio 74 utilizando un ciclo `do-while`.

77. Escriba una instrucción en Scheme para hacer una nueva lista aparte de la lista existente que contenga sólo el tercer y el cuarto elementos.

Ingeniería de software

En este capítulo presentamos el concepto de ingeniería de software. Comenzamos con la idea del ciclo de vida del software. Luego mostramos cómo se usan dos modelos para el proceso de desarrollo: el modelo de cascada y el modelo incremental. Finalmente, analizamos aspectos relacionados con la ingeniería de software, tales como la modularidad, la calidad y la documentación.

La ingeniería de software es el establecimiento y uso de métodos y principios de ingeniería sólidos para obtener software confiable que trabaje en máquinas reales. Esta definición, tomada de la primera conferencia internacional sobre ingeniería de software en 1969, se propuso 30 años después de que se construyó la primera computadora. Durante ese periodo el software fue más un arte que una ciencia. De hecho uno de los tratamientos más serios de la programación la describe como un arte: *The Art of Computer Programming*. Esta serie de tres volúmenes, originalmente escrita por Donald E. Knuth a finales de la década de 1960 y principios de la década de 1970, se considera el análisis más completo de muchos conceptos de las ciencias de la computación.

10.1 CICLO DE VIDA DEL SOFTWARE

Un concepto fundamental en la **ingeniería de software** es el **ciclo de vida del software**. El software, al igual que muchos otros productos pasa por un ciclo de fases repetitivas (figura 10.1).

El software primero se desarrolla por un grupo de desarrolladores/programadores. Por lo general, está en uso durante algún tiempo antes de que se requiera hacerle modificaciones. Las modificaciones a menudo son necesarias debido a errores encontrados en el software, a cambios en las normas o leyes, o a cambios en la compañía misma. El software debe modificarse antes para su uso posterior. Estos dos pasos, usar y modificar, continúan hasta que el software se vuelve obsoleto. Por "obsoleto" queremos decir que el software pierde su validez debido a su ineficiencia, la obsolescencia del lenguaje, cambios importantes en la compañía u otros factores. Algunos ejemplos de desarrollos de software que normalmente pasan por este ciclo son los sistemas de registro de estudiantes, los sistemas de facturación y los sistemas de contabilidad.

El proceso de desarrollo en el ciclo de vida del software implica cuatro fases: análisis, diseño, implementación y pruebas. La figura 10.2 muestra estas fases como parte del proceso de desarrollo.

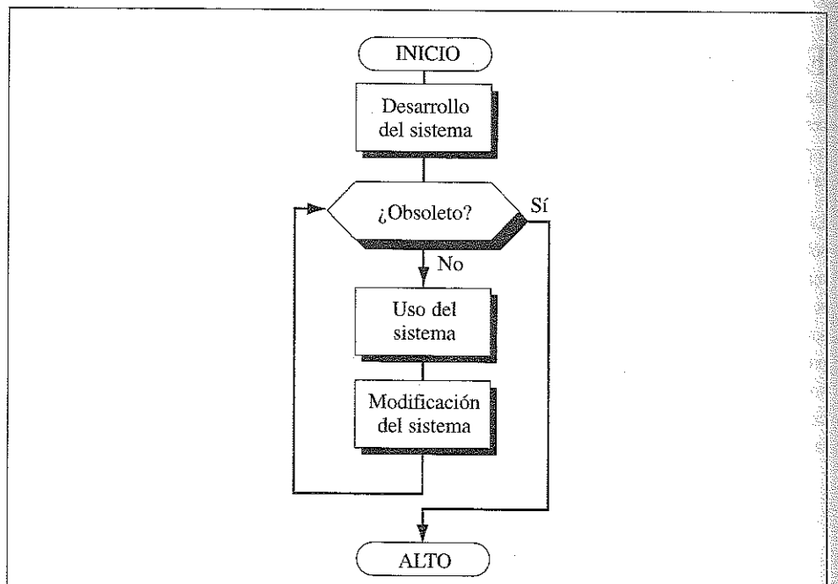


Figura 10.1 Ciclo de vida del sistema

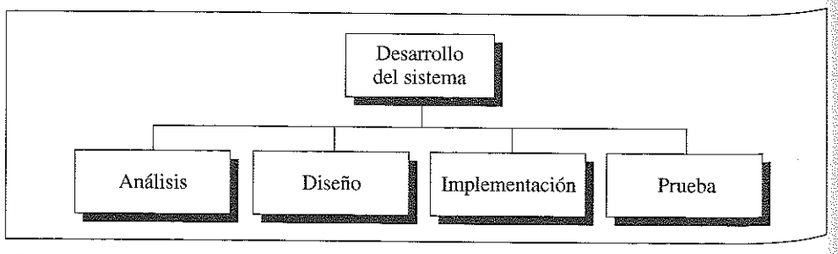


Figura 10.2 Fases de desarrollo del sistema

FASE DE ANÁLISIS

El proceso de desarrollo comienza con la **fase de análisis**, la cual muestra qué debe hacer el paquete. En esta fase, el analista de sistemas define los requisitos que especifican lo que el sistema propuesto va a lograr. Los requisitos por lo general se establecen en los términos que el usuario comprende. Hay cuatro pasos en la fase de análisis: definición del usuario, definición de las necesidades, definición de los requisitos y definición de los métodos.

Definición del usuario

Un paquete de software puede diseñarse para un usuario genérico o para un usuario específico. Por ejemplo, un paquete de contabilidad puede crearse para que lo utilice cualquier empresa. Por otra parte, un paquete bancario personalizado puede ser creado para un banco específico. El usuario del paquete debe definirse con claridad.

Definición de las necesidades

Después de que se ha identificado al usuario, los analistas definen claramente las necesidades. En este paso, la mejor respuesta viene del usuario. El usuario, o el representante del mismo, define con claridad sus expectativas del paquete.

Definición de los requisitos

Con base en las necesidades del usuario, el analista puede definir con precisión los requisitos para el sistema. Por ejemplo, si un paquete va a imprimir cheques al final del mes para cada empleado, ¿qué nivel de seguridad y precisión debe emplearse?

Definición de los métodos

Finalmente, después de que se definen los requisitos en términos claros, el analista puede elegir los métodos apropiados para cumplir estos requisitos.

FASE DE DISEÑO

La **fase de diseño** define *cómo* logrará el sistema *lo que* se definió en la fase de análisis. En la fase de diseño, se determinan los sistemas y el diseño de archivos y/o bases de datos se completa.

Modularidad

Actualmente la fase de diseño utiliza un principio bien establecido llamado **modularidad**. Todo el paquete se divide en pequeños **módulos**. Cada módulo se diseña, se prueba y se vincula a otros módulos a través de un programa principal. La modularidad se estudia posteriormente en este capítulo.

Herramientas

La fase de diseño utiliza varias herramientas, siendo la más común un diagrama de estructura (analizada en el capítulo 8). Un diagrama de estructura muestra cómo dividir un paquete en pasos lógicos; cada paso es un módulo independiente. El diagrama de estructura también muestra la interacción entre todas las partes (módulos).

FASE DE IMPLEMENTACIÓN

En la **fase de implementación** se crean los programas reales.

Herramientas

Esta fase utiliza varias herramientas para mostrar el flujo lógico de los programas antes de la escritura real del código. Una herramienta, aún generalizada, es el diagrama de flujo (analizado en el capítulo 8). Un **diagrama de flujo** utiliza símbolos gráficos estándar para representar el flujo lógico de datos a través de un módulo.

La segunda herramienta utilizada por los programadores es el pseudocódigo (también visto en el capítulo 8). El **pseudocódigo** es parte de idioma (español, inglés, etc.) y parte lógica de programa que describe, con detalles algorítmicos precisos, que indican qué hace el programa. Esto requiere definir los pasos con tanto detalle que la conversión a un programa de computadora pueda lograrse fácilmente.

Codificación

Después de la producción en un diagrama de flujo, pseudocódigo, o ambos, el programador en realidad escribe el código en un lenguaje específico para el proyecto. La opción del lenguaje se basa en la eficiencia del lenguaje para esa aplicación en particular.

FASE DE PRUEBA

Una vez que los programas se han escrito, deben probarse. La **fase de prueba** puede ser muy tediosa y consumir parte del tiempo del desarrollo del programa. Los programadores son completamente responsables de probar sus programas. En los proyectos de desarrollo grandes, con frecuencia hay especialistas llamados ingenieros de pruebas quienes son responsables de probar el sistema como un todo, es decir, de hacer pruebas para comprobar que todos los programas trabajan en conjunto.

Hay dos tipos de pruebas: caja negra y caja blanca. La prueba de caja negra es realizada por el ingeniero de pruebas del sistema y por el usuario. Las pruebas de caja blanca son responsabilidad del programador.

Pruebas de caja negra

Las **pruebas de caja negra** reciben su nombre del concepto de probar un programa sin saber qué hay dentro y sin saber cómo funciona. En otras palabras, el programa es como una caja negra en la que usted no puede ver.

Dicho en forma simple, los planes de pruebas de caja negra se desarrollan considerando sólo los requisitos. Por esta razón es muy importante tener un buen conjunto de requisitos definidos. El ingeniero de pruebas utiliza estos requisitos, su conocimiento en el desarrollo de sistemas y el entorno de trabajo del usuario para crear un plan de pruebas. Este plan se utilizará después cuando el sistema se pruebe como un todo. Usted debe pedir que le muestren estos planes de pruebas antes de escribir su programa.

Pruebas de caja blanca

Mientras que las pruebas de caja negra suponen que no se sabe nada acerca del programa, las **pruebas de caja blanca** asumen que usted sabe todo acerca del programa. En este caso, el programa es como una casa de vidrio en la cual todo es visible.

Las pruebas de caja blanca son responsabilidad del programador, quien sabe exactamente qué sucede dentro del programa. Usted debe asegurarse de que todas las instrucciones posibles y todas las situaciones posibles se hayan probado. ¡Y ésta no es una tarea simple!

La experiencia ayudará al programador a diseñar un buen plan de pruebas, pero algo que puede hacer desde el principio es adquirir el hábito de escribir planes de prueba. Comenzará su plan de pruebas cuando esté en la etapa de diseño. A medida que construya su diagrama de estructura, se preguntará a sí mismo qué situaciones necesita probar, especialmente las inusuales, y las anotará de inmediato, pues puede suceder que no las recuerde una hora más tarde.

Cuando el programador escribe sus diagramas de flujo o pseudocódigo, debe revisarlos poniendo atención a casos de prueba y tomar nota de los casos que necesita.

Al llegar el momento de construir sus casos de prueba, debe revisar sus notas y organizarlas en conjuntos lógicos. Excepto por programas tipo estudiante muy simples, un conjunto de datos de prueba no validará completamente un programa. Para proyectos de desarrollo a gran escala, tal vez sea necesario ejecutar 20, 30 o más casos de prueba para validar un programa.

Finalmente, cuando realiza las pruebas, pensará en más casos de prueba. De nuevo, los anotará e incorporará a su plan de pruebas. Después de que se ha terminado el programa y éste está en producción, el programador necesitará nuevamente los planes de prueba cuando modifique el programa.

10.2 MODELOS DEL PROCESO DE DESARROLLO

Existen varios modelos para el proceso de desarrollo. Estudiamos dos de los más comunes: el modelo de cascada y el modelo incremental.

MODELO DE CASCADA

Un modelo muy conocido para el proceso de desarrollo es el **modelo de cascada** (figura 10.3). En este modelo, el proceso de desarrollo fluye en una sola dirección. Esto significa que una fase no puede iniciarse hasta que se ha completado la fase anterior. Por ejemplo, la fase de análisis de todo el proyecto debe completarse antes de que su fase de diseño se inicie. La fase de diseño completa debe terminarse antes de que la fase de implementación pueda iniciarse.

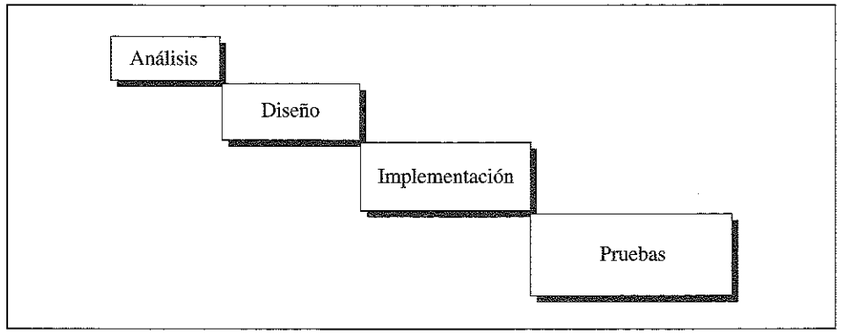


Figura 10.3 Modelo de cascada

Hay ventajas y desventajas en el modelo de cascada. Una ventaja es que cada fase se completa antes de que comience la siguiente. El grupo que trabaja en la fase de diseño, por ejemplo, sabe exactamente qué hacer debido a que se ha completado la fase de análisis. La fase de pruebas puede evaluar el paquete completo ya que todo el proyecto está listo.

Una desventaja del modelo de cascada es la dificultad para localizar un problema. Todo el proceso debe investigarse.

MODELO INCREMENTAL

En el **modelo incremental**, el proceso se desarrolla en una serie de pasos. El grupo de software primero completa una versión simplificada de todo el paquete. La versión representa el paquete completo pero no incluye los detalles.

Esta primera versión por lo general consiste sólo en los módulos principales con llamadas a submódulos vacíos. Por ejemplo, si el paquete tiene diez submódulos, el módulo principal llama a cada uno de ellos, con cada submódulo devolviendo sólo un mensaje indicando que se le llamó.

En la segunda versión, se completan uno o más submódulos, mientras que el resto quedan sin terminar (sólo se comunican). El paquete se prueba de nuevo para demostrar que el módulo principal puede utilizar estos submódulos sin problemas. Si hay un problema aquí, el desarrollador sabe que el problema es con estos submódulos y no con el módulo principal. No se añaden más submódulos hasta que el paquete trabaja en forma apropiada.

Este proceso continúa hasta que todos los submódulos se añaden. La figura 10.4 muestra el concepto del modelo incremental.

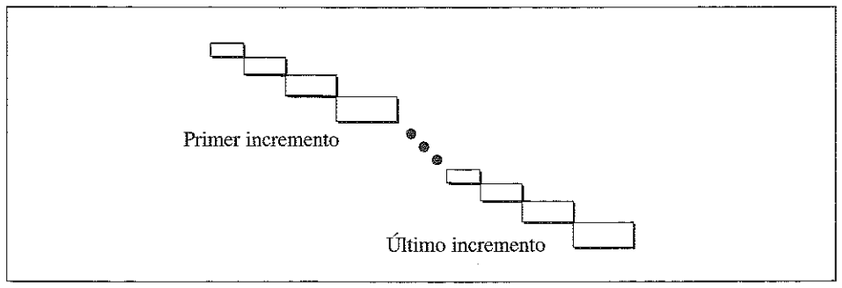


Figura 10.4 Modelo incremental

10.3 MODULARIDAD

Modularidad significa la división de un proyecto grande en partes más pequeñas que pueden entenderse y manejarse más fácilmente. En otras palabras, modularidad significa dividir un problema grande en programas más pequeños que pueden comunicarse entre sí.

HERRAMIENTAS

En la programación se pueden usar dos tipos de herramientas para lograr la modularidad: el diagrama de estructura o el diagrama de clase. Un **diagrama de estructura** se utiliza en la programación procedural para mostrar la relación entre los procedimientos o funciones. Un **diagrama de clase** se utiliza en la programación orientada a objetos para mostrar las relaciones entre las clases.

Se creó un nuevo estándar llamado **Lenguaje de modelado unificado** (UML: *Unified Modeling Language*), el cual incluye herramientas y diagramas para ayudar en lo que a esto se refiere.

ACOPLAMIENTO

El **acoplamiento** es una medida de qué tan estrechamente se ligan dos módulos el uno con el otro. Entre más estrechamente acoplados estén, menos independientes son. Como el objetivo es hacer a los módulos lo más independientes posible, usted quiere que éstos estén acoplados holgadamente. Hay varias razones para desear un **acoplamiento holgado**:

1. Es más probable que las funciones independientes, es decir con un acoplamiento holgado, se vuelvan a utilizar.
2. Las funciones acopladas holgadamente tienen menos probabilidades de crear errores en funciones relacionadas; a la inversa, entre más estrecho sea el acoplamiento, mayor es la probabilidad de que un error en una función genere un error en una función relacionada.
3. Las modificaciones de mantenimiento, las cuales son modificaciones necesarias para implementar nuevos requisitos del usuario, son más fáciles de hacer y menos propensas a crear errores con funciones acopladas holgadamente.

Estudiaremos unos cuantos tipos de acoplamiento.

Acoplamiento de datos

El acoplamiento de datos pasa sólo los datos mínimos requeridos de la función que llama a la función llamada. Todos los datos requeridos se pasan como parámetros y no se pasa ningún dato adicional. Ésta es la mejor forma de acoplamiento y debe utilizarse siempre que sea posible.

Acoplamiento de sello

Las funciones tienen un **acoplamiento de sello** si los parámetros son objetos compuestos como arreglos o estructuras. El acoplamiento de sello no es malo y con frecuencia es necesario. El peligro con el acoplamiento de sello es que a menudo es muy fácil enviar una estructura cuando no se requieren todos los datos de la estructura. Cuando se envían datos adicionales, usted comienza a abrirle la puerta a errores y efectos secundarios indeseables.

Acoplamiento de control

El **acoplamiento de control** es el paso de **banderas** que pueden utilizarse para dirigir el flujo lógico de una función. Es muy parecido al acoplamiento de datos con la excepción de que en vez de datos se pasa una bandera. Cuando se usa adecuadamente, el acoplamiento de control es un método de comunicación entre dos funciones necesario y válido. No obstante, al igual que el acoplamiento de sello, puede hacerse un uso incorrecto del mismo. Si se usa correctamente, comunica el estado: Se llegó al final del archivo. Se encontró el valor de búsqueda.

Por lo general un uso pobre de las banderas es una indicación de un diseño pobre del programa, tal es el caso de un proceso que se divide entre dos o más funciones independientes. Las banderas utilizadas para la comunicación horizontal entre varias funciones en el diagrama de estructura con frecuencia son una indicación de un diseño pobre. Las banderas de acción, contrariamente a las banderas de estado, que requieren que la función receptora realice algún procesamiento especial también son sumamente sospechosas. Un ejemplo de bandera de acción es aquel en que se rechaza la solicitud de compra de un cliente, en lugar de simplemente reportar que se ha excedido el límite de crédito o que no se recibió ningún pago en el último mes.

Acoplamiento global

El **acoplamiento global** utiliza variables globales para comunicarse entre dos o más funciones. Ésta no es una buena técnica de acoplamiento. De hecho, *nunca* debe utilizarse. Hay varias razones por las cuales usted nunca debe utilizar el acoplamiento global. Citaremos sólo las dos más importantes.

1. El acoplamiento global vuelve prácticamente imposible determinar cuáles módulos se están comunicando entre sí. Por consiguiente, cuando se necesita hacer un cambio en un programa, no es posible aislar y evaluar el impacto del cambio. A menudo esto provoca que las funciones que no se cambiaron fallen de repente.
2. El acoplamiento global liga estrechamente una función al programa. Esto significa que no puede transportarse fácilmente a otro programa.

Acoplamiento de contenido

El último tipo de acoplamiento, el **acoplamiento de contenido**, ocurre cuando una función hace referencia directamente a los datos o instrucciones en otra función. Evidentemente, este concepto rompe todos los principios de la programación estructurada. Hacer referencia a los datos en otra función requiere que los datos se hagan visibles externamente fuera de la función.

COHESIÓN

Otro problema en la modularidad es la cohesión. La **cohesión** es una medida de qué tan estrechamente se relacionan los procesos en un programa. Existen varios niveles de cohesión.

Cohesión funcional

Un módulo con **cohesión funcional** contiene sólo un proceso. Éste es el nivel de cohesión más alto y el nivel que usted debe tratar de modelar. Por ejemplo, en la impresión de un informe, la función report debe llamar a tres funciones de un nivel inferior: una para obtener los datos, una para dar formato al título del informe e imprimirlo y otra para dar formato a los datos e imprimirlos.

Sólo una cosa Cada función debe realizar sólo una cosa. Además, todas las instrucciones en la función deben contribuir únicamente a eso. Por ejemplo, supongamos que usted está escribiendo un programa que requiere las medidas estadísticas del promedio y la desviación estándar. Las dos medidas estadísticas están claramente relacionadas por la sola razón de que ambas son medidas de la misma serie de números. Pero usted no calcularía ambas medidas en una sola función, ya que esto implicaría realizar dos cálculos y cada función debe hacer sólo una cosa.

En un lugar El corolario es que una cosa que realiza una función debe realizarse en un solo lugar. Si el código para un proceso se esparce en varias partes distintas del programa y que no guardan ninguna relación, es muy difícil hacer cambios. Por lo tanto, todo el procesamiento para una tarea debe colocarse en una función y, de ser necesario, en sus subfunciones.

Cohesión secuencial

Un módulo con **cohesión secuencial** contiene dos o más tareas relacionadas que se vinculan estrechamente, por lo general con la salida de una que fluye como entrada para la otra. Un ejemplo de cohesión secuencial son los cálculos para una venta. El diseño para esta función podría ser el siguiente:

1. Determinar los precios de los artículos
2. Sumar los artículos
3. Calcular el impuesto sobre las ventas
4. Calcular el total

En este ejemplo, el primer proceso multiplica la cantidad adquirida por el precio. El proceso que calcula la suma de los artículos adquiridos utiliza los precios determinados. Esta suma luego se usa para calcular el impuesto sobre las ventas, el cual finalmente se suma al resultado para obtener el total de la venta. En cada caso, la salida de un proceso se utilizó como la entrada del siguiente proceso.

Aunque es bastante común encontrar el código detallado para estos procesos combinados en una sola función, esto vuelve a la función más compleja y menos reutilizable. La capacidad de reutilización es un motivo de preocupación si ocurren los mismos cálculos o cálculos similares en distintas partes de un programa.

Cohesión de comunicación

La **cohesión de comunicación** combina procesos que trabajan en los mismos datos. Es natural tener una cohesión de comunicación en los módulos superiores en un programa, pero usted nunca debe encontrarla en el nivel primario. Por ejemplo, considere una función que lee un archivo de inventario, imprime el estado actual de las partes y luego lo revisa para ver si alguna parte necesita ordenarse.

Los primeros tres niveles de cohesión se consideran principios de programación bien estructurados. Sin embargo, más allá de este punto, la facilidad de comprensión e implementación, la capacidad de mantenimiento y la precisión comienzan a caer rápidamente. Los siguientes niveles deben utilizarse sólo en los niveles más altos de un diagrama de estructura y luego casi nunca.

Cohesión de procedimiento

El cuarto nivel, la **cohesión de procedimiento**, combina procesos no relacionados que están vinculados mediante un flujo de control. (Esto difiere de la cohesión secuencial, donde los datos fluyen de un proceso al siguiente.)

Cohesión temporal

El quinto nivel, la **cohesión temporal**, es aceptable sólo sobre una variedad limitada de procesos. Combina procesos no relacionados que siempre ocurren juntos. Dos funciones temporalmente cohesivas son la inicialización y la terminación de una tarea. Son aceptables debido a que se utilizan sólo una vez en el programa y porque nunca son portables. Sin embargo, admita que aún deben contener llamadas para las funciones primarias funcionalmente cohesivas siempre que sea factible.

Cohesión lógica y cohesión casual

Los dos últimos niveles pocas veces se encuentran en los programas actuales. La **cohesión lógica** combina procesos que están relacionados sólo por la entidad que los controla. Una función que condicionalmente abrió diferentes series de archivos con base en una bandera que pasa como un parámetro sería lógicamente cohesiva. Finalmente, la **cohesión casual** combina procesos que no guardan ninguna relación entre sí. La cohesión casual existe sólo en teoría. Nunca hemos visto un programa profesional que contenga cohesión casual.

10.4 CALIDAD

No encontrará a nadie que haya llegado a considerar siquiera reducir al mínimo la calidad del software, al menos no lo ha hecho públicamente. Todos quieren el mejor software disponible y al escuchar a los creadores de sistemas en el mercado, todos sus sistemas son perfectos. Aun así, como usuarios de software, a menudo pensamos que el concepto de **software de calidad** es una contradicción. Todos tenemos nuestros productos de software favoritos, pero ninguno de ellos carece de una imperfección o dos.

La gente que anda por el mundo queriendo ser uno de esos creadores de software necesita estar consciente de los conceptos básicos de la calidad del software. En esta sección, estudiamos algunos de los atributos de un producto de calidad y cómo hacer para lograr la calidad.

DEFINICÍON DE CALIDAD

El software de calidad se define como sigue:

> Software que satisface los requisitos explícitos e implícitos del usuario, está bien documentado, cumple con las normas operativas de la organización y se ejecuta de manera eficiente en el hardware para el cual se desarrolló.

Cada uno de estos atributos de buen software recae de lleno en el diseñador y programador del sistema. Observe que colocamos en el programador la carga de satisfacer no sólo los requisitos explícitos del usuario sino también sus necesidades implícitas. Con frecuencia los usuarios no saben completamente qué necesitan. Cuando esto ocurre, es tarea del programador determinar sus requisitos implícitos, los cuales están ocultos en el fondo. Ésta es una tarea verdaderamente formidable.

Pero el software de calidad no es sólo un concepto vago. Si quiere alcanzarlo, tendrá que ser capaz de medirlo. Siempre que sea posible, estas mediciones pueden ser cuantitativas; es decir, deben ser susceptibles de ser medidas numéricamente. Por ejemplo, si una organización es seria respecto a la calidad, debe ser capaz de indicarle el número de errores (*bugs*) por cada mil líneas de código y el tiempo promedio entre las fallas de cada sistema de software que mantiene. Éstas son estadísticas de medición.

Por otra parte, algunas de las mediciones pueden ser cualitativas, lo cual significa que no pueden medirse numéricamente. La flexibilidad y la capacidad de pruebas son ejemplos de mediciones de software cualitativas. Esto no significa que no puedan medirse, sino más bien que se basan en el juicio de una persona para evaluar la calidad de un sistema.

FACTORES DE CALIDAD

La calidad del software puede dividirse en tres mediciones generales: operabilidad, capacidad de mantenimiento y capacidad de transferencia. Cada una de estas mediciones se puede dividir posteriormente como se muestra en la figura 10.5.

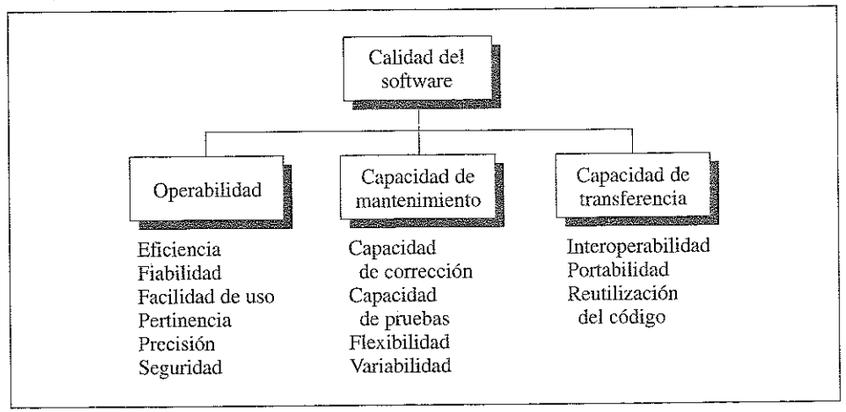


Figura 10.5 Factores de calidad

Operabilidad

La **operabilidad** se refiere a la operación básica de un sistema. Las primeras cosas que un usuario nota respecto a un sistema son su apariencia y sensación. Esto significa, en especial para un sistema en línea interactivo, qué tan fácil e intuitivo es usarlo. ¿Se adapta al sistema operativo bajo el cual se está ejecutando? Por ejemplo, si se está ejecutando en un ambiente Windows, sus menús desplegables y de aparición instantánea deben trabajar del mismo modo que los menús del sistema operativo. En pocas palabras, la operabilidad responde a la pregunta: "¿Cómo se maneja?"

Pero estos factores son subjetivos; no son mesurables. Así que he aquí los factores que conforman la operabilidad; se listan en orden alfabético.

Eficiencia La **eficiencia** es, en general, un término subjetivo. En algunos casos el usuario especificará una norma de rendimiento, tal como una respuesta en tiempo real que debe recibirse en un segundo el 95 por ciento de las veces. Esto es sin duda medible.

Fiabilidad La **fiabilidad** es en realidad la suma de los otros factores. Si los usuarios cuentan con el sistema para lograr terminar su trabajo y tienen confianza en él, entonces lo más probable es que sea confiable. Por otro lado, algunas medidas hablan directamente de la fiabilidad de un sistema, en particular el tiempo promedio entre fallas.

Facilidad de uso Ésta es otra área muy subjetiva. La mejor medición de la **facilidad de uso** es observar a los usuarios y ver si están usando el sistema. Las entrevistas a los usuarios a menudo revelan problemas con la facilidad de uso de un sistema.

Pertinencia La **pertinencia** en la ingeniería de software puede significar varias cosas distintas. ¿El sistema proporciona su salida de manera oportuna? Para los sistemas en línea, ¿el tiempo de respuesta satisface los requisitos de los usuarios? Para los sistemas por lotes, ¿los reportes se entregan oportunamente? También es posible, si el sistema tiene una buena capacidad de inspección para determinar si los datos en el sistema son pertinentes u oportunos. Es decir, ¿los datos se registran dentro de un tiempo razonable después de que ocurre la actividad que los crea?

Seguridad La seguridad de un sistema se refiere a qué tan fácil es para personas no autorizadas llegar a los datos del sistema. Aunque ésta es un área subjetiva, existen listas de control que ayudan en la evaluación de la **seguridad** del sistema. Por ejemplo, ¿el sistema tiene y requiere contraseñas para identificar a los usuarios?

Precisión Un sistema que no es preciso es peor que no tener ningún sistema. La mayoría de los empleados se basan más bien en la intuición y en la experiencia que en un sistema que saben proporciona información falsa y engañosa.

Cualquier sistema que se desarrolle, por lo tanto, debe ser probado a conciencia tanto por un ingeniero de pruebas del sistema como por el usuario. La **precisión** puede medirse a través de métricas tales como el tiempo promedio entre fallas, el número de errores por cada mil líneas de código y el número de solicitudes de usuario para que se le hagan cambios.

Capacidad de mantenimiento

La capacidad de mantenimiento se refiere a mantener a un sistema en ejecución de manera correcta y actualizado. Muchos sistemas requieren cambios regulares, no porque estén pobremente implementados, sino debido a los cambios en los factores externos. Por ejemplo, el sistema de nómina para una compañía debe cambiarse cada año, si no es que con mayor frecuencia, para cumplir con las modificaciones a las leyes y regulaciones gubernamentales.

Capacidad de corrección Una medición de la **capacidad de corrección** es el tiempo promedio de recuperación, el cual es el tiempo requerido para volver a poner un programa en operación después de que éste falla. Aunque ésta es una definición reactiva, actualmente no hay predictores de cuánto tiempo tomará corregir un programa cuando éste falla.

Capacidad de pruebas Usted podría pensar que ésta es un área sumamente subjetiva, pero un ingeniero de pruebas tiene una lista de control de factores que pueden evaluar la **capacidad de pruebas** de un programa.

Flexibilidad Los usuarios constantemente solicitan cambios en los sistemas. La **flexibilidad** es un atributo cualitativo que intenta medir qué tan fácil es hacer estos cambios. Si un programa necesita rescribirse por completo para llevar a cabo un cambio, entonces no es flexible. Por fortuna, este factor ya no es más un problema con la llegada de la programación estructurada.

Variabilidad Qué tan fácil es cambiar un sistema es un factor subjetivo. Sin embargo, los líderes de proyecto experimentados son capaces de calcular cuánto tiempo se llevará la realización de un cambio solicitado. Si se lleva mucho tiempo, tal vez se deba a que es difícil cambiar el sistema. Esto es especialmente cierto para los sistemas viejos.

Hay herramientas de medición de software en el mercado actual que calculan la complejidad y estructura de un programa. Deben utilizarse regularmente, y si la complejidad de un programa es alta, debe considerarse volver a escribir el programa. Los programas que han cambiado muchas veces con los años, a menudo pierden sus centros estructurados y es difícil cambiarlos. También deben rescribirse.

Capacidad de transferencia

La capacidad de transferencia se refiere a la capacidad para mover datos y/o un sistema de una plataforma a otra y para reutilizar el código. En muchas situaciones no es un factor importante. Por otra parte, si usted está escribiendo software generalizado, ésta puede ser crítica.

Interoperabilidad La **interoperabilidad** es la capacidad de enviar datos a otros sistemas. En los sistemas sumamente integrados de hoy, éste es un atributo deseable. De hecho se ha vuelto tan importante que los sistemas operativos ahora soportan la capacidad para mover datos entre sistemas, tales como un procesador de palabras y una hoja de cálculo.

Portabilidad La **portabilidad** es la capacidad para mover software de una plataforma de hardware a otra; por ejemplo, desde una Macintosh al ambiente Windows o de un mainframe IBM al ambiente VAX.

Reutilización del código Si las funciones están escritas de manera que puedan volver a utilizarse en diferentes programas y proyectos, entonces su reutilización es alta. Los buenos programadores escriben bibliotecas o funciones que pueden reutilizar para resolver problemas similares.

EL CÍRCULO DE LA CALIDAD

El primer y más importante punto a reconocer es que la calidad debe planearse en un sistema. No puede agregarse de último momento. Comienza en el paso 1, la determinación de los requisitos del usuario, y continúa a lo largo de la vida del sistema. Debido a que la calidad es un concepto continuo que, al igual que un círculo, nunca termina, nos referimos a ésta como el **círculo de la calidad.** Hay seis pasos para crear software de calidad: herramientas de calidad, revisiones técnicas, evaluación formal, controles de cambio, estándares, medición y creación de informes (figura 10.6).

Nadie puede negar que la calidad comienza con los ingenieros de software asignados al equipo, y que necesitan herramientas de calidad para desarrollar un producto de calidad. Por suerte, las herramientas de desarrollo actuales son excelentes. Un conjunto de software de herramientas de calidad, conocido como ingeniería de software asistida por computadora (CASE: *Computer-Aided Software Engineering*) guía el desarrollo del software a través de los requisitos, el diseño, la programación, las pruebas y la producción. Para el programador hay estaciones de trabajo que no sólo ayudan a escribir el programa, sino también en las pruebas y la depuración del mismo. Por ejemplo, es posible dar seguimiento a las pruebas mediante un programa y luego determinar cuáles instrucciones se ejecutaron y cuáles no. Las herramientas cómo ésta son invaluables para las pruebas de caja blanca.

Otro paso importante en el software de calidad es la revisión técnica. Estas revisiones deben realizarse en cada paso en el proceso de desarrollo, incluyendo los pasos de requisitos, diseño, programación y pruebas. Una revisión de programa común comienza después de que el programador ha diseñado las estructuras de datos y el diagrama de estructura para un programa. Luego se convoca a una junta de revisión del diseño compuesta por el analista de sistemas, el ingeniero de pruebas, un representante del usuario y una o dos personas más. Observe que no se permite asistir a una revisión técnica a ninguna persona del área administrativa. Durante la revisión, el programador explica el método y comenta las interfaces para otros programas mientras los revisores le hacen preguntas y sugerencias.

La calidad también requiere una evaluación formal, ya que ésta asegura que los programas trabajen en conjunto como un sistema y cumplan con los requisitos definidos. Después de que

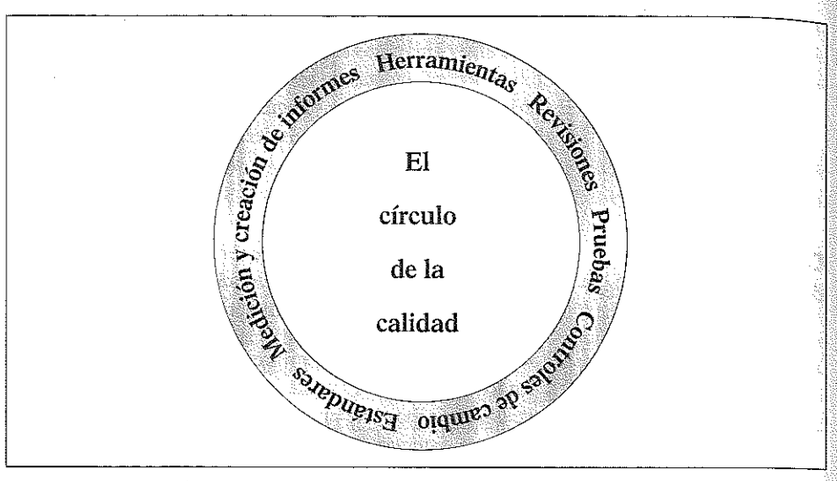


Figura 10.6 Círculo de la calidad

el programador ha completado las pruebas de unidad, el programa pasa a otro ingeniero de software para su integración y las pruebas del sistema. En un proyecto pequeño, es muy probable que el programa pase al analista de sistemas y/o al usuario. En un proyecto grande hay un equipo de pruebas separado.

El desarrollo de los sistemas grandes toma meses y en ocasiones años. Es muy natural que durante periodos largos, ocurran cambios en los requisitos y en el diseño. Para asegurar la calidad, cada cambio debe revisarse y aprobarse por una junta de control de cambios. El impacto de un cambio solicitado en cada programa necesita evaluarse y planearse apropiadamente. Un cambio incontrolado provoca que el calendario se extienda y el presupuesto se rebase, y que los productos sean de baja calidad.

Finalmente, un entorno de buena calidad mide todos los aspectos de la calidad y reporta los resultados con regularidad. Sin medición, usted no puede saber si la calidad es buena o mala, si mejora o empeora. Al mismo tiempo, las normas publicadas proveen el patrón para muchas mediciones de la calidad.

10.5 DOCUMENTACIÓN

Para que un paquete de software se utilice en forma adecuada y se mantenga eficientemente, se necesita la documentación. Por lo general, dos conjuntos independientes de documentación se preparan para un paquete de software: la documentación del usuario y la documentación del sistema.

DOCUMENTACIÓN DEL USUARIO

Para ejecutar el paquete adecuadamente, el usuario necesita documentación, habitualmente llamada el manual, que muestra cómo utilizar el paquete paso a paso. Por lo general contiene una sección de tutorial para guiar al usuario a través de cada característica del paquete. Un buen manual del usuario puede ser una herramienta de marketing muy poderosa. Debe escribirse tanto para el usuario novato como para el experto. La importancia de la documentación del usuario en marketing no puede exagerarse. Un paquete de software con buena documentación del usuario definitivamente aumentará las ventas.

DOCUMENTACIÓN DEL SISTEMA

La **documentación del sistema** define el paquete mismo. Debe redactarse de modo que el paquete pueda mantenerse y modificarse por personas distintas a los desarrolladores originales. Debe haber documentación del sistema para las cuatro fases del desarrollo de sistemas.

Documentación en la fase de análisis

En esta fase, la información recopilada debe documentarse con cuidado. Además, el analista debe definir la fuente de información. Los requisitos y métodos elegidos deben exponerse claramente con las razones en las que se basan.

Documentación en la fase de diseño

En esta fase, las herramientas utilizadas en la copia final deben documentarse. Por ejemplo, si un diagrama de estructura sufre varios cambios, la copia final debe documentarse con explicaciones completas.

Documentación en la fase de implementación

En esta fase debe documentarse cada herramienta y cada programa. Además, el programa por lo general debe autodocumentarse. Hay dos niveles de documentación de programas. El primero es la documentación general al principio del programa. El segundo nivel se encuentra dentro de cada bloque.

Documentación general Cada programa debe comenzar con una descripción general del programa. Enseguida está el nombre del autor y la fecha en que se escribió el programa. A continuación se incluye el historial de cambios del programa. Para un programa de producción que abarca varios años, el historial de cambios puede ser bastante grande.

Documentación de función Además, cuando sea necesario, debe incluirse un breve comentario para bloques de código. Un bloque de código es muy parecido a un párrafo en un informe. Contiene un pensamiento, es decir, una serie de instrucciones que realizan una tarea específica. Los bloques de código en su programa deben estar separados por líneas en blanco, justo como las que inserta en sus reportes entre párrafos.

Documentación en la fase de pruebas

Finalmente, los desarrolladores deben documentar cuidadosamente la fase de pruebas. Cada tipo de prueba aplicada al producto final debe mencionarse junto con el resultado. Incluso los resultados poco favorables y los datos que los produjeron deben documentarse.

DOCUMENTACIÓN COMO UN PROCESO EN CURSO

Observe que la documentación es un proceso en curso. Si el paquete tiene problemas después de su lanzamiento al mercado, éstos deben documentarse. Si el paquete se modifica, todas las modificaciones y sus relaciones con el paquete original también deben documentarse. La documentación se detiene cuando el paquete se vuelve obsoleto.

10.6 TÉRMINOS CLAVE

acoplamiento
acoplamiento de contenido
acoplamiento de control
acoplamiento de datos
acoplamiento global
acoplamiento de sello
acoplamiento holgado
bandera
capacidad de corrección
capacidad de pruebas
ciclo de vida del software
círculo de la calidad
cohesión
cohesión casual
cohesión de comunicación
cohesión de procedimientos
cohesión funcional
cohesión lógica
cohesión secuencial
cohesión temporal
diagrama de clase
diagrama de estructura
diagrama de flujo
documentación del sistema
eficacia
fase de análisis
fase de diseño
fase de implementación
fase de pruebas
fiabilidad
flexibilidad
ingeniería de software
interoperabilidad
modelo de cascada
modelo incremental
modularidad
módulos
operabilidad
pertinencia

portabilidad
precisión
pruebas de caja blanca
pruebas de caja negra
pseudocódigo
reutilización
seguridad
software de calidad
variabilidad

10.7 RESUMEN

- La ingeniería de software es el establecimiento y uso de métodos y principios de ingeniería sólidos para obtener software confiable que trabaje en máquinas reales.
- El proceso de desarrollo para un paquete de software implica cuatro fases: análisis, diseño, implementación y pruebas.
- La fase de análisis del proceso de desarrollo consiste en la definición de los usuarios, necesidades, requisitos y métodos.
- La fase de diseño del proceso de desarrollo consiste en la determinación de los sistemas y el diseño de los archivos y/o bases de datos.
- En la fase de implementación del proceso de desarrollo se escribe el código real.
- En la fase de pruebas del proceso de desarrollo, deben realizarse las pruebas de caja negra y de caja blanca.
- Existen dos modelos de desarrollo de software: el modelo de cascada y el modelo incremental.
- En el modelo de cascada cada módulo se termina completamente antes de que se inicie el siguiente módulo.
- En el modelo incremental, todo el paquete se construye, con cada módulo compuesto por sólo un intérprete de comandos; los módulos ganan complejidad con cada iteración del paquete.
- La modularidad es la división de un programa grande en partes más pequeñas que pueden comunicarse entre sí.
- El acoplamiento es una medida de qué tan estrechamente se ligan dos módulos el uno con el otro. Los tipos de acoplamiento incluyen de datos, de control, global y de contenido.
- La cohesión es una medida de qué tan estrechamente se relacionan los procesos en un programa. Los tipos de cohesión incluyen funcional, secuencial, de comunicación, de procedimiento, temporal, lógica y casual.
- La calidad del software puede dividirse en tres mediciones generales: operabilidad, capacidad de mantenimiento y capacidad de transferencia.
- Factores importantes para la operabilidad del software son la precisión, eficiencia, fiabilidad, seguridad, pertinencia y facilidad de uso.
- Factores importantes para la capacidad de mantenimiento son la variabilidad, capacidad de corrección, flexibilidad y capacidad de pruebas.
- Factores importantes para la capacidad de transferencia son la reutilización del código, interoperabilidad y portabilidad.
- Hay seis pasos para el software de calidad: estándares, medición y creación de informes, herramientas de calidad, revisiones técnicas, evaluación formal y controles de cambio.
- La documentación del usuario y la documentación del sistema son necesarias para que un paquete de software se utilice de manera apropiada y se mantenga de manera eficiente.

10.8 PRÁCTICA

PREGUNTAS DE REPASO

1. Defina la ingeniería de software.
2. ¿Qué se quiere decir con el ciclo de vida del software?
3. ¿Cuándo el software se vuelve obsoleto?
4. ¿Cuáles son las cuatro fases en el desarrollo del software?
5. ¿Qué implica la fase de análisis del desarrollo de sistemas?
6. ¿Qué son las pruebas de caja negra?
7. ¿Qué son las pruebas de caja blanca?
8. ¿Cuál es la diferencia entre cohesión y acoplamiento?

PREGUNTAS DE OPCIÓN MÚLTIPLE

9. Los principios de ingeniería de software se establecieron por primera vez hace _____ años.
   a. 10
   b. 30
   c. 100
   d. 1000

10. Un sistema de software se vuelve obsoleto cuando_____.
    a. se encuentra un error en el código
    b. el lenguaje con el cual se escribe ya no se utiliza
    c. el programador principal abandona el proyecto
    d. a y b

11. Una fase en el desarrollo de sistemas es _____.
    a. análisis
    b. pruebas
    c. diseño
    d. todas las anteriores

12. La definición de los usuarios, las necesidades, los requisitos y los métodos es parte de la fase de _____.
    a. análisis
    b. diseño
    c. implementación
    d. pruebas

13. En el proceso de desarrollo de sistemas, la escritura del código es parte de la fase de _____.
    a. análisis
    b. diseño
    c. implementación
    d. pruebas

14. En el proceso de desarrollo de sistemas, el diagrama de estructura es una herramienta utilizada en la fase de _____.
    a. análisis
    b. diseño
    c. implementación
    d. pruebas

15. En el proceso de desarrollo de sistemas, el diagrama de flujo es una herramienta usada en la fase de _____.
    a. análisis
    b. diseño
    c. implementación
    d. pruebas

16. En el proceso de desarrollo de sistemas, el pseudocódigo es una herramienta usada en la fase de _____.
    a. análisis
    b. diseño
    c. implementación
    d. pruebas

17. Realizar pruebas a un paquete de software puede involucrar las pruebas de _____.
    a. caja negra
    b. caja blanca
    c. caja de pan
    d. a y b

18. Las pruebas de caja negra se realizan por el_____.
    a. usuario
    b. ingeniero de pruebas del sistema
    c. programador
    d. a y b

19. Las pruebas de caja blanca se realizan por el_____.
    a. programador
    b. usuario
    c. ingeniero de pruebas del sistema
    d. CTO

20. En la primera versión del modelo _____, cada módulo llamado sólo devuelve un mensaje indicando que se le llamó.
    a. cascada
    b. incremental
    c. instrumental
    d. de caja negra

21. En el modelo de _____, una fase completa del proyecto se termina antes de que comience la fase siguiente.
    a. cascada
    b. incremental
    c. instrumental
    d. de caja negra

22. El(la) _____ es la separación de un programa grande en partes pequeñas.
    a. acoplamiento
    b. aumento incremental
    c. obsolescencia
    d. modularidad

23. La modularidad se vuelve más visible mediante herramientas como _____.
    a. el diagrama de estructura
    b. el diagrama de clase
    c. la cascada incremental
    d. a y b

24. El(la) _____ es una medida de qué tan estrechamente se ligan dos módulos el uno con el otro.
    a. modularidad
    b. acoplamiento
    c. interoperabilidad
    d. cohesión

25. El(la) _____ una medida de qué tan estrechamente se relacionan los procesos en un programa.
    a. modularidad
    b. acoplamiento
    c. interoperabilidad
    d. cohesión

26. El acoplamiento de _____ sólo pasa el mínimo de datos requerido de la función que llama a la función llamada.
    a. datos
    b. sello
    c. control
    d. global

27. El acoplamiento de _____ ocurre cuando una función se refiere directamente a los datos o instrucciones en otra función.
    a. datos
    b. sello
    c. contenido
    d. control

28. El acoplamiento de _____ utiliza variables globales para comunicarse entre dos o más funciones.
    a. datos
    b. sello
    c. contenido
    d. control

29. El acoplamiento de _____ pasa banderas que pueden dirigir el flujo lógico de una función.
    a. datos
    b. sello
    c. control
    d. global

30. El acoplamiento de _____ pasa parámetros que son objetos compuestos como arreglos o estructuras.
    a. datos
    b. sello
    c. control
    d. global

31. La cohesión _____ es el nivel de cohesión más alto.
    a. funcional
    b. secuencial
    c. de comunicación
    d. lógica

32. La cohesión _____ combina procesos no relacionados que siempre ocurren juntos.
    a. lógica
    b. de procedimiento
    c. temporal
    d. funcional

33. La cohesión _____ combina procesos que sólo están relacionados por la entidad que los controla.
    a. lógica
    b. de procedimiento
    c. temporal
    d. funcional

34. La cohesión _____ combina dos o más tareas relacionadas que están estrechamente ligadas entre sí.
    a. funcional
    b. secuencial
    c. de comunicación
    d. lógica

35. La cohesión _____ combina procesos no relacionados que están ligados por un control de flujo.
    a. funcional
    b. secuencial
    c. de procedimiento
    d. lógica

36. La cohesión _____ combina procesos que trabajan sobre los mismos datos.
    a. funcional
    b. secuencial
    c. de comunicación
    d. lógica

37. La precisión, eficiencia, fiabilidad, seguridad, pertinencia y facilidad de uso son factores importantes para la _____ del software.
    a. operabilidad
    b. capacidad de mantenimiento
    c. capacidad de transferencia
    d. longevidad

38. La variabilidad, capacidad de corrección, flexibilidad y capacidad de pruebas son factores importantes para la _____ del software.
    a. operabilidad
    b. capacidad de mantenimiento
    c. capacidad de transferencia
    d. longevidad

39. La reutilización de código, la interoperabilidad y la portabilidad son factores importantes para la _____ del software.
    a. operabilidad
    b. capacidad de mantenimiento
    c. capacidad de transferencia
    d. longevidad

EJERCICIOS

40. Una función se escribe para encontrar el número más pequeño entre una lista de números. La lista se pasa a la función como un arreglo. El número menor se devuelve a la función que llama. ¿Qué tipo(s) de acoplamiento se usó entre la función que llama y la función llamada?

41. Se escribe una función para ordenar una lista de números. La función utiliza la lista en la función que llama, pero devuelve una bandera para mostrar si se realizó el ordenamiento satisfactoriamente o no. ¿Qué tipo(s) de acoplamiento se utilizó entre la función que llama y la función llamada?

42. Se escribe una función para intercambiar datos entre dos variables. La función utiliza directamente una variable definida en la función que llama. ¿Qué tipo de acoplamiento se utilizó?

43. Una programadora escribe un programa que incluye una función de suma. Posteriormente escribe otro programa que también requiere una suma. Cuando trata de usar la función anterior, se da cuenta que debe escribir una función completamente nueva. ¿Qué principio de calidad se viola aquí?

44. Imagine que le asignan la tarea de realizar la documentación del sistema en la fase de análisis para un gran proyecto. Idee la hoja de plantilla apropiada para utilizarla en los cuatro pasos involucrados en esta fase.

45. Repita el ejercicio 44 para la fase de diseño.

46. Repita el ejercicio 44 para la fase de implementación.

47. Repita el ejercicio 44 para la fase de pruebas.

V

Organización de datos

Estructuras de datos

En los capítulos anteriores utilizamos variables que almacenan una sola entidad. Aunque las variables individuales se utilizan ampliamente en los lenguajes de programación, no pueden usarse para resolver problemas complejos de una manera eficiente.

En este capítulo presentamos la estructura de datos. Una **estructura de datos** utiliza una colección de variables relacionadas a las que se puede acceder de manera individual o como un todo. En otras palabras, una estructura de datos representa un conjunto de elementos de datos con una relación específica entre ellos.

Analizamos tres estructuras de datos en este capítulo: arreglos, registros y listas ligadas. La mayoría de los lenguajes de programación tiene una implementación implícita de los primeros dos. La tercera, sin embargo, se simula utilizando apuntadores y registros.

Este capítulo es un preludio para el siguiente capítulo, en el cual presentamos tipos de datos abstractos (ADT: *abstract data type*).

11.1 ARREGLOS

Imagine que tiene un problema que requiere el procesamiento de veinte números. Usted necesita leerlos, procesarlos e imprimirlos. También debe mantener estos veinte números en la memoria para la duración del programa. Puede definir veinte **variables**, cada una con un nombre diferente, como se muestra en la figura 11.1.

Pero tener veinte nombres diferentes crea otro problema. ¿Cómo puede leer veinte números desde el teclado y almacenarlos? Para hacerlo, necesita veinte referencias, una para cada variable. Además, una vez que éstas estén en la memoria, ¿cómo puede usted imprimirlas? Para imprimirlas necesita otras veinte referencias. En otras palabras, necesita el diagrama de flujo de la figura 11.2 para leer, procesar e imprimir estos veinte números.

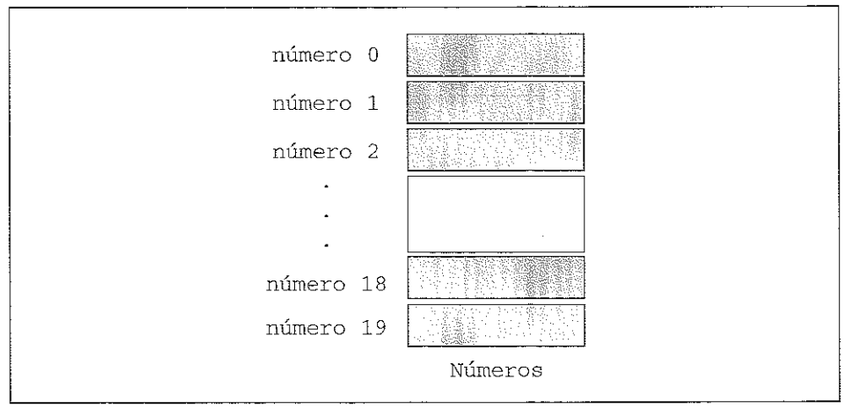


Figura 11.1 Veinte variables individuales

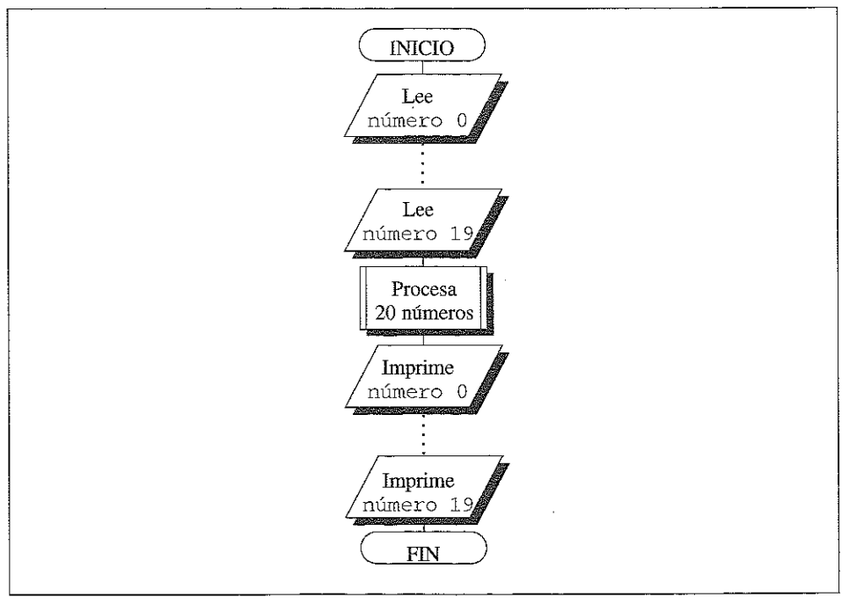


Figura 11.2 Procesamiento de variables individuales

Aunque esto puede ser aceptable para veinte números, definitivamente no es aceptable para 200, 2 000 o 20 000 números. Para procesar grandes cantidades de datos se necesita una estructura de datos poderosa, tal como un arreglo. Un **arreglo** es una colección en secuencia, de tamaño fijo, de elementos del mismo tipo. Dado que un arreglo es una colección en secuencia, usted puede referirse a los demás elementos en el arreglo como el primer elemento, el segundo elemento y así sucesivamente hasta que llegue al último elemento. Si usted fuera a poner veinte números en un arreglo, podría designar el primer elemento como $número_0$ como se muestra en la figura 11.1. De una manera similar, podría referirse al segundo número como $número_1$ y al tercero como $número_2$. Al continuar la serie el último número sería $número_{19}$. El **subíndice** indica el número ordinal del elemento contado desde el principio del arreglo.

Lo que usted ha visto es que los elementos del arreglo se direccionan a través de sus subíndices (figura 11.3). El arreglo como un todo tiene un nombre, número, pero se puede tener acceso a cada número en forma individual a través de su subíndice.

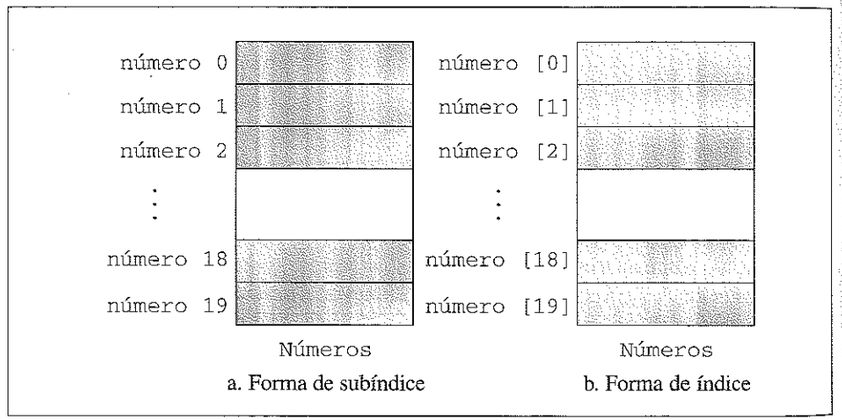


Figura 11.3 Arreglos con subíndices e índices

Las ventajas del arreglo estarían limitadas si usted no tuviera además constructores de programación que le permitan procesar los datos de una manera más conveniente. Por fortuna, hay un poderoso conjunto de constructores de programación, **ciclos**, que facilitan el procesamiento de arreglos.

Usted puede utilizar ciclos para leer y escribir elementos en un arreglo. Los ciclos se pueden usar para sumar, restar, multiplicar y dividir los elementos. Usted también puede usar ciclos para un procesamiento más complejo tal como el cálculo de promedios. Ahora no importa si hay 2, 20, 200, 2 000 o 20 000 elementos a ser procesados. Los ciclos facilitan el manejo de todos ellos.

Pero una pregunta sigue sin respuesta: ¿Cómo puede escribir una instrucción de modo que una vez se refiera al primer elemento de un arreglo y la siguiente vez se refiera a otro elemento? En realidad es muy simple: Tan sólo se pide prestado el concepto de subíndice que se ha estado usando. No obstante, en lugar de usar subíndices, se coloca el valor del subíndice entre corchetes. Usando esta notación, usted hace referencia al $número_0$ como número[0].

Siguiendo esta convención, el $número_1$ se convierte en el número[1] y el $número_{19}$ se convierte en el número[19]. Esto se conoce como indexación. Usando una referencia típica, usted ahora hace referencia a su arreglo utilizando la variable. El diagrama de flujo para procesar sus veinte números utilizando un arreglo y los ciclos aparece en la figura 11.4.

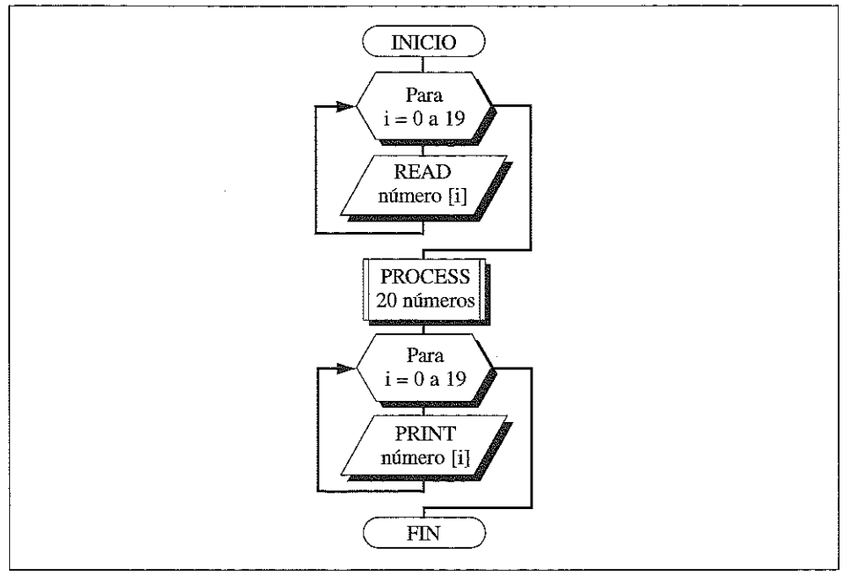


Figura 11.4 Procesamiento de un arreglo

APLICACIONES DE LOS ARREGLOS

Arreglos de frecuencia

En esta sección estudiamos una aplicación de los arreglos: el arreglo de frecuencia y su representación gráfica.

Un **arreglo de frecuencia** muestra el número de elementos con el mismo valor encontrado en una serie de números. Por ejemplo, suponga que ha tomado una muestra de 100 valores entre 0 y 19. Usted quiere saber cuántos de los valores son 0, cuántos son 1, cuántos son 2 y así sucesivamente hasta 19.

Puede leer estos valores en un arreglo llamado números. Luego se crea un arreglo de veinte elementos que mostrará la frecuencia de cada valor en la serie (figura 11.5).

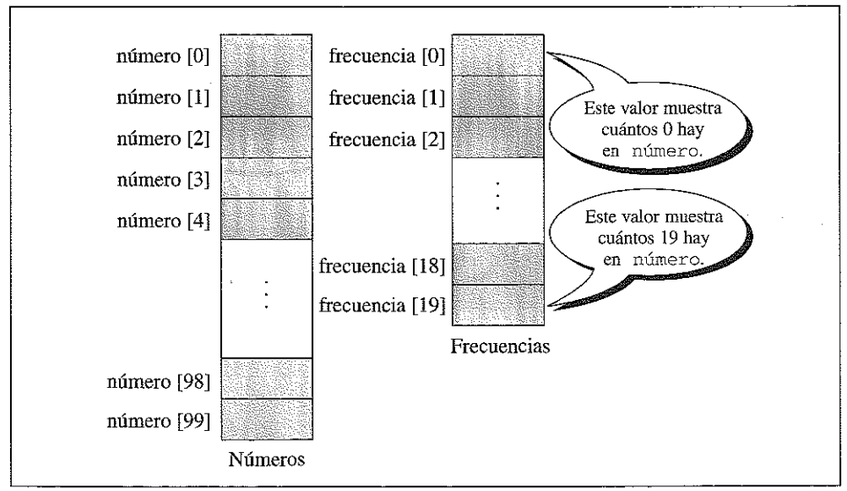


Figura 11.5 Arreglo de frecuencia

Histogramas

Un **histograma** es una representación pictórica de un arreglo de frecuencia. En lugar de imprimir los valores de los elementos para mostrar la frecuencia de cada número, se imprime un histograma en forma de **gráfica de barras**. Por ejemplo, la figura 11.6 es un histograma para una serie de números en el rango de 0 a 19. En este ejemplo, los asteriscos (*) se utilizan para construir la barra. Cada asterisco representa una ocurrencia del valor de datos.

ARREGLOS BIDIMENSIONALES

Los arreglos analizados hasta ahora se conocen como arreglos **unidimensionales** debido a que los datos se organizan linealmente en una sola dirección. Muchas aplicaciones requieren que los datos se almacenen en más de una dimensión. Un ejemplo común es una tabla, que es un arreglo que consiste en filas y columnas. La figura 11.7 muestra una tabla, a la cual por lo general se le llama **arreglo bidimensional**. Observe que los arreglos pueden tener tres, cuatro o más dimensiones. Sin embargo, el análisis de los **arreglos multidimensionales** está más allá del ámbito de este libro.

Disposición en la memoria

Los índices en la definición de un arreglo bidimensional representan filas y columnas. Este formato relaciona la forma en que los datos se disponen en la **memoria**. Si usted fuera a considerar la memoria como una fila de bytes con las direcciones inferiores a la izquierda y las direcciones superiores a la derecha, entonces se podría colocar un arreglo en la memoria con el primer elemento a la izquierda y el último elemento a la derecha. De manera similar, si el arreglo es bidimensional, entonces la primera dimensión es una fila de elementos que se almacena a la izquierda. Esto se conoce como **almacenamiento de "filas mayores"** (figura 11.8).

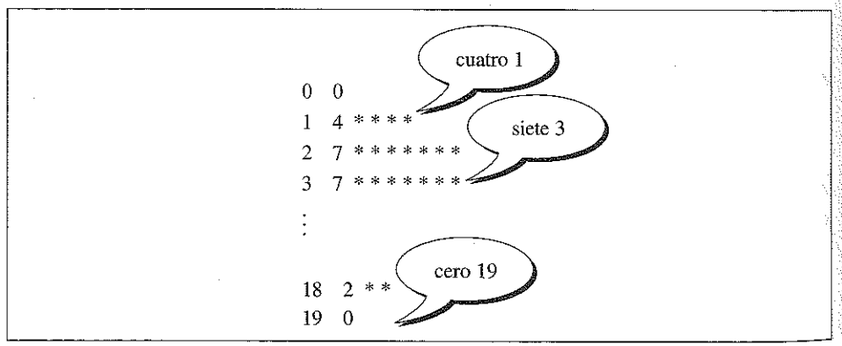


Figura 11.6 Histograma

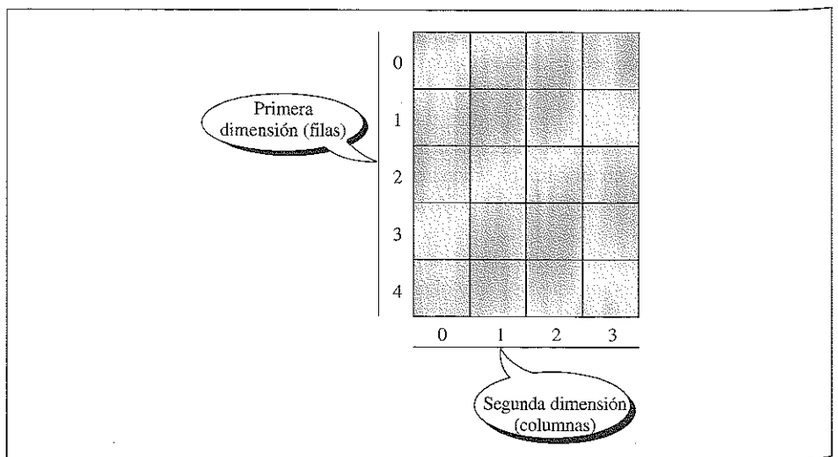


Figura 11.7 Arreglo bidimensional

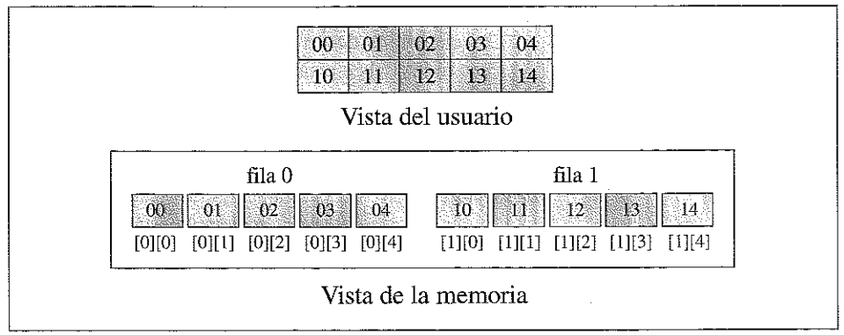


Figura 11.8 Disposición en la memoria

11.2 REGISTROS

Un **registro** es una colección de elementos relacionados, posiblemente de distintos tipos, que tienen un solo nombre. Cada elemento de un registro se llama campo. Un **campo** es el elemento más pequeño de datos con nombre que tiene un significado. Tiene un tipo y existe en la memoria. Se le pueden asignar valores, a los que se tiene acceso por selección o manipulación. Un campo difiere de una variable principalmente en que éste es parte de un registro.

La diferencia entre un arreglo y un registro radica en que todos los elementos en un arreglo deben ser del mismo tipo, mientras que los elementos en un registro pueden ser del *mismo tipo* o de uno *diferente*.

La figura 11.9 contiene dos ejemplos de registros. El primer ejemplo, `fracción`, tiene dos campos, ambos formados por enteros. El segundo ejemplo, `estudiante`, tiene tres campos formados por dos tipos distintos.

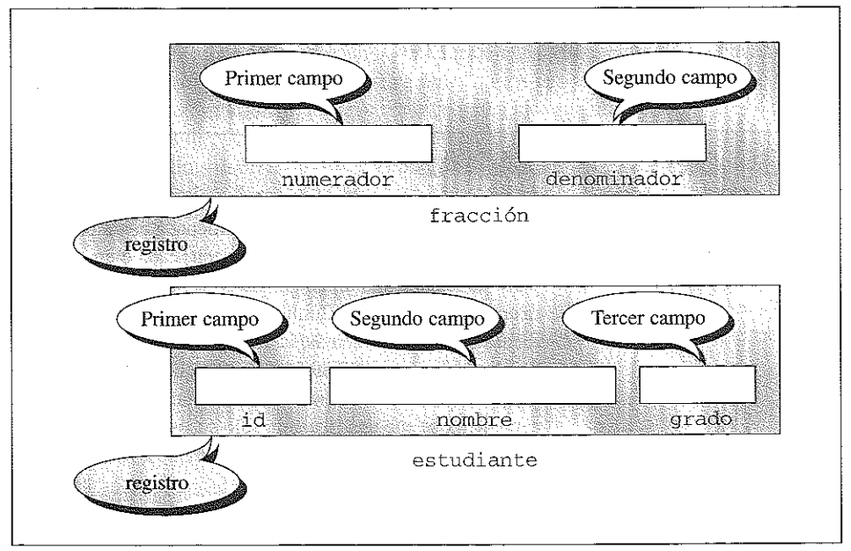


Figura 11.9 Registros

Los elementos de un registro pueden ser del mismo tipo o de tipos diferentes. Pero todos deben estar relacionados.

Sin embargo, hay una advertencia de diseño. Los datos de un registro deben estar relacionados a un objeto. En la figura 11.9, los enteros de la fracción pertenecen a la misma fracción, y los datos en el segundo ejemplo están relacionados con un estudiante. No combine datos que no están relacionados por conveniencia de programación.

ACCESO A REGISTROS

Primero se describirá cómo acceder a componentes individuales de un registro y después se examinará la asignación de todos los registros.

Acceso a campos individuales

Se puede tener acceso y manipular cada campo de un registro al emplear expresiones y operadores. Todo lo que se puede hacer con una variable individual se puede hacer también con un campo del registro. El problema está en identificar los campos individuales en los que usted está interesado.

Dado que cada campo en un registro tiene un nombre, se puede usar simplemente el nombre. El problema con esta sencilla opción es que si usted quiere comparar el `id` de un estudiante en un registro con el `id` de otro estudiante en otro registro, la instrucción podría ser ambigua. Por lo tanto, necesita identificar de algún modo los registros que contienen identificadores de campo, en este caso los `id`. Muchos lenguajes de programación utilizan un punto (.) para separar el nombre del registro del nombre del campo.

Por ejemplo, si tiene dos registros (`Estudiante1` y `Estudiante2`) del tipo *estudiante*, puede referirse a los campos individuales de esos registros como sigue:

```
Estudiante1.id,   Estudiante2.nombre   y   Estudiante1.grado
Estudiante2.id,   Estudiante2.nombre   y   Estudiante2.grado
```

También puede leer y escribir datos de los miembros de un registro del mismo modo que lo hace en las variables individuales.

11.3 LISTAS LIGADAS

Una **lista ligada** es una colección ordenada de datos en la que cada elemento contiene la ubicación del siguiente elemento; es decir, cada elemento contiene dos partes: datos y **liga**. La parte de los datos mantiene la información útil, los datos a procesarse. La liga se emplea para encadenar los datos. Contiene un **apuntador** (una dirección) que identifica el siguiente elemento en la lista. Además, la variable apuntador identifica el primer elemento en la lista. El nombre de la lista es el mismo nombre que el de esta variable apuntador. La sencilla lista ligada que se describe aquí se conoce comúnmente como **lista simplemente ligada** porque únicamente contiene una liga a un solo sucesor. Una lista ligada es un ejemplo de una estructura de datos derivada en la que la estructura no existe dentro del lenguaje pero está simulada por otras estructuras disponibles.

La figura 11.10 muestra una lista ligada, `pList` –por apuntador (*pointer*) en el encabezamiento de la lista–, que contiene cuatro elementos. La liga en cada elemento, excepto el último, apunta a su sucesor. La liga en el último elemento contiene un **indicador nulo**, que indica

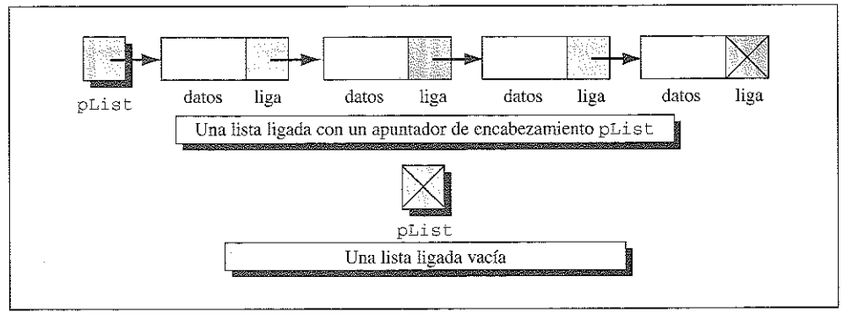


Figura 11.10 Listas ligadas

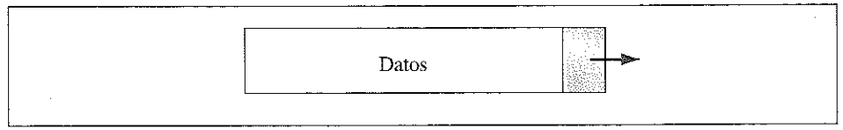


Figura 11.11 Nodo

el final de la lista. Definimos como lista ligada vacía aquella que tiene un apuntador de encabezamiento nulo. La figura 11.10 también contiene un ejemplo de una lista ligada vacía.

NODOS

Los elementos de una lista ligada tradicionalmente se llaman nodos. Un **nodo** de una lista ligada es un registro que tiene al menos dos campos: uno contiene los datos y el otro la dirección del siguiente nodo en la secuencia (figura 11.11).

Los nodos de una lista ligada se llaman **registros autorreferenciales**. En un registro autorreferencial, cada instancia del registro contiene un apuntador a otra instancia del mismo tipo estructural.

APUNTADORES A LISTAS LIGADAS

Una lista ligada siempre debe tener un apuntador de encabezamiento. Dependiendo de cómo se empleará la lista, también es posible tener varios apuntadores. Por ejemplo, si va a buscar en una lista ligada, sin duda debe tener un apuntador a la ubicación (`pLoc`) donde encontrará los datos que busca. En muchos registros, el programa es más eficiente si existen apuntadores al último nodo en la lista y a apuntadores de encabezamiento. Este último apuntador por lo general se llama `pLast` (apuntador al último) o `pRear` (apuntador a la parte posterior).

OPERACIONES EN LISTAS LIGADAS

Hay cinco operaciones básicas en una lista ligada, las cuales son suficientes para resolver cualquier problema en la secuencia de la lista. Si una aplicación requiere operaciones adicionales en la lista, éstas se pueden agregar fácilmente.

Inserción de un nodo

Para insertar un nodo en una lista ligada, siga estos tres pasos (figura 11.12):

1. Asigne memoria para el nuevo nodo y escriba los datos.
2. Haga que el nuevo nodo apunte a su sucesor.
3. Haga que el predecesor apunte al nuevo nodo.

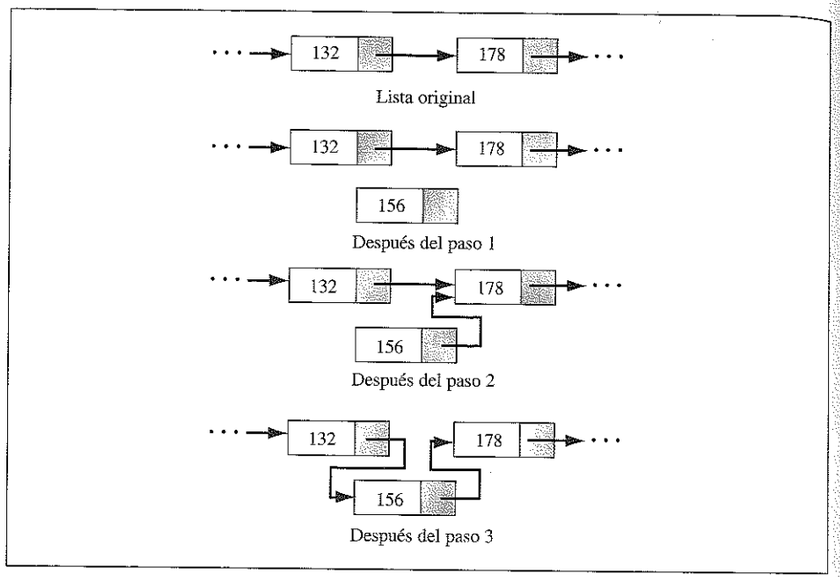


Figura 11.12 Inserción de un nodo

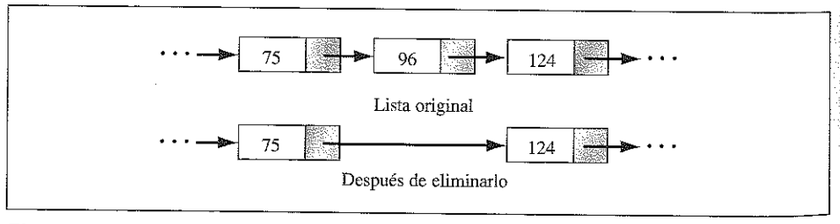


Figura 11.13 Eliminación de un nodo

Observe que en algunos casos especiales (al agregar al final de una lista, al agregar al comienzo de una lista o al agregar a una lista vacía), estos tres pasos deben modificarse según el caso.

Eliminación de un nodo

Para eliminar un nodo primero debe ubicar el nodo mismo. Una vez que localice el nodo a eliminar, simplemente cambie el campo de la liga del predecesor para que apunte al sucesor del nodo eliminado (figura 11.13). Sin embargo, debe tener cuidado al eliminar el único nodo de una lista porque esto dará como resultado una lista vacía. Recuerde que la eliminación del primer nodo o del único nodo son casos especiales que necesitan un manejo especial.

Búsqueda en una lista

La búsqueda en una lista se emplea por otras operaciones para localizar datos en una lista. Por ejemplo, para insertar datos necesita conocer el predecesor lógico de los nuevos datos. Para eliminar datos, necesita encontrar el nodo a ser eliminado e identificar a su predecesor lógico. Para recuperar datos de una lista, necesita buscar en la lista y encontrar los datos. Además, muchas aplicaciones de usuarios requieren que se realicen búsquedas en las listas para localizar datos.

Para **buscar una lista** en una clave, necesita un campo de clave. Para las listas sencillas, la clave y los datos pueden estar en el mismo campo. Para registros más complejos se necesita un campo de clave independiente.

Dada una clave objetivo, la lista ordenada intenta localizar el nodo solicitado en la lista ligada. Si un nodo en la lista concuerda con su valor objetivo, entonces la búsqueda regresa un valor de verdadero; si no hay coincidencias con las claves, devuelve el valor de falso.

Recuperación de un nodo

Ahora que sabe cómo localizar un nodo en la lista, podemos describir la recuperación de un nodo. Para recuperar un nodo, busque en la lista para localizar los datos. Si éstos se encuentran, se mueven al área de salida en el módulo que llama. Si éstos no se encuentran, no se recuperará nada.

Cruce de una lista

Los algoritmos que cruzan una lista inician en el primer nodo y examinan cada nodo en sucesión hasta que se ha procesado el último nodo. El cruce lógico se utiliza por varios tipos de algoritmos, tales como el cambio de un valor en cada nodo, la impresión de la lista, la suma de un campo en la lista o el cálculo del promedio de un campo. Cualquier aplicación que requiera que la lista entera se procese utiliza un cruce.

Para cruzar la lista, usted necesita un apuntador auxiliar, el cual es un apuntador que se mueve de un nodo a otro conforme se procesa cada elemento. Comience por establecer el apuntador auxiliar para el primer nodo en la lista. Luego, utilizando un ciclo, continúe hasta que todos los datos se han procesado. Cada ciclo llama a un módulo de proceso y le pasa los datos, luego avanza el apuntador auxiliar al siguiente nodo. Cuando el último nodo se ha procesado, el apuntador auxiliar se vuelve nulo y el ciclo termina (figura 11.14).

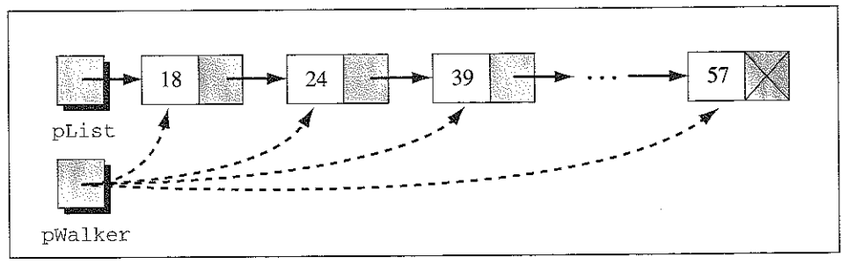


Figura 11.14 Cruce de una lista

11.4 TÉRMINOS CLAVE

almacenamiento de filas mayores
apuntador
apuntador nulo
arreglo
arreglo bidimensional
arreglo de frecuencia
arreglo multidimensional
arreglo unidimensional
búsqueda en una lista
campo
ciclo
estructura de datos
gráfica de barras
histograma
liga
lista ligada
lista ligada individualmente
memoria
nodo
registro
registro autorreferencial
subíndice
variable

11.5 RESUMEN

- Arreglos, registros y listas ligadas son estructuras de datos que se estudian en este capítulo.
- Un arreglo unidimensional es una secuencia de tamaño fijo de elementos del mismo tipo.
- Puede acceder a los elementos individuales de un arreglo utilizando el nombre del arreglo y el índice.
- Puede procesar los valores de un arreglo usando un ciclo.
- Un arreglo de frecuencia es aquel cuyos elementos muestran el número de ocurrencias de valores de datos de otro arreglo.
- Un histograma es una representación gráfica de un arreglo de frecuencia.
- Un arreglo bidimensional es una presentación en una tabla con filas y columnas.
- Un arreglo multidimensional es una extensión de un arreglo bidimensional a uno de tres, cuatro o más dimensiones.
- Un registro es un conjunto de elementos relacionados, posiblemente de diferentes tipos, que tienen un solo nombre.
- Cada elemento de un registro se llama campo.
- Una diferencia entre un arreglo y un registro es que todos los elementos dentro del arreglo deben ser del mismo tipo, mientras que los elementos del registro pueden ser del mismo tipo o de uno diferente.
- Una lista ligada es un conjunto de datos ordenados en la cual cada elemento contiene la ubicación (dirección) del siguiente elemento; es decir, cada elemento contiene dos partes: datos y liga.
- El nodo en una lista ligada individualmente contiene únicamente una liga a un solo sucesor, a menos que sea el último, en cuyo caso no está ligado con ningún otro nodo.
- Cuando se desea insertar un elemento en una lista ligada, deben considerarse cuatro casos: agregar a una lista vacía, agregar al inicio de la lista, agregar a la mitad y agregar al final.
- Cruzar una lista ligada significa recorrer la lista, nodo por nodo, y procesar cada nodo. Tres ejemplos de cruces de listas son el conteo de la cantidad de nodos, la impresión del contenido de los nodos y la suma de los valores de uno o más campos.

11.6 GRUPO DE PRÁCTICAS

PREGUNTAS DE REPASO

1. ¿Por qué existe la necesidad de las estructuras de datos?
2. Mencione tres tipos de estructuras de datos.
3. ¿Qué es un arreglo?
4. ¿En qué difiere un elemento en un arreglo de un elemento en un registro?
5. ¿Cuál es la diferencia entre un elemento en un arreglo y un elemento en una lista ligada?
6. ¿Por qué debe usar índices en lugar de subíndices para identificar los elementos de un arreglo?
7. ¿Qué es un arreglo de frecuencia?
8. ¿Cómo se relaciona un histograma con un arreglo de frecuencia?
9. ¿Cómo se almacenan los elementos de un arreglo en la memoria?
10. ¿Cuál es la definición de un campo en un registro?
11. ¿Cuáles son los campos de un nodo en una lista ligada?
12. ¿Cuál es la función del apuntador en una lista ligada?
13. ¿Cómo se puede apuntar hacia el primer nodo en una lista ligada?
14. ¿A qué apunta el apuntador en el último nodo de una lista ligada?
15. ¿Qué es una lista ligada individualmente?
16. Describa el predecesor del apuntador cuando se agrega un nodo a:
    a. el inicio de una lista ligada
    b. el punto medio de la lista ligada
    c. el final de una lista ligada, y
    d. una lista vacía

PREGUNTAS DE OPCIÓN MÚLTIPLE

17. Una estructura de datos puede ser _____.
    a. un arreglo
    b. un registro
    c. una lista ligada
    d. todas las anteriores

18. _____ es un conjunto de tamaño fijo y secuenciado de elementos del mismo tipo de datos.
    a. un arreglo
    b. un registro
    c. una lista ligada
    d. una variable

19. Dado el arreglo llamado `objeto` con veinte elementos, si usted ve el término $\texttt{objeto}_{10}$, sabrá que el arreglo tiene una forma de _____.
    a. variable
    b. registro
    c. índice
    d. subíndice

20. Dado el arreglo llamado `objeto` con veinte elementos, si usted ve el término `objeto[10]`, sabrá que el arreglo tiene una forma de _____.
    a. variable
    b. registro
    c. índice
    d. subíndice

21. Un(a) _____ es una representación gráfica de un arreglo de frecuencia.
    a. lista ligada
    b. histograma
    c. registro
    d. nodo

22. Un arreglo que consiste simplemente en filas y columnas probablemente es un arreglo _____.
    a. unidimensional
    b. bidimensional
    c. tridimensional
    d. multidimensional

23. En un arreglo bidimensional con cuatro filas, la fila con las direcciones superiores en la memoria es la _____ fila.
    a. primera
    b. segunda
    c. tercera
    d. cuarta

24. Un(a) _____ es un conjunto de elementos relacionados, posiblemente de distintos tipos, que tienen un solo nombre.
    a. arreglo
    b. registro
    c. lista ligada
    d. todas las anteriores

25. Cada elemento de un registro se conoce como _____.
    a. una variable
    b. un índice
    c. un campo
    d. un nodo

26. Todos los miembros de un arreglo deben ser de _____.
    a. el mismo tipo
    b. distintos tipos
    c. tipo entero
    d. tipo carácter

27. Todos los miembros de un registro deben ser de _____.
    a. el mismo tipo
    b. tipos relacionados
    c. tipo entero
    d. tipo carácter

28. Un(a) _____ es un conjunto ordenado de datos en el cual cada elemento contiene la ubicación del siguiente elemento.
    a. arreglo
    b. registro
    c. lista ligada
    d. nodo

29. En una lista ligada, cada elemento contiene _____.
    a. datos
    b. una liga
    c. un registro
    d. a y b

30. El(la) _____ es un apuntador que identifica el siguiente elemento dentro de la lista ligada.
    a. liga
    b. nodo
    c. arreglo
    d. a o b

31. Dada una lista ligada llamada `niños`, la variable del apuntador `niños` identifica al _____ elemento de la lista ligada.
    a. primer
    b. segundo
    c. último
    d. cualquiera

32. Una lista ligada vacía consiste de _____.
    a. una variable
    b. dos nodos
    c. datos y una liga
    d. un apuntador de encabezamiento nulo

33. En un registro autorreferencial, cada instancia del registro contiene un apuntador hacia otra instancia de _____ tipo.

a. el mismo
b. diferente
c. similar
d. a o b

34. Dado un nodo para insertar en una lista vinculada, cuando el nodo predecesor tiene un apuntador nulo, entonces usted lo estará agregando a _____.

a. una lista vacía
b. el inicio de la lista
c. el final de la lista
d. a o b

35. Dado un nodo para insertar en una lista ligada, cuando el nodo predecesor tiene un apuntador no nulo, entonces usted lo estará agregando a _____.

a. el inicio de la lista
b. el cuerpo de la lista
c. el final de la lista
d. a o b

36. Para cruzar una lista, se necesita un apuntador _____.

a. nulo
b. auxiliar
c. inicial
d. de inserción

EJERCICIOS

37. Usted tiene dos arreglos, A y B, cada uno de diez enteros. Escriba un algoritmo que compruebe si cada elemento del arreglo A es igual al elemento correspondiente del arreglo B.

38. Escriba un algoritmo que invierta los elementos de un arreglo de manera que el último elemento se convierta en el primero, el antepenúltimo en el segundo, y así en adelante.

39. Escriba un algoritmo para imprimir el contenido de un arreglo bidimensional de *I* filas y *J* columnas.

40. Utilizando la figura 11.9, ¿cuál es el resultado de las siguientes instrucciones, suponiendo que `Fr1` es del tipo fracción?

```
Fr1.numerador = 7;
Fr1.denominador = 8;
```

41. Usando la figura 11.9, escriba un algoritmo para sumar una fracción (`Fr1`) a otra fracción (`Fr2`).

42. Utilizando la figura 11.9, escriba un algoritmo para restar una fracción (`Fr1`) de otra fracción (`Fr2`).

43. Usando la figura 11.9, escriba un algoritmo para multiplicar una fracción (`Fr1`) por otra fracción (`Fr2`).

44. Usando la figura 11.9, escriba un algoritmo para dividir una fracción (`Fr1`) entre otra fracción (`Fr2`).

45. Usted puede tener un arreglo de registros. Muestre gráficamente un arreglo de fracciones (utilice la figura 11.9).

46. Puede tener una lista ligada en la cual la parte de los datos sea un registro. Muestre gráficamente una lista ligada de registros de estudiantes (utilice la figura 11.9).

47. En la mayoría de los lenguajes de programación, un arreglo es una estructura de datos estática. Cuando usted define un arreglo, el tamaño es fijo. ¿Qué problema creará esta restricción?

48. Una lista ligada es una estructura de datos dinámica. El tamaño de una lista ligada puede modificarse de manera dinámica (durante la ejecución del programa). ¿De qué forma esta característica beneficia al programador?

49. ¿Cuál operación cree usted que es más sencilla, agregar un elemento a un arreglo o agregar un elemento a una lista ligada? Justifique su respuesta.

50. ¿Cuál operación cree usted que es más sencilla, eliminar un elemento de un arreglo o eliminar un elemento de una lista ligada? Justifique su respuesta.

51. ¿Cuál operación considera que es más sencilla, acceder a un elemento de un arreglo o acceder a un elemento de una lista ligada? Justifique su respuesta.

52. ¿Cuál operación piensa usted que es más fácil, realizar una búsqueda en un arreglo o en una lista ligada? Justifique su respuesta.

53. ¿Cuál operación cree usted que es más sencilla, ordenar elementos en un arreglo u ordenar una lista ligada? Justifique su respuesta.

12 Tipos de datos abstractos

Comenzamos este capítulo con un breve resumen de los antecedentes del tipo de datos abstracto (TDA). Enseguida damos una definición y proponemos un modelo. El resto del capítulo estudia diversos TDA, tales como la lista lineal, la pila, la cola de espera, el árbol, el árbol binario y el grafo.

12.1 ANTECEDENTES

En los primeros días de la programación no había tipos de datos abstractos. Si queríamos leer un archivo, escribíamos el código para leer el dispositivo de archivo físico. No tomaba mucho tiempo darse cuenta de que el mismo código se escribía una y otra vez. Así que los científicos de la computación crearon lo que se conoce ahora como **tipo de datos abstracto (TDA)**. Escribimos el código para leer un archivo y luego lo colocamos en una biblioteca para uso de todos los programadores.

Este concepto se encuentra en los lenguajes modernos de hoy en día. El código para leer el teclado es un TDA. Tiene una estructura de datos, carácter y una serie de operaciones que pueden usarse para leer esa estructura de datos. Las reglas permiten no sólo leer la estructura de datos sino además convertirla en diferentes estructuras de datos tales como enteros y cadenas.

Con un TDA, a los usuarios no les preocupa *cómo* se realiza la tarea sino *qué* puede hacer ésta. En otras palabras, el TDA consiste en una serie de definiciones que permiten a los programadores utilizar las funciones mientras se oculta la implementación. Esta generalización de operaciones con implementaciones inespecíficas se conoce como **abstracción**. Usted abstrae la esencia del proceso y deja ocultos los detalles de la implementación.

El concepto de abstracción significa:
1. **Usted sabe qué puede hacer un tipo de datos.**
2. **La manera de hacerlo queda oculta.**

Como ejemplo, considere a un analista de sistemas que debe simular la fila de espera de un banco para determinar cuántos cajeros se necesitan para atender a los clientes de manera eficiente. Este análisis requiere la simulación de una **cola de espera**. Sin embargo, las colas de espera por lo general no están disponibles en los lenguajes de programación. Incluso si un tipo de cola de espera está disponible, el analista aún necesitaría algunas operaciones básicas, como la inserción (*enqueuing*) y la eliminación (*dequeuing*) de un elemento, para la **simulación de la cola de espera**.

Hay dos soluciones posibles a este problema: (1) se puede escribir un programa que simule la cola de espera que el analista necesita (en este caso, la solución es buena sólo para una aplicación próxima) o (2) se puede escribir un TDA de cola de espera que se utilice para resolver cualquier problema de cola de espera. Si se elige esta última opción, el analista todavía necesitará escribir un programa para simular la aplicación bancaria, pero hacerlo será mucho más fácil y rápido debido a que él o ella podrán concentrarse en la aplicación en vez de concentrarse en la cola de espera.

DEFINICIÓN

Definamos formalmente un TDA. Un tipo de datos abstracto es una declaración de datos empacados junto con las operaciones que tienen sentido para el tipo de datos. Después se encapsulan los datos y las operaciones que se aplican a los datos, y se ocultan al usuario.

Tipo de datos abstracto
1. Declaración de los datos
2. Declaración de las operaciones
3. Encapsulamiento de los datos y operaciones

No podemos exagerar la importancia de ocultar la implementación. El usuario no tiene que conocer la **estructura de datos** para usar el TDA. Respecto al ejemplo de cola de espera, el programa de aplicación no debe tener conocimiento de la estructura de datos. Todas las referencias a los datos y la manipulación de los mismos en la cola de espera debe manejarse a través de interfaces definidas para la estructura. Permitir que el programa de aplicación haga referencia directamente a la estructura de datos es una falta común en muchas implementaciones que impide que el TDA sea completamente portátil a otras aplicaciones.

MODELO PARA UN TIPO DE DATOS ABSTRACTO

El modelo TDA aparece en la figura 12.1. El área sombreada con un contorno irregular representa el modelo. Dentro del área abstracta hay dos aspectos del modelo: la estructura de datos y las funciones operativas. Ambas están totalmente contenidas en el modelo y no dentro del ámbito del usuario. No obstante, la estructura de datos está disponible para todas las operaciones con TDA según se requiera y una operación puede llamar a otras funciones para cumplir su tarea. Dicho de otra forma, las estructuras de datos y las funciones están dentro del ámbito de la una y de las otras.

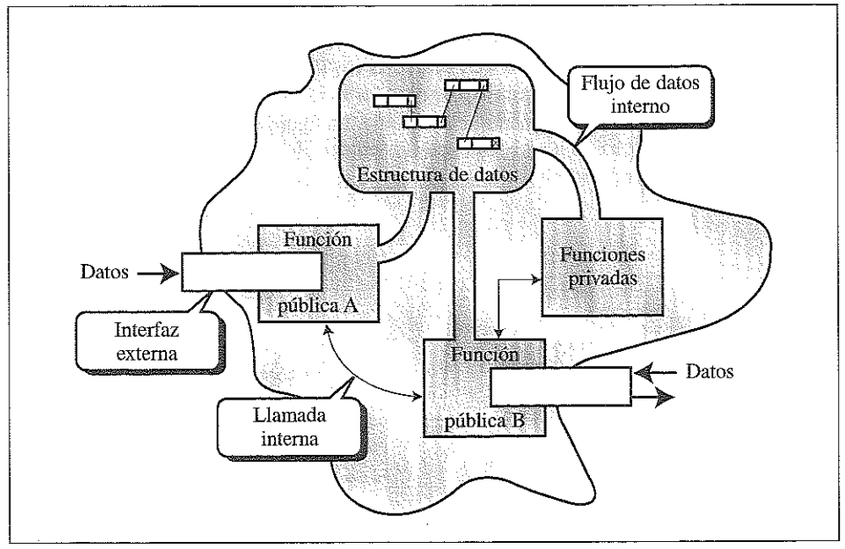


Figura 12.1 Modelo para TDA

OPERACIONES CON TDA

Los datos se introducen, se accede a ellos, se modifican y se eliminan mediante las interfaces operativas dibujadas como rectángulos parcialmente dentro y parcialmente fuera de la estructura. Para el encabezado (*header*) de cada operación, hay un algoritmo que realiza una operación específica. Sólo el nombre de la operación y sus parámetros están visibles para el usuario y proporcionan la única interfaz para el TDA. Operaciones adicionales pueden crearse para cumplir requisitos específicos.

12.2 LISTAS LINEALES

Primero definiremos unos cuantos TDA comenzando por la lista lineal. Una **lista lineal** es una lista en la cual cada elemento tiene un sucesor único. En otras palabras, una lista lineal tiene una estructura secuencial. La propiedad de secuencia de una lista lineal se muestra en el diagrama de la figura 12.2.

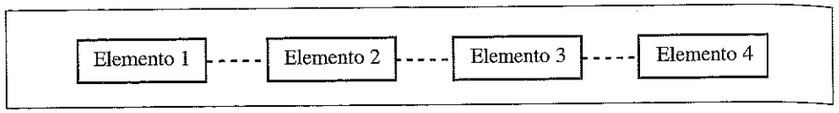


Figura 12.2 Lista lineal

Las listas lineales pueden dividirse en dos categorías: generales y restringidas. En una **lista general**, los datos pueden insertarse y eliminarse en cualquier parte, y no hay restricciones sobre las operaciones que pueden utilizarse para procesar la lista. Las estructuras generales pueden describirse posteriormente mediante sus datos, ya sea como listas aleatorias u ordenadas. En una **lista aleatoria** no hay un orden en los datos. En una **lista ordenada** los datos se acomodan según una clave. Una clave es uno o más campos dentro de una estructura que se utilizan para identificar los datos o para controlar de algún otro modo su uso. En un arreglo simple, los datos también son las claves.

En una **lista restringida**, los datos sólo pueden añadirse o eliminarse en los extremos de la estructura, y el procesamiento está restringido a operaciones de los datos en los extremos de la lista. Describimos dos estructuras de lista restringida: la lista **primero en entrar, primero en salir (FIFO:** *first in, first out*) y la lista **último en entrar, primero en salir (LIFO:** *last in, first out*). La lista FIFO por lo general se llama cola de espera; la lista LIFO comúnmente se llama pila. La figura 12.3 muestra los tipos de listas lineales.

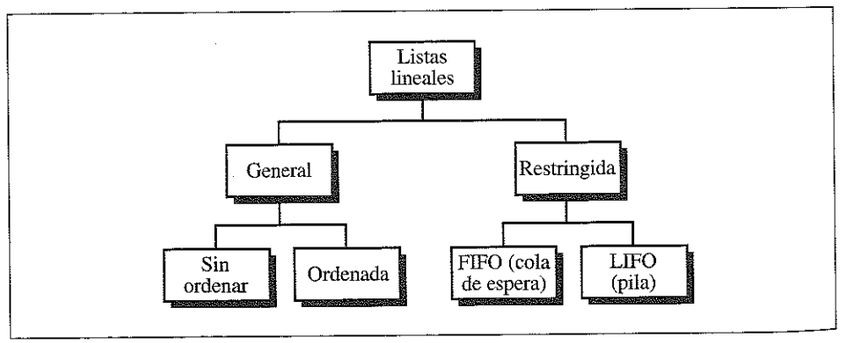


Figura 12.3 Categorías de listas lineales

En esta sección nos concentramos en un tipo de lista lineal: la lista lineal general con datos ordenados. Las listas lineales restringidas se analizarán posteriormente en el capítulo.

OPERACIONES CON LISTAS LINEALES

Aun cuando podemos definir muchas operaciones con una lista lineal general, sólo analizaremos cuatro operaciones de las más comunes: la inserción, la eliminación, la recuperación y el recorrido.

Inserción

Los datos deben insertarse en listas ordenadas de modo que se mantenga el orden de la lista. Para mantener el orden se requiere insertar los datos al principio o al final de la lista, pero la mayor parte del tiempo los datos se insertan en algún lugar dentro de la lista. Para determinar dónde se van a colocar los datos, los científicos de la computación utilizan un algoritmo de búsqueda. El único problema posible con esta simple operación es que ya no haya espacio para el elemento nuevo. Si no hay espacio suficiente, la lista entra en un estado de desbordamiento y el elemento no puede añadirse. La inserción se muestra gráficamente en la figura 12.4. Los datos insertados se identifican mediante el elemento sombreado, en este caso, el tercer elemento de la lista revisada.

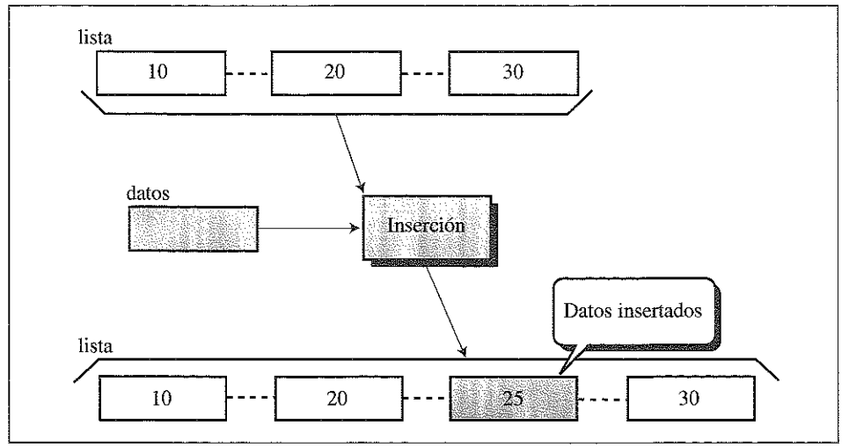


Figura 12.4 Inserción en una lista lineal

Eliminación

La eliminación de un elemento de una lista general (figura 12.5) requiere que se realice una búsqueda en la lista para localizar los datos que se van a eliminar. Cualquier algoritmo de búsqueda secuencial puede emplearse para localizar los datos. Una vez localizados, los datos se eliminan de la lista. El único problema posible con esta simple operación es la lista vacía. Si la lista está vacía, ésta se encuentra en un estado de **sobre desbordamiento** y la operación fracasa.

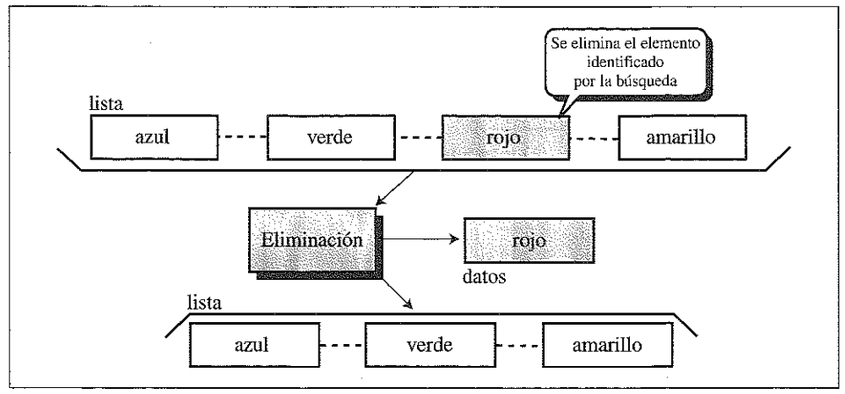


Figura 12.5 Eliminación de una lista lineal

Recuperación

La **recuperación** de lista requiere que los datos se localicen en una lista (figura 12.6). Una copia de los datos debe recuperarse sin cambiar el contenido de la lista. Al igual que sucede con la inserción y la eliminación, cualquier algoritmo de búsqueda secuencial puede utilizarse para localizar los datos a recuperar de una lista general. El único problema posible con esta simple operación es una lista vacía. No se puede recuperar un elemento de una lista lineal vacía.

Recorrido

El recorrido de una lista es una operación en la cual todos los elementos en la lista se procesan en forma secuencial, uno a uno (figura 12.7). En esta figura, la variable llamada Walker apunta al elemento que debe procesarse. El procesamiento aquí puede ser la recuperación, el almacenamiento, etc. El recorrido de listas requiere un algoritmo iterativo en vez de una búsqueda. Cada iteración de un ciclo procesa un elemento en la lista. El ciclo termina cuando todos los elementos se han procesado.

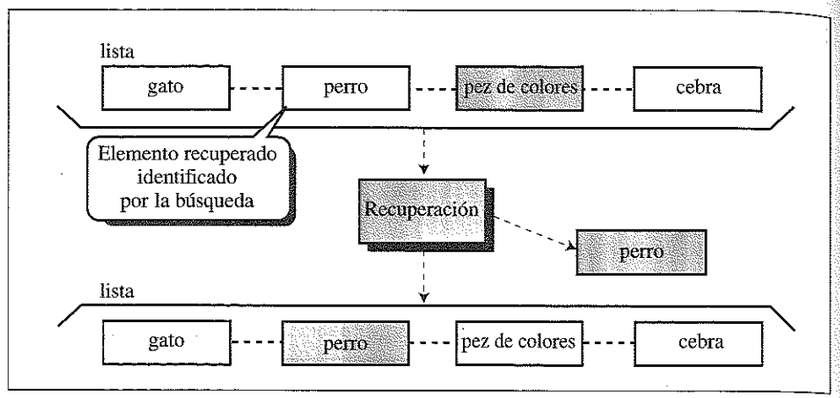


Figura 12.6 Recuperación desde una lista lineal

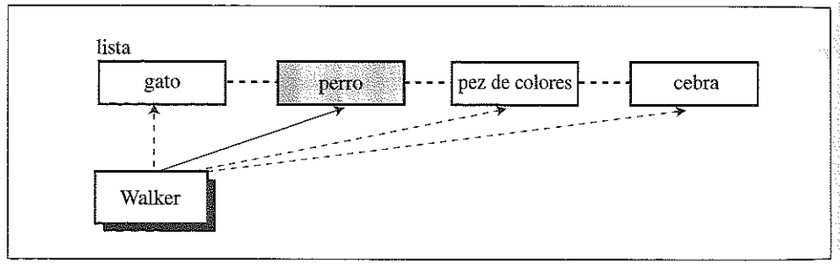


Figura 12.7 Recorrido de una lista lineal

IMPLEMENTACIÓN DE UNA LISTA LINEAL GENERAL

Dos métodos comunes de implementación de una lista lineal general son un **arreglo** y una **lista ligada**.

APLICACIONES DE LISTA LINEAL

Las listas lineales se utilizan en situaciones donde se accede a los elementos en forma aleatoria. Por ejemplo, en una universidad una lista lineal puede utilizarse para almacenar información sobre los estudiantes que están inscritos en cada semestre. La información puede accederse en forma aleatoria.

12.3 PILAS

Una **pila** es una lista lineal restringida en la cual las adiciones y eliminaciones se realizan en un extremo llamado cima. Si usted inserta una serie de datos en una pila y luego la elimina, el orden de los datos se invertirá. La entrada de datos como 5, 10, 15, 20 debe eliminarse como 20, 15, 10 y 5. Este atributo de inversión es la razón por la cual las pilas se conocen como una estructura de datos último en entrar, primero en salir (LIFO).

Una persona utiliza en su vida diaria muchos tipos de pila distintos. A menudo hablamos de una pila de monedas o de una pila de platos. Cualquier situación en la cual usted simplemente pueda añadir o eliminar un objeto en la cima es una pila. Si desea eliminar cualquier

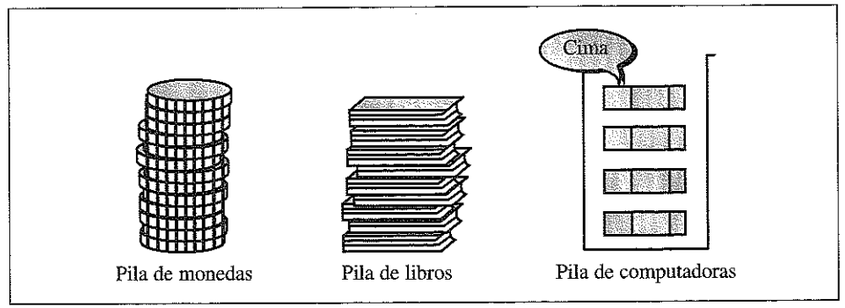


Figura 12.8 Tres representaciones de una pila

objeto diferente del que está en la cima, primero debe eliminar todos los objetos que están debajo de él. En la figura 12.8 se muestran tres representaciones de una pila.

OPERACIONES CON PILAS

Aunque podemos definir muchas operaciones para una pila, hay tres que son básicas: insertar, extraer y vaciar.

Insertar

Al **insertar** (*push*) se añade un elemento en la parte superior de la pila (figura 12.9). Después de insertar, el elemento se vuelve la cima. El único problema posible con esta operación simple es quedarse sin espacio para el nuevo elemento. Si no hay espacio suficiente, la pila está en un estado de desbordamiento y el elemento no puede añadirse.

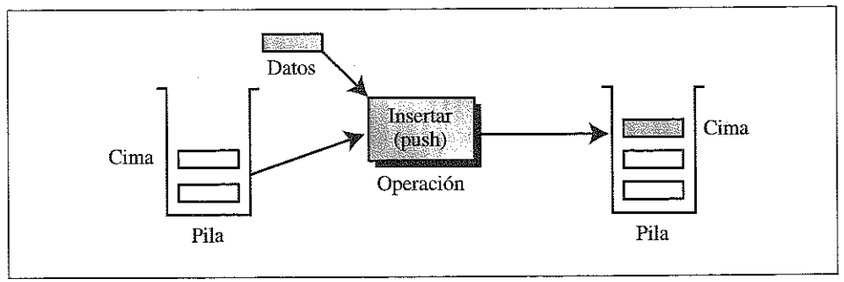


Figura 12.9 Operación de insertar en una pila

Extraer

Cuando usted **extrae** (*pop*) un elemento de una pila, elimina el elemento en la parte superior de la pila y lo devuelve al usuario (figura 12.10). Cuando el último elemento en la pila se elimina, la pila debe establecerse a su estado vacío. Si se hace una llamada a la operación de **extraer** cuando la pila está vacía, ésta se encuentra en un estado de sobre desbordamiento.

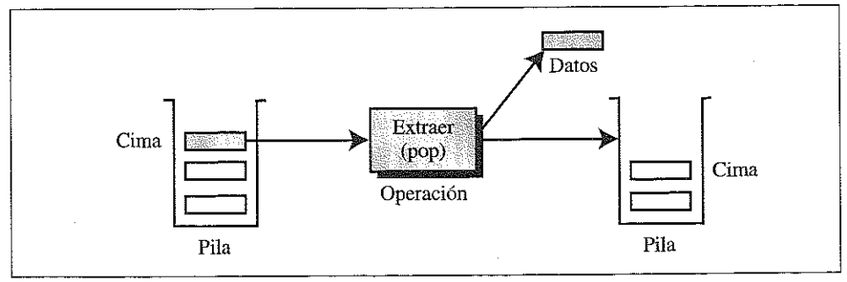


Figura 12.10 Operación de extraer de una pila

Vaciar

Esta operación (*empty*) hace una revisión para ver si una pila está vacía o no. La respuesta es ya sea verdadera o falsa.

EJEMPLO 1

Muestre el resultado de las siguientes operaciones en una pila S.

```
push (S, 10)
push (S, 12)
push (S, 8)
if not empty (S), then pop (S)
push (S, 2)
```

SOLUCIÓN

La figura 12.11 muestra las operaciones y el resultado final.

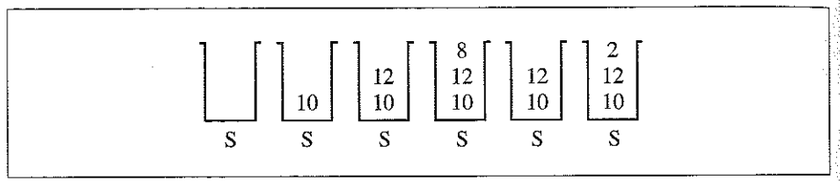


Figura 12.11 Ejemplo 1 ■

IMPLEMENTACIÓN DE UNA PILA

Aunque una pila puede implementarse ya sea como un arreglo o como una lista ligada, la forma más común es una lista ligada debido a que las operaciones de extracción e inserción pueden implementarse mucho más fácilmente en una lista de este tipo.

APLICACIONES DE PILA

Las aplicaciones de pila pueden clasificarse en cuatro amplias categorías: inversión de datos, análisis sintáctico de datos, postergación del uso de los datos y retroceso sobre los pasos.

Inversión de datos

La inversión de datos requiere que un conjunto dado de datos se reordene de modo que el primer elemento y el último se intercambien, con todas las posiciones entre el primero y el último también intercambiadas relativamente. Por ejemplo, 1 2 3 4 se convierte en 4 3 2 1.

Análisis sintáctico

Otra aplicación de pilas es el **análisis sintáctico**. Este tipo de análisis es cualquier lógica que divide los datos en piezas independientes para su procesamiento posterior. Por ejemplo, para traducir un programa fuente a lenguaje de máquina, un compilador debe analizar el programa en partes individuales como palabras clave, nombres y elementos sintácticos (*tokens*).

Un problema de programación común son los paréntesis sin par en una expresión algebraica. Cuando los paréntesis no tienen par pueden ocurrir dos tipos de errores: pueden faltar ya sea el paréntesis de apertura o el paréntesis de cierre. Siempre que se encuentra un paréntesis de apertura, éste se inserta en la pila. Cuando se encuentra un paréntesis de cierre, un paréntesis de apertura (de la cima de la pila) se extrae y se descarta. Si al final la pila no está vacía, significa que hay más paréntesis de apertura que de cierre. También ocurre un error cuando se encuentra un paréntesis de cierre y no hay paréntesis de apertura en la cima de la fila.

Postergación

Cuando se utiliza una pila para invertir una lista, toda la lista es leída antes de que los resultados se comiencen a generar. Con frecuencia, la lógica de una aplicación requiere que el uso de los datos se postergue hasta un momento posterior. Una pila puede ser útil cuando la aplicación requiere la postergación del uso de los datos.

Retroceso

El **retroceso,** dar marcha atrás a los datos previos, es un uso de pila que se encuentra en aplicaciones como los juegos de computadoras, el análisis de decisiones y los sistemas expertos.

12.4 COLAS DE ESPERA

Una cola de espera es una lista lineal en la cual los datos sólo pueden insertarse en un extremo, llamado extremo trasero, y eliminarse en el otro extremo, llamado frente. Estas restricciones aseguran que los datos se procesen a través de la cola de espera en el orden en el cual se reciben. En otras palabras, una cola de espera es una estructura de primero en entrar, primero en salir (FIFO).

Una cola de espera es lo mismo que una fila. De hecho, si usted estuviera en Inglaterra, no entraría en una fila sino en una cola de espera. Una fila de gente esperando el autobús en una estación es una cola de espera; una lista de llamadas que se ponen en espera para que les responda un operador telefónico es una cola de espera, y una lista de tareas que esperan ser procesadas por una computadora también es una cola de espera.

La figura 12.12 exhibe dos representaciones de una cola de espera: la primera, una cola de espera de personas y la otra, una cola de espera de computadora. Tanto las personas como los datos entran a la cola de espera en la parte trasera y avanzan por la misma hasta que llegan al frente. Una vez que están en la parte de adelante, dejan la cola de espera y son atendidos.

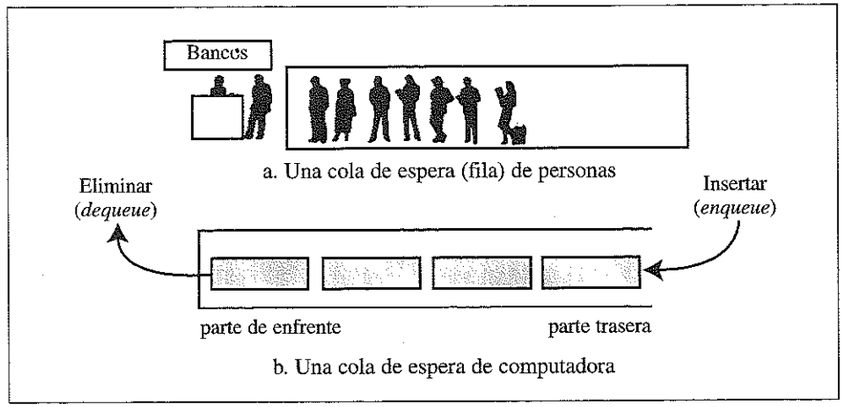


Figura 12.12 Representaciones de colas de espera

OPERACIONES CON COLAS DE ESPERA

Aun cuando sea posible definir muchas operaciones para una cola de espera, tres son las básicas: eliminar, insertar y vaciar.

Insertar

La operación de **insertar** (*enqueue*) en una cola de espera aparece en la figura 12.13. Después de que los datos se han insertado en la cola, el nuevo elemento se vuelve la parte trasera. Como se vio con las pilas, el único problema posible con la inserción es quedarse sin espacio para los datos. Si no hay suficiente espacio para otro elemento en la cola de espera, ésta entra en un estado de desbordamiento.

Eliminar

La operación de **eliminar** (*dequeue*) un elemento de una cola de espera se muestra en la figura 12.14. Los datos en la parte de enfrente de la cola de espera se eliminan de la misma y se devuelven al usuario. Si no hay datos en la cola de espera cuando se intenta una eliminación, la cola está en un estado de sobre desbordamiento.

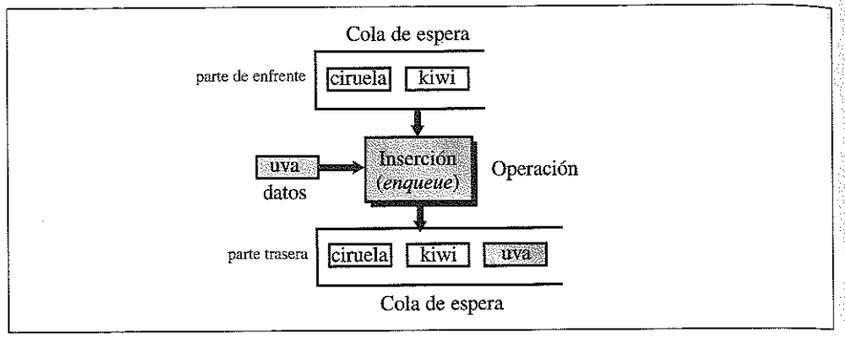


Figura 12.13 Operación de insertar

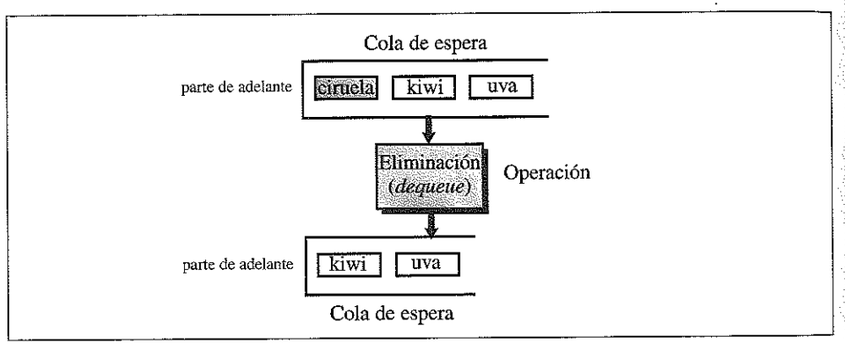


Figura 12.14 Operación de eliminar

Vaciar

Esta operación (*empty*) hace una revisión para ver si una cola de espera está vacía o no. El resultado es verdadero o falso.

EJEMPLO 2

Muestre el resultado de las siguientes operaciones en una cola de espera Q.

```
enqueue (Q, 23)
if not empty (Q), dequeue (Q)
enqueue (Q, 20)
enqueue (Q, 19)
if not empty (Q), dequeue (Q)
```

SOLUCIÓN

La figura 12.15 muestra las operaciones y el resultado final.

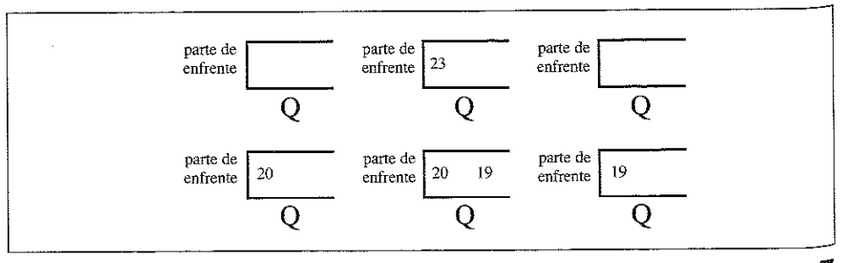


Figura 12.15 Ejemplo 2 ■

IMPLEMENTACIÓN DE UNA COLA DE ESPERA

Una cola de espera puede implementarse ya sea como un arreglo o como una lista ligada.

APLICACIONES DE LA COLA DE ESPERA

Las colas de espera son una de las estructuras de procesamiento de datos más comunes. Se encuentran en prácticamente todo sistema operativo y red en incontables áreas distintas. Por ejemplo, las colas de espera se utilizan en aplicaciones de negocios en línea tales como el procesamiento de solicitudes del cliente, tareas y pedidos. En un sistema de cómputo una cola de espera es necesaria para procesar tareas y para servicios del sistema tales como los servicios de impresión.

Las colas de espera pueden volverse bastante complejas; damos una implementación simple de cola de espera: una aplicación que es útil para clasificar datos. Una cola de espera preserva el orden de los datos. Por ejemplo, usted tiene una lista de números y necesita clasificarlos en grupos (100 o menos, entre 100 y 201, etc.). Puede leer los datos y crear varias colas de espera. Un número se inserta en la cola de espera apropiada y puede recuperarse en el orden en que se leyó pero en su propio grupo. Por ejemplo, todos los números en la categoría de 100 o menos pueden imprimirse primero en el orden en que se leyeron. Después, todos los números entre 100 y 201 pueden imprimirse en el orden en que se leyeron y así por el estilo.

12.5 ÁRBOLES

Los árboles se usan exhaustivamente en las ciencias de la computación como una estructura eficiente para realizar búsquedas en listas dinámicas grandes y para aplicaciones diversas como los sistemas de inteligencia artificial y los algoritmos de codificación. En esta sección analizamos el concepto básico de árbol. En la siguiente sección presentamos un tipo especial de árbol, llamado árbol binario, el cual es una estructura común en las ciencias de la computación.

CONCEPTOS BÁSICOS DE ÁRBOL

Un **árbol** consiste de un conjunto finito de elementos, llamados **nodos,** y un conjunto finito de líneas dirigidas, llamadas **ramas**, que conectan los nodos. El número de ramas asociadas con un nodo es el **grado** del nodo. Cuando la rama se dirige hacia el nodo, es una rama de **grado de entrada**, cuando la rama se aleja del nodo, es una rama **de grado de salida.** La suma de las ramas de grado de entrada y de grado de salida es el grado del nodo. En la figura 12.16 el grado del nodo B es 3.

Si el árbol no está vacío, el primer nodo se conoce como **raíz.** El grado de entrada de la raíz es, por definición, cero. Con excepción de la raíz todos los nodos en un árbol deben tener un grado de entrada de exactamente uno. Todos los nodos en el árbol pueden tener cero, una o más ramas que salen de ellos, es decir, pueden tener un grado de salida de cero, uno o más.

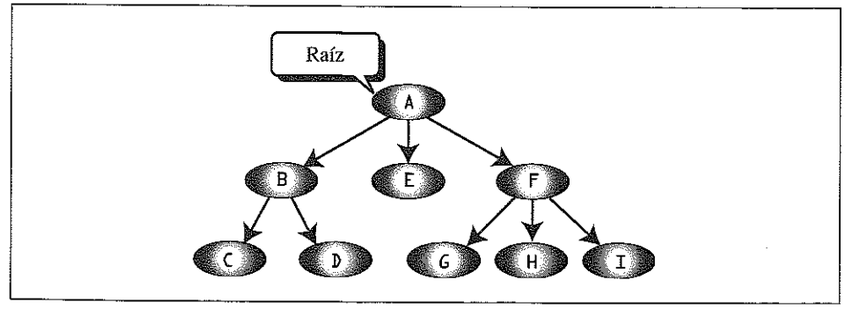


Figura 12.16 Representación de un árbol

Terminología

Además de la *raíz*, muchos términos distintos se utilizan para describir los atributos de un árbol. Una **hoja** es cualquier nodo con un grado de salida de cero. En la figura 12.16, el nodo E es una hoja. Un nodo que no es una raíz o una hoja se conoce como **nodo interno** debido a que se encuentra en la parte media de un árbol. En la figura 12.16 el nodo B es un nodo interno.

Un nodo es un **padre** si tiene nodos sucesores –es decir, si tiene un grado de salida mayor que cero. A la inversa, un nodo con un predecesor es un **hijo**. Un nodo hijo tiene un grado de entrada de uno. Dos o más nodos con el mismo padre son **hermanos**. Por fortuna, usted no tiene que preocuparse por las tías, los tíos, las sobrinas, los sobrinos y los primos. Aunque algunos libros utilizan el término *abuelo*, nosotros no lo hacemos; preferimos el término más general de *ancestro*. Un **ancestro** es cualquier nodo en el camino desde la raíz hasta el nodo. Un **descendiente** es cualquier nodo en el camino debajo del nodo padre; es decir, todos los nodos en los caminos de un nodo dado hacia una hoja son descendientes del nodo. La figura 12.17 muestra el uso de estos términos.

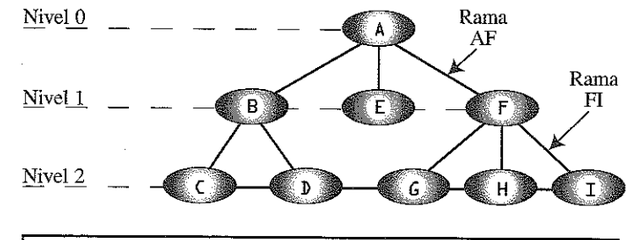


Figura 12.17 Terminología de árbol

Varios términos tomados de las matemáticas o creados por científicos de la computación describen los atributos de los árboles y sus nodos. Un **camino** es una secuencia de nodos en la cual cada nodo es adyacente al siguiente. Se puede llegar a cada nodo en el árbol al seguir un camino único que principia en la raíz. En la figura 12.17 el camino de la raíz a la hoja I está designada como AFI. Incluye dos ramas distintas: AF y FI.

El **nivel** de un nodo es su distancia desde la raíz. Debido a que la raíz tiene una distancia cero desde sí misma, la raíz está en el nivel 0. Los hijos de la raíz están en el nivel 1 y los hijos de éstos, en el nivel 2, y así sucesivamente. Observe la relación entre los niveles y los hermanos de la figura 12.17. Los hermanos siempre están en el mismo nivel, pero no necesariamente todos los nodos en un nivel son hermanos. Por ejemplo, en el nivel 2, C y D son hermanos al igual que G, H e I. Sin embargo, D y G no son hermanos porque tienen diferentes padres.

La **altura** de un árbol es el nivel de la hoja en el camino más largo desde la raíz más 1. Por definición, la altura de un árbol vacío es –1. La figura 12.17 contiene nodos en tres niveles: 0, 1 y 2. Su altura es 3. Debido a que el árbol se dibuja de arriba hacia abajo, algunos textos se refieren a la **profundidad** de un árbol en vez de a su altura.

Un árbol puede dividirse en subárboles. Un subárbol es cualquier estructura conectada debajo de la raíz. El primer nodo en un subárbol se conoce como la raíz del subárbol y se utiliza para nombrar al subárbol. Además, cada subárbol puede dividirse a su vez en subárboles. En la figura 12.18 BDC es un subárbol, como E y FGHI. Observe que según esta definición, un nodo individual es un subárbol. De esta manera, el subárbol B puede dividirse en dos subárboles, C y D, y el subárbol F contiene los subárboles G, H e I.

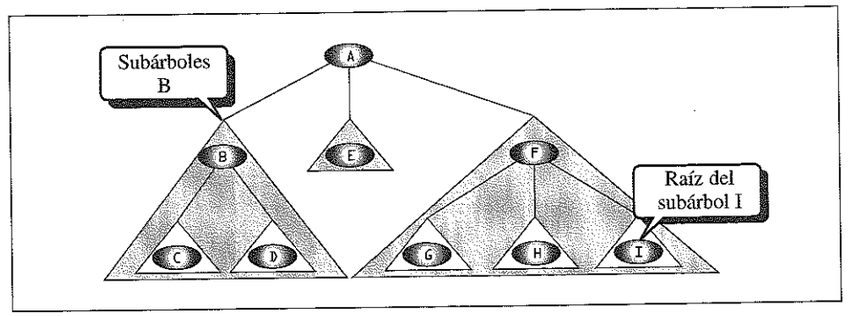


Figura 12.18 Subárboles

OPERACIONES CON ÁRBOLES

Las operaciones con un árbol son complejas y su explicación está más allá del ámbito de este libro. Se analizan en un curso de estructura de datos.

12.6 ÁRBOLES BINARIOS

Un **árbol binario** es un árbol en el cual ningún nodo puede tener más de dos subárboles. En otras palabras, un nodo puede tener cero, uno o dos subárboles. Estos subárboles están diseñados como el subárbol izquierdo y el subárbol derecho. La figura 12.19 muestra un árbol binario con sus dos subárboles. Observe que cada subárbol es por sí mismo un árbol binario.

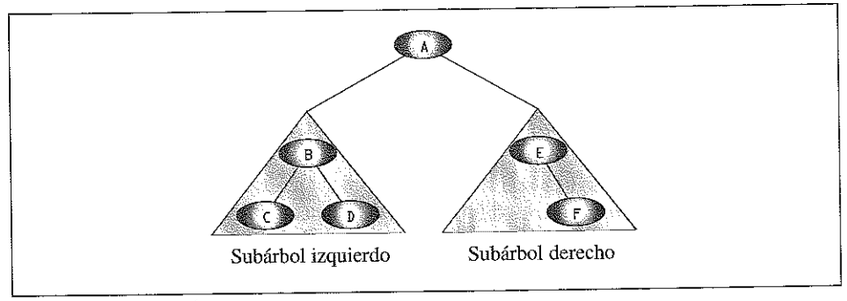


Figura 12.19 Árbol binario

Propiedades

Para comprender mejor la estructura de los árboles binarios, estudie la figura 12.20. Ésta contiene ocho árboles, el primero de los cuales es un árbol nulo. Un **árbol nulo** es un árbol que no tiene nodos (figura 12.20[a]). A medida que estudie esta figura, notará que la simetría no es requisito de un árbol.

Ahora definimos varias propiedades para los árboles binarios que los distinguen de los árboles generales.

Altura de árboles binarios La altura de los árboles binarios puede predecirse matemáticamente. Dado que usted necesita almacenar N nodos en un árbol binario, la altura máxima, $H_{máx}$, puede definirse como sigue:

$$H_{máx} = N$$

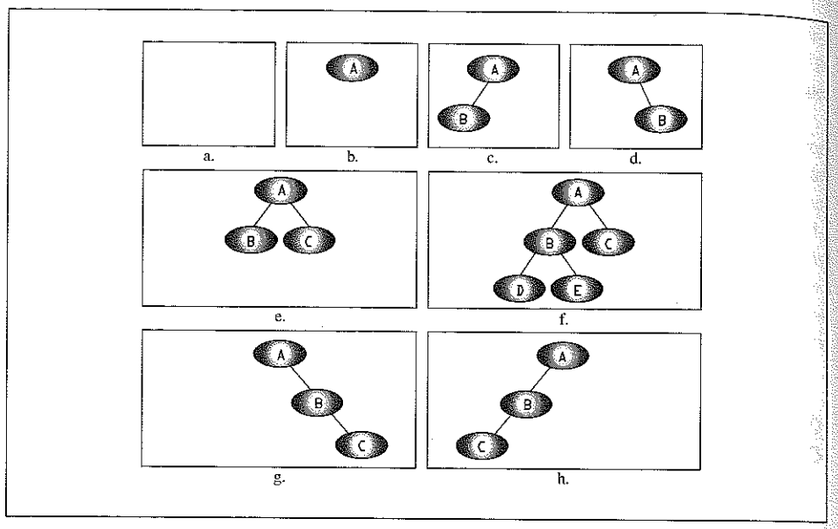


Figura 12.20 Ejemplos de árboles binarios

Un árbol con una altura máxima es raro. Ocurre cuando todo el árbol se construye en una dirección, como se muestra en las figuras 12.20(g) y 12.20(h). La altura mínima del árbol, $H_{mín}$, se determina mediante la fórmula siguiente:

$$H_{mín} = [\log_2 N] + 1$$

Dada la altura de un árbol binario, H, el número mínimo y el número máximo de nodos en el árbol se dan como:

$$N_{mín} = H$$

$$N_{máx} = 2^H - 1$$

Equilibrio La distancia de un nodo desde la raíz determina qué tan eficientemente puede localizarse. Por ejemplo, los hijos de cualquier nodo en un árbol pueden accederse siguiendo sólo uno camino de rama, aquel que conduce al nodo deseado. Por ejemplo, es posible acceder a los nodos en el nivel 2 de un árbol únicamente siguiendo dos ramas desde la raíz. Por lo tanto, es lógico que entre más corto sea el árbol, más fácil será localizar cualquier nodo deseado en el árbol.

Este concepto nos conduce a una característica muy importante de un árbol binario: su **equilibrio**. Para determinar si un árbol está en equilibrio, calcule su factor de equilibrio. El **factor de equilibrio** de un árbol binario es la diferencia en altura entre sus subárboles izquierdo y derecho. Si se define la altura del subárbol izquierdo como H_I y la altura del subárbol derecho como H_D, entonces el factor de equilibrio del árbol, B, está determinado por la siguiente fórmula:

$$B = H_I - H_D$$

Utilizando esta fórmula, los equilibrios de los ocho árboles en la figura 12.20 son (a) 0 por definición, (b) 0, (c) 1, (d) −1, (e) 0, (f) 1, (g) −2 y (h) 2.

Un árbol está equilibrado si su factor de equilibrio es 0 y sus subárboles también están equilibrados. Debido a que esta definición ocurre rara vez, una definición alternativa se aplica de una manera más general: Un árbol binario está equilibrado si la altura de sus subárboles difiere por no más de 1 (su factor de equilibrio es −1, 0 o +1) y sus subárboles también están equilibrados. Esta definición fue creada por Adelson-Velskii y Landis en su definición de un árbol AVL.

OPERACIONES CON ÁRBOLES BINARIOS

Las tres operaciones más comunes definidas para un árbol binario son insertar, eliminar y recorrer. Las operaciones de inserción y eliminación son complejas y su explicación está más allá del ámbito de este libro. Analizamos el recorrido de árbol binario en esta sección.

Recorridos de árbol binario

Un **recorrido de árbol binario** requiere que cada nodo del árbol se procese una vez y sólo una vez en una secuencia predeterminada. Los dos métodos generales para la secuencia de recorrido son primero en profundidad y primero en anchura.

Recorrido primero en profundidad Dado que un árbol binario consiste en una raíz, un subárbol izquierdo y un subárbol derecho, podemos definir seis secuencias diferentes de **recorrido primero en profundidad**. Los científicos de la computación han asignado a tres de estas secuencias nombres estándar en la literatura; los otros tres no tienen nombre pero se derivan fácilmente. Los recorridos estándar se muestran en la figura 12.21.

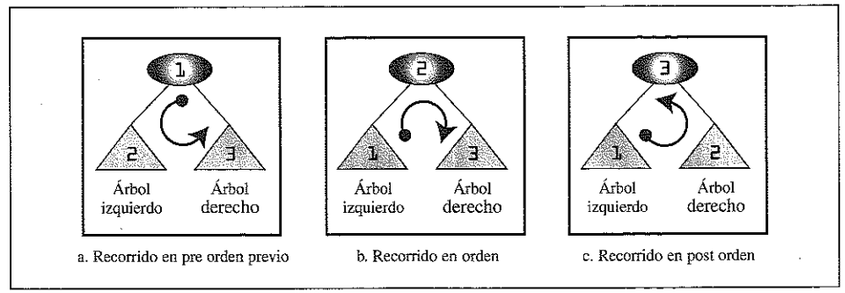


Figura 12.21 Recorrido de primero en profundidad de un árbol binario

La designación tradicional de los recorridos utiliza una designación de `node` (N) para la raíz, `left` (L) para el subárbol izquierdo y `right` (R) para el subárbol derecho.

- **Recorrido en pre orden (NLR).** En el **recorrido en pre orden**, el nodo raíz se procesa primero, seguido por el subárbol izquierdo y luego por el subárbol derecho. Toma su nombre del prefijo en latín *pre*, el cual significa "va antes de". Así, la raíz va antes que los subárboles. La figura 12.22 muestra otra forma de visualizar el recorrido del árbol. Imagine que está caminando alrededor de un árbol, comenzando por la izquierda de la raíz y manteniéndose tan cerca de los nodos como le es posible. En el recorrido de orden, usted procesa el nodo cuando está a la izquierda del mismo. Esto aparece como una caja negra a la izquierda del nodo. El camino se muestra como una línea que sigue una trayectoria completamente alrededor del árbol y de regreso a la raíz.

- **Recorrido en orden (LNR).** El **recorrido en orden** primero procesa el subárbol izquierdo, luego el nodo raíz y finalmente el subárbol derecho. El significado de la palabra *en* es que la raíz se procesa "dentro" de los subárboles. La figura 12.23 muestra otra forma de visualizar el recorrido de un árbol. Imagine que está caminando alrededor de un árbol, comenzando por la izquierda de la raíz y manteniéndose tan cerca de los nodos como le es posible. En el recorrido en orden, usted procesa el nodo cuando está debajo de él. Esto aparece como una caja negra bajo el nodo. El camino se muestra como una línea que sigue una trayectoria completamente alrededor del árbol y de regreso a la raíz.

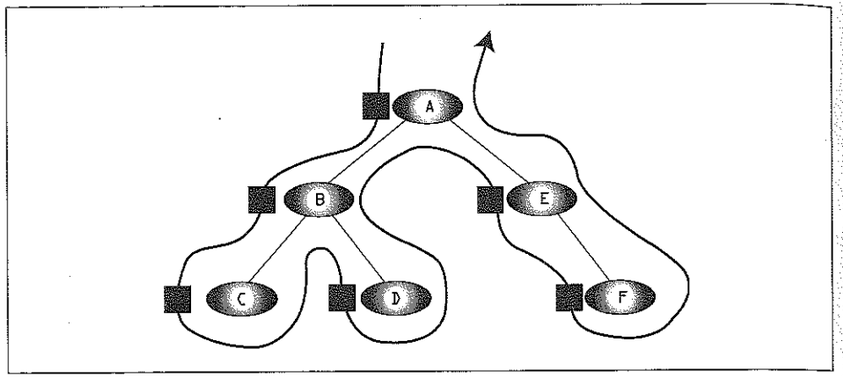


Figura 12.22 Recorrido en pre orden de un árbol binario

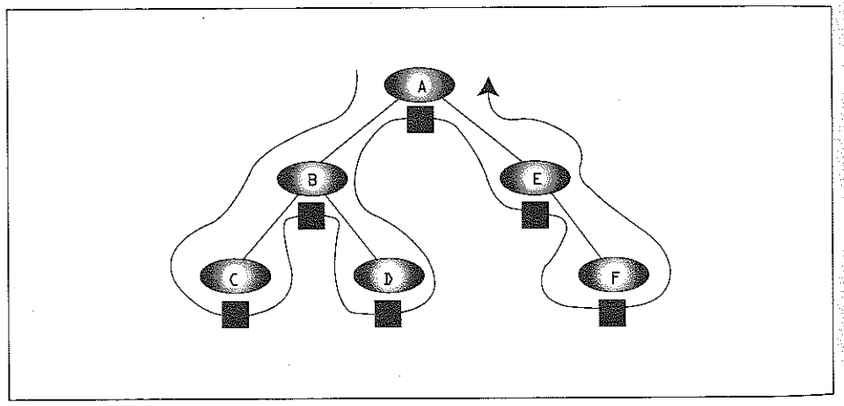


Figura 12.23 Recorrido en orden de un árbol binario

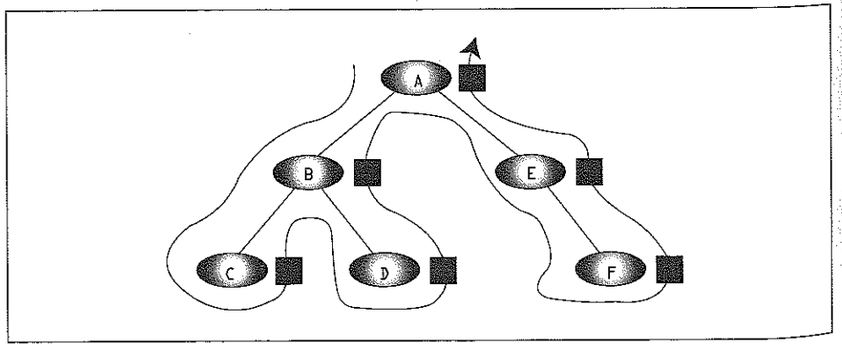


Figura 12.24 Recorrido en post orden de un árbol binario

- **Recorrido en post orden (LRN).** El último de los recorridos estándar es el **recorrido en post orden**, el cual procesa el nodo raíz después (*post*) de que los subárboles izquierdo y derecho se han procesado. Comienza localizando la hoja en el extremo izquierdo y procesándola. Luego procesa el hermano derecho, incluyendo sus subárboles (si los hay). Finalmente procesa el nodo raíz. La figura 12.24 muestra otra forma de visualizar el recorrido del árbol. Imagine que usted está caminando alrededor de un árbol, comenzando por la izquierda de la raíz y manteniéndose tan cerca de los nodos como le es posible. En el recorrido en post orden, usted procesa el nodo cuando está a la derecha del mismo. Esto aparece como una caja negra a la derecha del nodo. El camino se muestra como una línea que sigue una trayectoria completamente alrededor del árbol y de regreso a la raíz.

Recorridos primero en anchura

En el **recorrido primero en anchura** de un árbol binario, usted procesa todos los hijos de un nodo antes de continuar con el siguiente nivel. Dicho de otra forma, dada una raíz en el nivel n, se procesan todos los nodos del nivel n antes de continuar con los nodos del nivel $n + 1$. Para recorrer un árbol en orden de primero en profundidad, se utiliza una pila. Por otro lado, para recorrer un árbol primero en anchura, se utiliza una cola de espera.

Al igual que con los recorridos de primero en profundidad, se puede trazar el recorrido con un camino. Esta vez, no obstante, el camino continúa a modo de escalera, primero atraviesa el nivel raíz, luego atraviesa el nivel 1, enseguida el nivel 2 y así sucesivamente hasta que todo el árbol se recorre (figura 12.25).

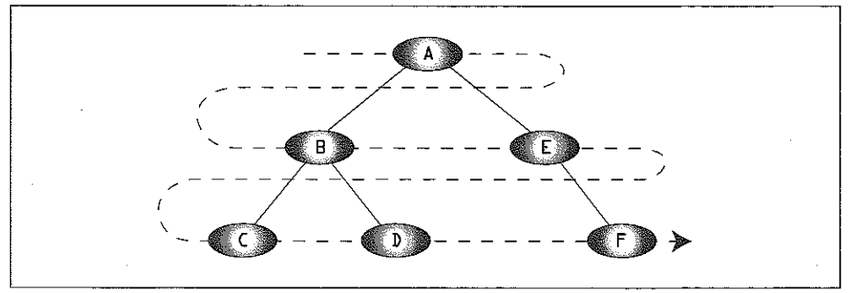


Figura 12.25 Recorrido primero en amplitud de un árbol binario

IMPLEMENTACIÓN DE UN ÁRBOL BINARIO

Un árbol binario normalmente se implementa como una lista ligada.

APLICACIONES DEL ÁRBOL BINARIO

Una interesante aplicación de árbol binario es el árbol de expresiones. Una expresión es una secuencia sintáctica que siguen las reglas prescritas. Un **elemento sintáctico** (*token*) puede ser ya sea un operando o un operador. En este análisis consideramos sólo los operadores aritméticos binarios en la forma de operando-operador-operando. Para simplificar el análisis, usamos únicamente cuatro operadores: suma, resta, multiplicación y división.

Un **árbol de expresiones** es un árbol binario con las propiedades siguientes:

1. Cada hoja es un operando.
2. La raíz y los nodos internos son operadores.
3. Los subárboles son subexpresiones, con la raíz siendo un operador.

Para un árbol de expresiones, los tres recorridos estándar representan tres formatos de expresión diferentes: **infijo**, **posfijo** y **prefijo**. El recorrido en orden produce la expresión de infijo, el recorrido en post orden genera la expresión de posfijo y el recorrido en pre orden produce la expresión de prefijo. La figura 12.26 muestra una expresión de infijo y su árbol de expresiones.

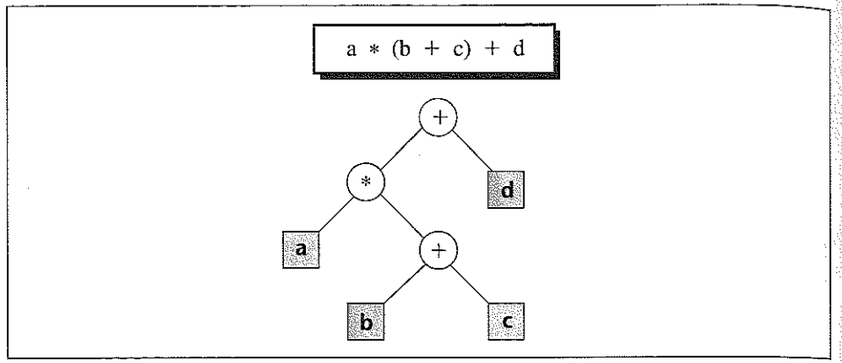


Figura 12.26 Árbol de expresiones

12.7 GRAFOS

Un grafo también puede clasificarse como un tipo de datos abstracto. Los grafos se pueden usar para resolver problemas de enrutamiento complejos, tales como el diseño y la elección de rutas de las aerolíneas entre los aeropuertos en que operan. De manera similar, pueden utilizarse para enrutar mensajes en una red de computadoras de un nodo a otro.

TERMINOLOGÍA

Un **grafo** es una colección de nodos, llamados vértices (*vertex*), y una colección de segmentos de línea, llamados **líneas**, que conectan pares de vértices. Es decir, un grafo se compone de dos conjuntos: un conjunto de vértices y un conjunto de líneas. Los grafos pueden ser ya sea dirigidos o no dirigidos. En un **grafo dirigido**, o **dígrafo**, para abreviar, cada línea tiene una dirección (punta de flecha) a su sucesor. Las líneas en un grafo dirigido se conocen como **arcos**. En este tipo de grafo el flujo a lo largo de los arcos entre dos vértices puede seguir sólo la dirección indicada. En un **grafo no dirigido** ninguna de las líneas, las cuales se conocen como **aristas**, tiene dirección (punta de flecha). En un grafo no dirigido el flujo entre dos vértices puede tomar cualquier dirección. La figura 12.27 contiene un ejemplo de ambos tipos, un grafo dirigido (a) y un grafo no dirigido (b).

Se dice que dos vértices en un grafo son **vértices adyacentes** (o vecinos) si una línea los conecta directamente. En la figura 12.27, A y B son adyacentes mientras que D y F no lo son.

Un **camino** es una secuencia de vértices en la cual cada vértice es adyacente al siguiente. En la figura 12.27, {A, B, C, E} es un camino y {A, B, E, F} es otro. Observe que tanto los grafos dirigidos como los no dirigidos tienen caminos. En un grafo no dirigido, usted podría viajar en cualquier dirección.

Un **ciclo** es un camino que consiste en al menos tres vértices e inicia y termina con el mismo vértice. En la figura 12.27(b), B, C, D, E, B es un ciclo. Sin embargo, advierta que los mismos vértices en la figura 12.27(a) no constituyen un ciclo porque en un dígrafo un camino sólo puede seguir la dirección del arco, mientras que en un grafo no dirigido un camino puede moverse en cualquier dirección a lo largo de la arista. Un **bucle** es un caso especial de ciclo en el cual un solo arco comienza y termina en el mismo vértice.

Se dice que dos vértices están conectados si hay un camino entre ellos. Se dice que un grafo está conectado si, al suprimir la dirección, hay un camino desde cualquier vértice a cualquier otro vértice. Además, un grafo dirigido está **fuertemente conectado** si hay un camino desde cada vértice a cada uno de los otros vértices en el digrafo. Un grafo dirigido está **débilmente conectado** si al menos dos vértices no están conectados. (Un grafo no dirigido conectado siempre está fuertemente conectado, así que el concepto normalmente no se usa con los grafos no dirigidos.) Un **grafo inconexo** no está conectado.

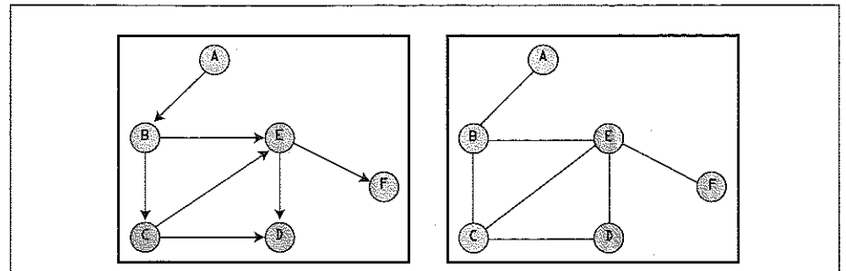


Figura 12.27 Grafos dirigidos y no dirigidos

El *grado* de un vértice es el número de líneas incidentes a él. El *grado de salida* de un vértice en un dígrafo es el número de arcos que salen del vértice; el *grado de entrada* es el número de arcos que entran al vértice.

OPERACIONES CON GRAFOS

En esta sección, definimos seis operaciones primitivas con grafos que proporcionan los módulos básicos necesarios para mantener un grafo: añadir un vértice, quitar un vértice, añadir una arista, quitar una arista, encontrar un vértice y recorrer un grafo. Como verá, hay dos métodos para el recorrido de grafos.

Añadir un vértice

La adición de un vértice inserta un nuevo vértice en un grafo. Cuando un vértice se añade, está inconexo; es decir, no está conectado a ningún otro vértice en la lista. Obviamente la adición de un vértice sólo es el primer paso en un proceso de inserción. Después de que se añade un vértice, éste debe conectarse. La figura 12.28 muestra un grafo antes y después de que se ha añadido un nuevo vértice.

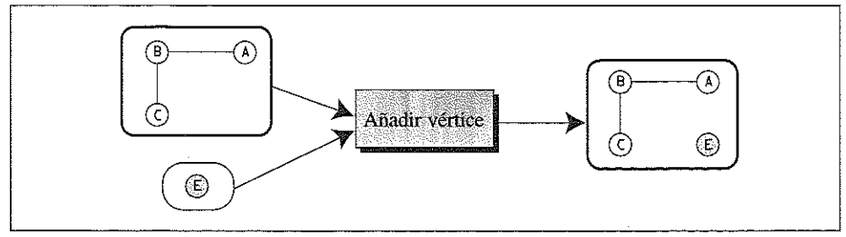


Figura 12.28 Adición de un vértice

Quitar un vértice

La eliminación de un vértice quita el vértice del grafo. Cuando se quita un vértice, todas las aristas que conecta también se eliminan. La figura 12.29 contiene un ejemplo de eliminación de un vértice.

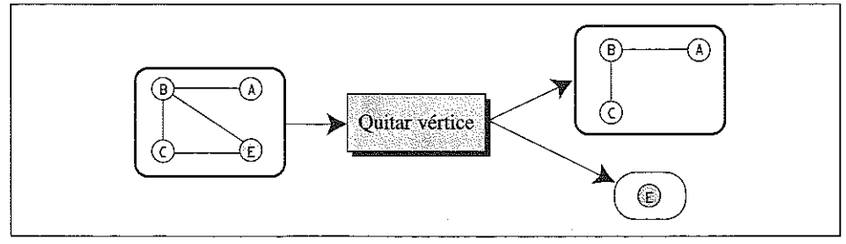


Figura 12.29 Eliminación de un vértice

Añadir una arista

La adición de una arista conecta un vértice con un vértice de destino. Si un vértice requiere múltiples aristas, entonces el módulo de añadir una arista debe llamarse una vez para cada vértice adyacente. Para añadir una arista, deben especificarse dos vértices. Si el grafo es un dígrafo, uno de los vértices debe especificarse como la fuente y otro como el destino. La figura 12.30 contiene un ejemplo de adición de la arista {A, E} a un grafo.

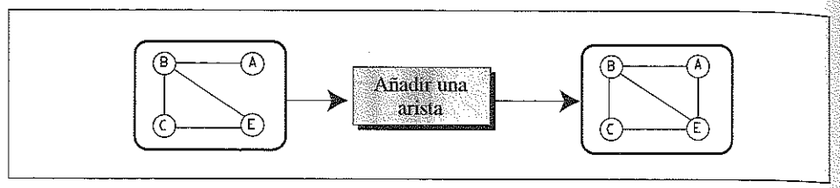


Figura 12.30 Adición de una arista

Quitar una arista

La eliminación de una arista quita la arista del grafo. La figura 12.31 contiene un ejemplo de la eliminación de la arista {A, E} de un grafo.

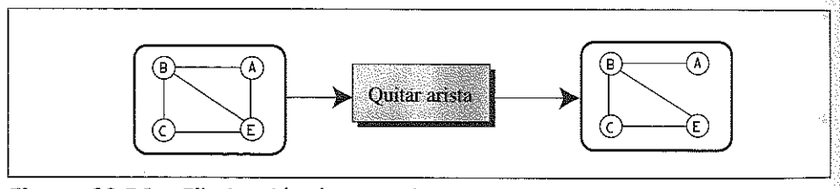


Figura 12.31 Eliminación de una arista

Encontrar un vértice

La operación encontrar un vértice recorre un grafo en busca de un vértice especificado. Si éste se encuentra, sus datos son devueltos. Si no se encuentra, se indica un error. En la figura 12.32, encontrar un vértice recorre el grafo en busca del vértice C.

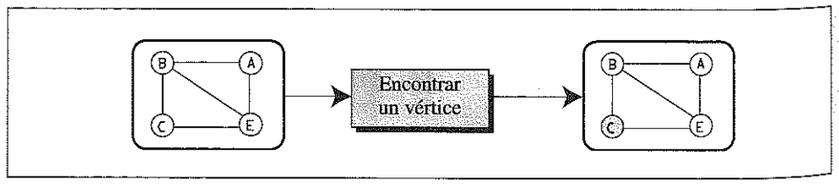


Figura 12.32 Encontrar un vértice

Recorrer un grafo

Siempre hay cuando menos una aplicación que requiere que todos los vértices en un grafo sean visitados; es decir, al menos una aplicación requiere que sea recorrido el grafo. Debido a que un vértice en un grafo puede tener múltiples padres, el recorrido de un grafo presenta algunos problemas que no se presentan en el recorrido de listas lineales y árboles. Específicamente, usted debe asegurarse de alguna manera de que procesa los datos en cada vértice sólo una vez. Sin embargo, debido a que hay varios caminos a un vértice, se puede llegar al mismo desde más de una dirección a medida que se recorre el grafo. La solución tradicional a este problema es incluir una bandera de visitado a cada vértice. Antes de hacer el recorrido, debe desactivarse la bandera de visitado en cada vértice. Luego, a medida que se recorre el grafo, la bandera de visitado se activa para indicar que los datos se han procesado.

Los dos recorridos de grafos estándar son primero en profundidad y primero en anchura. Ambos utilizan la bandera de visitado.

Recorrido primero en profundidad En el **recorrido primero en profundidad**, usted procesa los descendientes de un vértice antes de pasar a un vértice adyacente. Este concepto se explica más fácilmente cuando el grafo es un árbol. En la figura 12.33 mostramos un recorrido de pre orden, uno de los recorridos primero en profundidad estándar.

El recorrido primero en profundidad de un grafo comienza con el procesamiento del primer vértice del grafo. Después de procesar el primer vértice, se selecciona un vértice adyacente a éste y se procesa. Conforme se procesa cada vértice, se selecciona un vértice adyacente hasta que se llega a un vértice sin entradas adyacentes. Esto es similar a llegar a una hoja de un árbol. Luego usted retrocede sobre sus pasos por la estructura, procesando los vértices adyacentes mientras avanza. Es evidente que esta lógica requiere una pila (o recursión) para completar el recorrido.

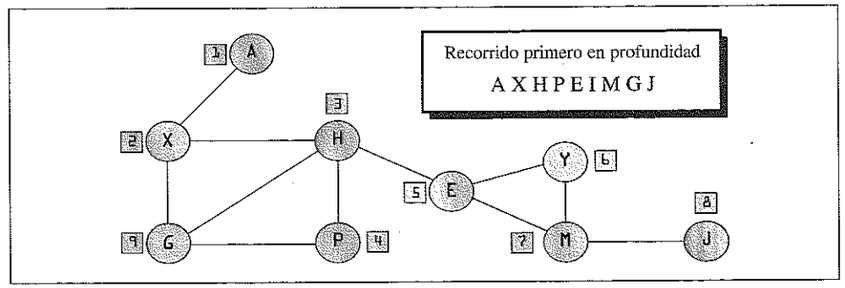


Figura 12.33 Recorrido primero en profundidad de un grafo

Recorrido primero en anchura En el **recorrido primero en anchura** de un grafo, usted procesa todos los vértices adyacentes de un vértice antes de que vaya al siguiente nivel. Al observar el árbol de la figura 12.34 usted ve que su recorrido primero en anchura comienza en el nivel 0 y luego procesa todos los vértices del nivel 1 antes de seguir adelante con el procesamiento de los vértices del nivel 2.

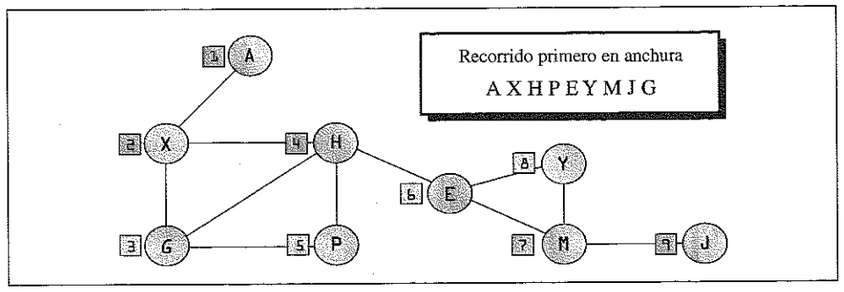


Figura 12.34 Recorrido primero en anchura de un grafo

El recorrido primero en anchura de un grafo sigue el mismo concepto. Comience seleccionando un vértice inicial; después procéselo y procese todos sus vértices adyacentes. Una vez que haya procesado todos los vértices adyacentes del primer vértice, seleccione el primer vértice adyacente y procese todos sus vértices; luego seleccione el segundo vértice adyacente y procese todos sus vértices, y así sucesivamente hasta que haya terminado con todos.

IMPLEMENTACIÓN DE UN GRAFO

Para representar un grafo, se necesitan dos conjuntos de datos. El primer conjunto representa los vértices del grafo y el segundo, las aristas o arcos. Las dos estructuras más comunes utilizadas para almacenar estos conjuntos son el arreglo (**matriz de adyacencia**) y la lista ligada (**lista de adyacencia**). La figura 12.35 muestra estas dos implementaciones para un grafo simple. Los números entre los vértices son pesos (por ejemplo, la distancias entre redes en una red interconectada).

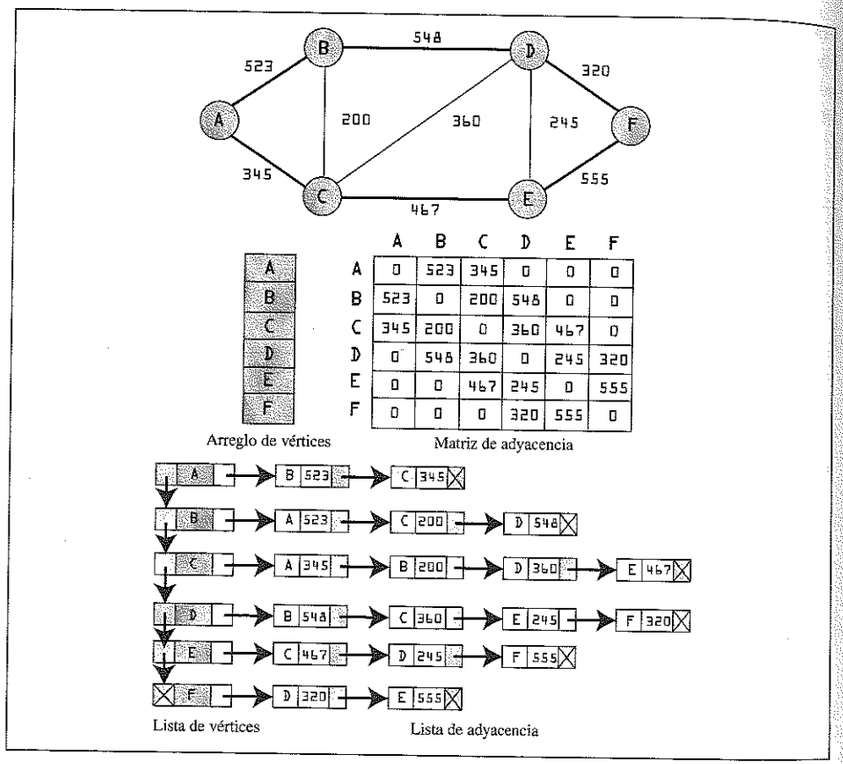


Figura 12.35 Implementaciones de grafos

APLICACIONES DE GRAFOS

Los grafos tienen muchas aplicaciones en las ciencias de la computación. Mencionamos dos de ellas aquí: las redes y los árboles de expansión mínima.

Redes

Una red es un grafo con líneas pesadas. También se le conoce como **grafo con peso**. El significado de los pesos depende de la aplicación. Por ejemplo, una aerolínea podría utilizar un grafo para representar las rutas entre las ciudades en las que da servicio. En este ejemplo los vértices representan las ciudades, y la arista es un camino entre dos ciudades. El peso de las aristas podría representar las millas de vuelo entre las dos ciudades o el precio del vuelo. Una red para una aerolínea hipotética pequeña se exhibe en la figura 12.35. En este caso, los pesos representan el millaje entre las ciudades.

Debido a que el peso es un atributo de una arista, se almacena en la estructura que contiene la arista. En una matriz de adyacencia, el peso se almacenaría como el valor de intersección. En una lista de adyacencia, éste se almacenaría como el valor en la lista ligada de adyacencia.

Árbol de expansión mínima

Podemos derivar uno o más árboles de expansión desde una red conectada. Un **árbol de expansión** (*spanning tree*) es un árbol que contiene todos los vértices del grafo. Un **árbol de expansión mínima** es un árbol de expansión tal que la suma de sus pesos es el mínimo. Hay muchas aplicaciones para los árboles de expansión mínima, todas con los requisitos para minimizar algún aspecto del grafo, como la distancia entre todos los vértices del grafo. Por ejemplo, dada una red de computadoras, usted podría crear un árbol que conecte todas las computadoras. El árbol de expansión mínima le da la menor longitud de cable que puede emplearse para conectar todas las computadoras en tanto que asegura que haya un camino entre dos computadoras cualesquiera.

12.8 TÉRMINOS CLAVE

abstracción
altura
análisis sintáctico
antecesor
árbol
árbol binario
árbol de expansión
árbol de expansión mínima
árbol de expresión
árbol nulo
arco
arista
arreglo
bucle
camino
ciclo
clave
cola de espera
descendiente
dígrafo
elemento sintáctico
eliminar de la cola de espera
equilibrio
extraer (*pop*)
factor de equilibrio
grado
grado de entrada
grado de salida
grafo
grafo con pesografo débilmente conectado
grafo dirigido
grafo fuertemente conectado
grafo inconexo
grafo no dirigido tipo de datos abstracto (TDA)
hermanos
hijo
hoja
infijo
insertar en la cola de espera
insertar (*push*)
línea
lista
lista aleatoria
lista de adyacencia
lista general
lista lineal
lista ordenada
lista restringida
matriz de adyacencia
nivel
nodo
nodo interno
padre
pila
posfijo
postergación
prefijo
primero en entrar, primero en salir (FIFO)
profundidad
raíz
rama
recorrido
recorrido de árbol binario
recorrido en orden (LNR)
recorrido en pre orden
recorrido en post orden
recorrido primero en anchura
recorrido primero en profundidad
recuperación
red
retroceso
simulación de cola de espera
sobre desbordamiento
subárbol
último en entrar, primero en salir (LIFO)
vértice
vértices adyacentes

12.9 RESUMEN

- En una lista lineal cada elemento tiene un sucesor único.
- Las listas lineales pueden dividirse en dos categorías: general y restringida.
- En una lista general, los datos puede insertarse y eliminarse en cualquier parte, y no hay restricciones a las operaciones que pueden usarse para procesar la lista.
- En una lista restringida, los datos pueden añadirse o eliminarse en los extremos de la estructura, y el procesamiento está restringido a las operaciones aplicadas a los datos en los extremos de la lista.
- Dos estructuras de lista restringidas comunes son pilas (listas último en entrar, primero en salir [FIFO]) y las colas de espera (listas primero en entrar, primero en salir [FIFO]).
- Cuatro operaciones comunes están asociadas con listas lineales: inserción, eliminación, recuperación y recorrido.
- Una pila es una lista lineal en la cual las adiciones y eliminaciones están restringidas a un extremo llamado cima. Una pila también se conoce como lista LIFO.
- Definimos dos operaciones para una pila: insertar (push) y extraer (pop).
- La inserción añade un elemento a la parte superior de la pila. Después de la inserción, el nuevo elemento se vuelve la cima.
- La extracción elimina el elemento en la parte superior de la pila. Después de la extracción, el elemento que le sigue, si hay alguno, se convierte en la cima.
- En este capítulo se analizaron cuatro aplicaciones de pilas: inversión de datos, análisis sintáctico de datos, postergación del uso de los datos y retroceso sobre los pasos.
- Una cola de espera es una lista lineal en la cual los datos sólo pueden insertarse en un extremo, llamado parte

trasera, y eliminarse en el otro extremo, la parte frontal. Una cola de espera es una estructura primero en entrar, primero en salir (FIFO).

- En este capítulo analizamos dos operaciones de cola de espera: la inserción (*enqueuing*) y la eliminación (*dequeuing*) de un elemento.
- Una cola de espera puede usarse para clasificar los datos.
- Un árbol consiste en un conjunto finito de elementos llamados nodos y un conjunto finito de líneas dirigidas llamadas ramas que conectan los nodos. El número de ramas asociadas con un nodo es el grado del nodo. Cuando la rama se dirige hacia el nodo, es una rama de grado de entrada; cuando se dirige hacia afuera del nodo, es una rama de grado de salida. La suma de las ramas de grado de entrada y de grado de salida es el grado del nodo.
- Si el árbol no está vacío, el primer nodo se llama raíz, la cual tiene un grado de entrada de cero. Todos los nodos en el árbol, excepto la raíz, deben tener un grado de entrada de uno.
- Una hoja es un nodo con un grado de salida de cero.
- Un nodo interno no es ni raíz ni hoja.
- Un nodo puede ser padre, hijo o ambos. Dos o más nodos con el mismo padre se llaman hermanos.
- Un camino es una secuencia de nodos en la cual cada nodo es adyacente al siguiente.
- Un antecesor es cualquier nodo en el camino desde la raíz hasta un nodo dado. Un descendiente es cualquier nodo en todos los caminos de un nodo dado a una hoja.
- El nivel de un nodo es su distancia desde la raíz.
- La altura de un árbol es el nivel de la hoja en el camino más largo desde la raíz más 1; la altura de un árbol vacío es -1.
- Un subárbol es cualquier estructura conectada debajo de la raíz.
- Un árbol puede definirse de manera recursiva como un conjunto de nodos que ya sea (1) está vacío o (2) tiene un nodo designado llamado raíz desde el cual descienden jerárquicamente cero o más subárboles, los cuales también son árboles.
- En un árbol binario, ningún nodo puede tener más de dos hijos.
- Un recorrido de árbol binario visita cada nodo del árbol una y sólo una vez en una secuencia predeterminada.
- Los dos métodos del recorrido de árbol binario son primero en profundidad y primero en anchura.
- Utilizando el método de primero en profundidad, usted recorre un árbol binario en seis secuencias diferentes; sin embargo, sólo tres de estas secuencias reciben nombres estándar: pre orden, en orden y post orden.
  - **a.** En el recorrido en pre orden, primero se procesa la raíz, seguida por el subárbol izquierdo y luego el subárbol derecho.
  - **b.** En el recorrido en orden, primero se procesa el subárbol izquierdo, luego la raíz y por último el subárbol derecho.
  - **c.** En el recorrido en post orden, primero se procesa el subárbol izquierdo, luego el derecho y al último la raíz.
- En el método primero en anchura, usted procesa todos los nodos en un nivel antes de continuar con el nivel siguiente.
- Definimos una aplicación para un árbol binario en este capítulo: árbol de expresión.
- Un grafo es una colección de nodos llamados vértices, y una colección de segmentos de línea que conectan pares de nodos llamados aristas o arcos.
- Los grafos pueden ser dirigidos o no dirigidos. En un grafo dirigido o dígrafo cada línea tiene una dirección. En un grafo no dirigido no hay dirección en las líneas. Una línea en un grafo dirigido se llama arco. Una línea en un grafo no dirigido se llama arista.
- En un grafo se dice que dos vértices son adyacentes si una arista los conecta directamente.
- Un camino es una secuencia de vértices en la cual cada vértice es adyacente al siguiente.
- Un ciclo es un camino de al menos tres vértices que inicia y termina con el mismo vértice.
- Un bucle es un caso especial de ciclo en el cual un solo arco comienza y termina con el mismo nodo.
- Se dice que un grafo está conectado si, para dos vértices cualesquiera, hay un camino del uno al otro. Un grafo es inconexo si no está conectado.
- El grado de un vértice es el número de vértices adyacentes al mismo. El grado de salida de un vértice es el número de arcos que salen del nodo, mientras que el grado de entrada es el número de arcos que entran al nodo.
- Se definieron seis operaciones para un grafo: añadir un vértice, quitar un vértice, añadir una arista, quitar una arista, encontrar un vértice y recorrer un grafo.
- Hay dos recorridos de grafos estándar: primero en profundidad y primero en amplitud.
- En el recorrido primero en profundidad, todos los descendientes de un nodo se procesan antes de moverse a un nodo adyacente.
- En el recorrido primero en amplitud todos los vértices adyacentes se procesan antes de procesar a los descendientes de un vértice.

- Para representar un grafo en una computadora, se necesita almacenar dos conjuntos de información: El primer conjunto representa los vértices y el segundo representa las aristas.
- Los métodos más comunes utilizados para almacenar un grafo son el método de matriz de adyacencia y el método de lista de adyacencia.
- En el método de matriz de adyacencia, se usa un arreglo para almacenar los vértices y una matriz para almacenar las aristas.
- En el método de lista de adyacencia, se utiliza una lista ligada para almacenar los vértices y una matriz para almacenar las aristas.
- Definimos dos aplicaciones para un grafo en este capítulo: una red y un árbol de expansión mínima.

12.10 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Qué es un tipo de datos abstracto?
2. En un TDA, ¿qué se conoce y qué está oculto?
3. ¿Qué es una lista lineal?
4. ¿Cuál es la diferencia entre una lista general y una lista restringida?
5. ¿Cuáles son dos implementaciones comunes de una lista general?
6. ¿Qué ventaja tiene la implementación de una lista ligada de una lista lineal general sobre la implementación con un arreglo de una lista lineal general?
7. ¿Cuál es la diferencia entre la operación de insertar y la operación de extraer?
8. ¿Cuáles son tres operaciones de pila comunes?
9. Describa las operaciones de inserción (*enqueuing*) y eliminación (*dequeuing*).
10. ¿Qué significa que un árbol binario esté equilibrado?
11. ¿Cuál es la diferencia entre un recorrido primero en profundidad y un recorrido primero en anchura de un árbol binario?
12. ¿Cuál es la relación entre un camino y un ciclo?
13. ¿Cuál es la diferencia entre un recorrido primero en profundidad y un recorrido primero en amplitud de un grafo?
14. ¿Cuál es la principal limitación de usar un arreglo para implementar un grafo? ¿Por qué esto representa un problema?
15. ¿Cuál es la diferencia entre un grafo y una red?

PREGUNTAS DE OPCIÓN MÚLTIPLE

16. En un tipo de datos abstracto, _____ .
    - **a.** se conoce la implementación de TDA
    - **b.** la implementación de TDA está oculta
    - **c.** las funciones de TDA están ocultas
    - **d.** ninguna de las anteriores
17. Un(a) _____ podría ser un TDA.
    - **a.** matriz
    - **b.** lista ligada
    - **c.** árbol
    - **d.** todas las anteriores
18. En un TDA, _____.
    - **a.** los datos se declaran
    - **b.** las operaciones se declaran
    - **c.** los datos y las operaciones están encapsulados
    - **d.** todas las anteriores
19. Un(a) _____ es una lista en la cual cada elemento tiene un sucesor único.
    - **a.** matriz
    - **b.** red
    - **c.** lista lineal
    - **d.** lista ligada
20. Una lista FIFO es una lista lineal _____.
    - **a.** general
    - **b.** restringida
    - **c.** sin ordenar
    - **d.** a o b
21. Una lista lineal _____ puede ser ordenada o sin ordenar.
    - **a.** general
    - **b.** restringida
    - **c.** FIFO
    - **d.** LIFO
22. Una lista _____ también se conoce como cola de espera.
    - **a.** LIFO
    - **b.** FIFO
    - **c.** sin ordenar
    - **d.** ordenada
23. Una lista _____ también se conoce como pila.
    - **a.** LIFO
    - **b.** FIFO
    - **c.** sin ordenar
    - **d.** ordenada

24. Cuando no hay suficiente espacio para inserción, una lista ordenada está en un estado _____.
- **a.** de desbordamiento
- **b.** de sobre desbordamiento
- **c.** lento
- **d.** restringido

25. Cuando una lista ordenada está en un estado _____, la lista está vacía.
- **a.** de desbordamiento
- **b.** de sobre desbordamiento
- **c.** lento
- **d.** restringido

26. En la _____ de listas para una lista ordenada, los datos en la lista y el número de elementos de la lista permanecen sin cambio.
- **a.** inserción
- **b.** eliminación
- **c.** recuperación
- **d.** todos los anteriores

27. En la(el) _____ de listas para una lista ordenada, cada elemento de la lista se procesa en forma secuencial.
- **a.** inserción
- **b.** eliminación
- **c.** recuperación
- **d.** recorrido

28. Si A es el primer elemento de datos que se introduce en una pila seguido por B, C y D, _____ es el primer elemento a ser eliminado.
- **a.** A
- **b.** B
- **c.** C
- **d.** D

29. Si A es el primer elemento de datos que se introduce en una cola de espera seguido por B, C y D, _____ es el primer elemento a ser eliminado.
- **a.** A
- **b.** B
- **c.** C
- **d.** D

30. La operación de extraer _____ de la pila.
- **a.** elimina un elemento de la parte superior
- **b.** elimina un elemento de la parte inferior
- **c.** añade un elemento a la parte superior
- **d.** añade un elemento a la parte inferior

31. La operación de insertar _____ de la pila.
- **a.** elimina un elemento de la parte superior
- **b.** elimina un elemento de la parte inferior
- **c.** añade un elemento a la parte superior
- **d.** añade un elemento a la parte inferior

32. Cuando los datos se dividen en piezas independientes para su procesamiento posterior, a esto se le llama _____.
- **a.** inversión de los datos
- **b.** postergación de los datos
- **c.** análisis sintáctico de los datos
- **d.** retroceso de los datos

33. En una cola de espera, los datos se insertan sólo en _____ y se eliminan sólo en _____.
- **a.** la parte trasera; la parte frontal o la parte trasera
- **b.** la parte frontal; la parte trasera
- **c.** la parte trasera; la parte frontal
- **d.** la parte trasera o la parte frontal; la parte frontal

34. El grado de entrada de _____ de un árbol siempre es cero.
- **a.** cualquier nodo
- **b.** una rama
- **c.** la raíz
- **d.** una hoja

35. Si un nodo interno tiene cuatro ramas de grado de salida, su grado es _____.
- **a.** 9
- **b.** 1
- **c.** 4
- **d.** 5

36. Un nodo de un árbol tiene un grado de 3. Esto significa que su grado de salida es _____.
- **a.** 0
- **b.** 2
- **c.** 4
- **d.** ninguno de los anteriores

37. Un(a) _____ es una secuencia de nodos en la cual cada nodo es adyacente al siguiente.
- **a.** hoja
- **b.** raíz
- **c.** descendente
- **d.** camino

38. Si la altura de un árbol es 10, el nivel más alto del árbol es _____.
- **a.** 10
- **b.** 9
- **c.** 5
- **d.** 1

39. En un árbol binario, cada nodo tiene _____ dos subárboles.
- **a.** más de
- **b.** menos de
- **c.** cuando mucho
- **d.** cuando menos

40. Si hay 22 nodos a almacenar en un árbol binario, la altura máxima del árbol es _____.
- **a.** mayor que 22
- **b.** menor que 22
- **c.** igual que 22
- **d.** ninguna de las anteriores

41. Si hay 16 nodos a almacenar en un árbol binario, la altura mínima del árbol es _____.
- **a.** 16
- **b.** 5
- **c.** 4
- **d.** 1

42. Un árbol binario tiene una altura de 5. ¿Cuál es el número mínimo de nodos?
- **a.** 31
- **b.** 15
- **c.** 5
- **d.** 1

42. Un árbol binario tiene una altura de 5. ¿Cuál es el número máximo de nodos?
- **a.** 31
- **b.** 15
- **c.** 5
- **d.** 1

44. En un recorrido en pre orden, _____ se procesa primero.
- **a.** el subárbol izquierdo
- **b.** el subárbol derecho
- **c.** la raíz
- **d.** a o b

45. En un recorrido _____, el subárbol derecho se procesa al final.
- **a.** en pre orden
- **b.** en orden
- **c.** en post orden
- **d.** a o b

46. En un recorrido en post orden, la raíz se procesa _____.
- **a.** primero
- **b.** en segundo lugar
- **c.** al final
- **d.** a o b

47. En un recorrido en post orden, el subárbol izquierdo se procesa _____.
- **a.** primero
- **b.** en segundo lugar
- **c.** al final
- **d.** a o b

48. En un recorrido _____, el subárbol izquierdo se procesa al final.
- **a.** en pre orden
- **b.** en orden
- **c.** en post orden
- **d.** ninguno de los anteriores

49. En un recorrido en orden, la raíz se procesa _____.
- **a.** primero
- **b.** en segundo lugar
- **c.** al último
- **d.** a o b

50. En un recorrido primero en anchura de un árbol binario con tres niveles (0, 1 y 2), ¿cuál nivel se procesa al último?
- **a.** 0
- **b.** 1
- **c.** 2
- **d.** ninguno de los anteriores

51. Un(a) _____ es una línea entre dos vértices en un digrafo.
- **a.** nodo
- **b.** arco
- **c.** arista
- **d.** camino

52. Una arista es _____ entre dos vértices en un grafo no dirigido.
- **a.** un nodo
- **b.** un arco
- **c.** una línea
- **d.** un camino

53. Si C y D son dos vértices adyacentes en un grafo no dirigido, entonces _____.
- **a.** hay dos caminos
- **b.** sólo hay un camino {C, D}
- **c.** sólo hay un camino {D, C}
- **d.** no hay caminos

54. Un vértice en un dígrafo tiene cuatro arcos que entran y tres arcos que salen. El grado de salida del vértice es _____.
- **a.** 4
- **b.** 3
- **c.** 2
- **d.** 1

EJERCICIOS

55. Muestre el contenido de la pila s1 después de las operaciones siguientes:

```
push (s1, 5)
push (s1, 3)
push (s1, 2)
pop (s1)
pop (s1)
push (s1, 6)
```

56. Utilice un ciclo `while` para vaciar el contenido de la pila s2.

57. Utilice un ciclo `while` para mover el contenido de la pila s1 a s2. Después de la operación la pila s1 debe estar vacía.

58. Utilice un ciclo `while` para copiar el contenido de la pila s1 a s2. Después de la operación el contenido de las pilas s1 y s2 debe ser el mismo.

59. Utilice un bucle `while` para concatenar el contenido de la pila s2 con el contenido de la pila s1. Después de la concatenación los elementos de la pila s2 deben estar encima de los elementos de la pila s1. La pila s2 debe estar vacía.

60. Un palíndromo es una cadena que puede leerse hacia atrás y hacia delante con el mismo resultado. Por ejemplo, la creación siguiente es un palíndromo:

```
Anita lava la tina
```

Escriba un algoritmo utilizando una pila para probar si una cadena es un palíndromo.

61. Escriba un algoritmo para comparar el contenido de dos pilas.

62. Muestre el contenido de la pila s1 y de la cola de espera q1 después de las operaciones siguientes:

```
push (s1, 3)
push (s1, 5)
enqueue (q1, 6)
enqueue (q1, 9)
pop (s1)
enqueue (q1, 9)
dequeue (q1)
```

63. Utilice un ciclo `while` para vaciar el contenido de la cola de espera q3.

64. Utilice un ciclo `while` para mover el contenido de la cola de espera q2 a la cola q3. Después de la operación la cola de espera q2 debe estar vacía.

65. Utilice un ciclo `while` para copiar el contenido de la cola de espera q2 a q3. Después de la operación el contenido de las colas de espera q2 y q3 debe ser el mismo.

66. Utilice un ciclo `while` para concatenar el contenido de la cola de espera q2 con el contenido de q1. Después de la concatenación, los elementos de la cola de espera q2 deben estar al final de los elementos de q1. La cola de espera q2 debe estar vacía.

67. Escriba un algoritmo para comparar el contenido de dos colas de espera.

68. Encuentre la raíz de cada uno de los árboles binarios siguientes:

a. Árbol con recorrido en post orden: FCBDG
b. Árbol con recorrido en pre orden: IBCDFEN
c. Árbol con recorrido en post orden: CBIDFGE

69. Un árbol binario tiene 10 nodos. El recorrido en orden en pre orden del árbol se muestra enseguida. Dibuje el árbol:

En pre orden: JCBADEFIGH
En orden: ABCEDFJGIH

70. Un árbol binario tiene ocho nodos. El recorrido en orden y en post orden del árbol se muestra enseguida. Dibuje el árbol:

En post orden: FECHGDBA
En orden: FECABHDG

71. Un árbol binario tiene siete nodos. El recorrido en orden y en post orden del árbol se muestra enseguida. ¿Puede dibujar el árbol? Si no es así, explique por qué.

En post orden: GFDABEC
En orden: ABDCEFG

72. Dibuje todos los árboles diferentes posibles con tres nodos (A, B y C).

73. Haga el recorrido primero en profundidad del grafo de la figura 12.36 partiendo del vértice A.

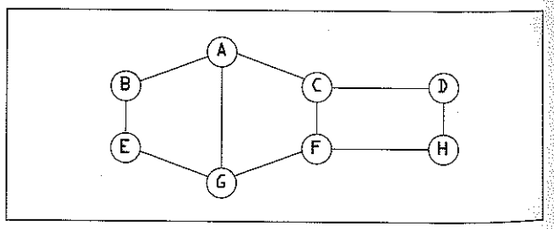


Figura 12.36 Ejercicios 73 y 74

74. Haga el recorrido primero en anchura del grafo de la figura 12.36 partiendo del vértice A.

75. Dé la representación de la matriz de adyacencia del grafo de la figura 12.37.

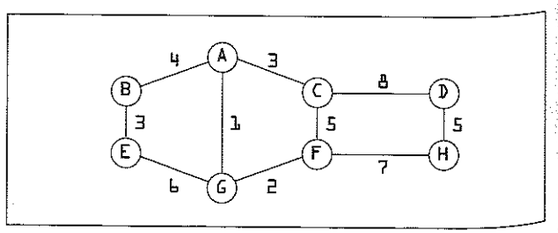
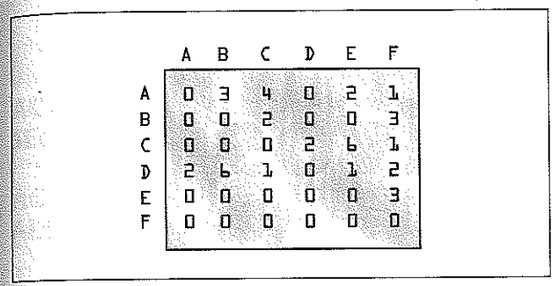


Figura 12.37 Ejercicios 75 y 76

76. Haga una representación de la lista de adyacencia del grafo de la figura 12.37.

77. Trace el grafo para la representación de la matriz de adyacencia de la figura 12.38.



	A	B	C	D	E	F
A	0	3	4	0	2	1
B	0	0	2	0	0	3
C	0	0	0	2	6	1
D	2	6	1	0	1	2
E	0	0	0	0	0	3
F	0	0	0	0	0	0

Figura 12.38 Ejercicio 77

El código siguiente muestra cómo tener acceso a todos los registros en un archivo secuencial.

```
while Not EOF
{
    Lee el registro siguiente
    Procesa el registro
}
```

Programa 13.1 Procesamiento de registros en un archivo secuencial

Para leer y procesar los registros uno a uno se utiliza un ciclo. Después de que el sistema operativo procesa el último registro se detecta el EOF y se termina el ciclo.

El archivo secuencial se utiliza en aplicaciones que necesitan tener acceso a todos los registros de principio a fin. Por ejemplo, si la información personal sobre cada empleado en una compañía se almacena en un archivo, se puede usar el **acceso secuencial** para recuperar cada registro al final del mes con el fin de imprimir los cheques de sueldo. En este punto, debido a que usted debe procesar cada registro, el acceso secuencial es más eficiente y fácil que el acceso aleatorio.

Sin embargo, el archivo secuencial no es eficiente para el acceso aleatorio. Por ejemplo, si sólo se puede tener acceso a todos los registros de clientes en un banco en forma secuencial, un cliente que necesita obtener dinero de un cajero automático tendría que esperar mientras el sistema revisa cada registro desde el principio del archivo hasta que llega al registro del cliente. Si este banco tiene un millón de clientes, el sistema, en promedio, recuperaría medio millón de registros antes de obtener el registro del cliente. Esto es muy ineficiente.

ACTUALIZACIÓN DE ARCHIVOS SECUENCIALES

Los archivos secuenciales deben actualizarse en forma periódica para reflejar cambios en la información. El proceso de actualización es muy problemático debido a que todos los registros necesitan revisarse y actualizarse (de ser necesario) en forma secuencial.

Archivos involucrados en la actualización

Hay cuatro archivos asociados con un programa de actualización: el archivo maestro nuevo, el archivo maestro viejo, el archivo de transacción y el archivo de informe de errores.

- **Archivo maestro nuevo.** Primero está el nuevo archivo de datos permanente o, como se le conoce comúnmente, el **archivo maestro nuevo**, el cual contiene los datos más actuales.
- **Archivo maestro viejo.** El **archivo maestro viejo** es el archivo permanente que debe actualizarse. Aun después de la actualización, el archivo maestro viejo debe mantenerse para referencia.
- **Archivo de transacción**. El tercer archivo es el **archivo de transacción**, el cual contiene los cambios a aplicar en el archivo maestro. Hay tres tipos básicos de cambios en todas las actualizaciones de archivos. La *adición de transacciones* contiene los datos de un nuevo registro a añadir en el archivo maestro. La *eliminación de transacciones* identifica registros a eliminar del archivo. Y el *cambio de transacciones* contiene revisiones a registros específicos en el archivo. Para procesar cualquiera de estas transacciones, usted necesita una llave. Una **llave** es uno o más campos que identifican de forma única los datos en el archivo. Por ejemplo, en un archivo de estudiantes, la llave podría ser el ID (identificador) de estudiante. En un archivo de empleados, la llave podría ser el número del seguro social.
- **Archivo de informe de errores.** El cuarto archivo requerido en un programa de actualización es un **archivo de informe de errores**. Es muy raro que un proceso de actualización no produzca al menos un error. Cuando un error ocurre, se necesita informarlo al usuario. El *informe de error* contiene un listado de todos los errores descubiertos durante el proceso de actualización y se presenta al usuario para que lo rectifique.

La figura 13.3 es una representación gráfica de la actualización de un archivo secuencial. En esta figura, aparecen los cuatro archivos que se han analizado. Aunque utilizamos el símbolo de cinta para los archivos, podríamos haberlos representado con la misma facilidad mediante el símbolo de disco. Observe que después de que se completa el programa de actualización, el nuevo archivo maestro se envía a almacenamiento fuera de línea donde se guarda hasta que se le necesita de nuevo. Cuando el archivo va a actualizarse, el archivo maestro se recupera del almacenamiento fuera de línea y se vuelve el archivo maestro viejo.

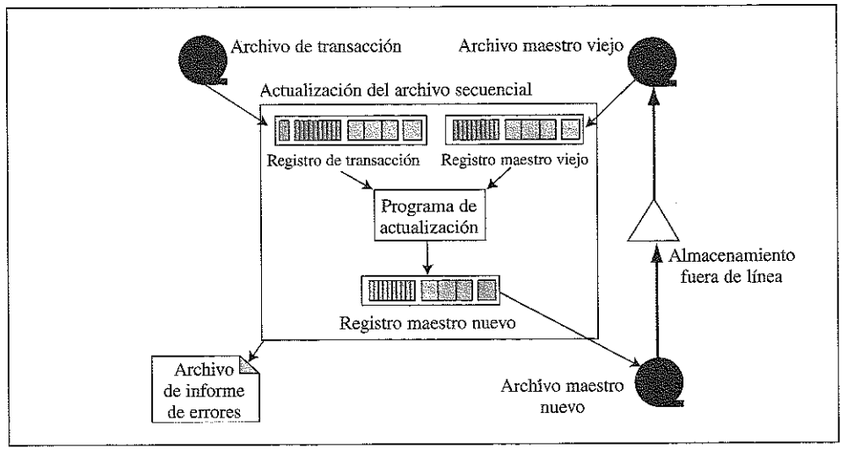


Figura 13.3 Actualización de un archivo secuencial

Para hacer el proceso de actualización eficiente, todos los archivos se almacenan en la misma llave. El proceso de actualización requiere que usted compare las llaves de los archivos maestro y de transacción y, suponiendo que no hay errores, realice una de las tres acciones siguientes:

1. Si la llave del archivo de transacción es menor que la llave del archivo maestro, añada la transacción al archivo maestro nuevo.
2. Si la llave del archivo de transacción es igual a la llave del archivo maestro, opte por una de estas dos opciones:
   a. Cambie el contenido de los datos del archivo maestro si la transacción es una revisión (R).
   b. Borre los datos del archivo maestro si la transacción es una eliminación (D).
3. Si la llave del archivo de transacción es mayor que la llave del archivo maestro, escriba el registro del archivo maestro viejo en el archivo maestro nuevo. En este caso, el código de transacción debe ser la adición (A) o hay un error.

Este proceso de actualización se exhibe en la figura 13.4. En el archivo de transacción los códigos de transacción son A para la adición, D para la eliminación y R para revisión. El proceso comienza con la comparación de las llaves para el primer registro de cada archivo.

13.3 ARCHIVOS INDEXADOS

Para tener acceso aleatorio a un archivo en un registro, se necesita conocer la dirección del registro. Por ejemplo, suponga que una clienta quiere revisar su cuenta bancaria. Ni la clienta ni la cajera conocen la dirección del registro de la clienta. Ésta sólo puede darle a la cajera su número de cuenta (llave). Aquí, un archivo indexado puede relacionar el número de cuenta (llave) con la dirección del registro (figura 13.5).

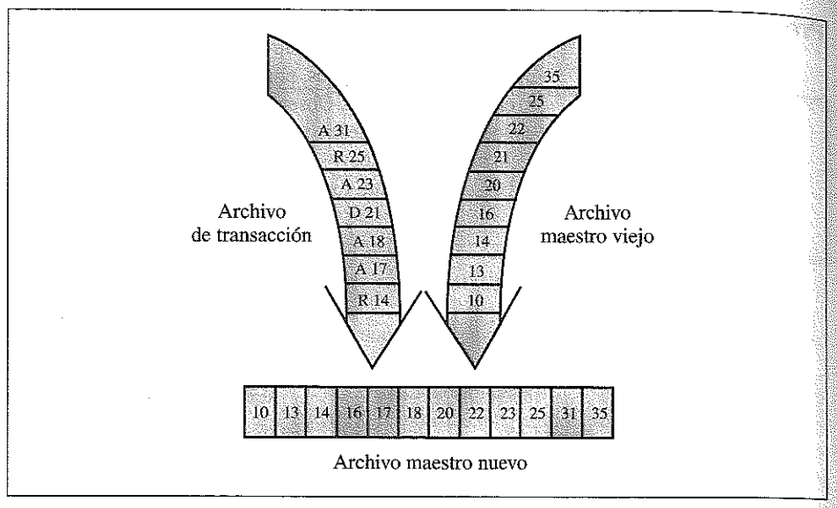


Figura 13.4 Proceso de actualización

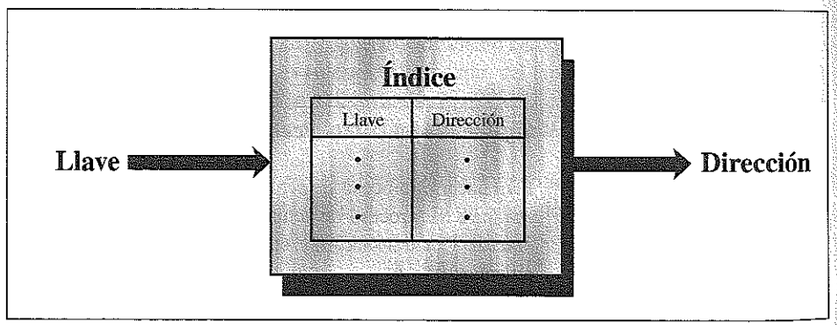


Figura 13.5 Relación de un archivo indexado

Un **archivo indexado** se compone de un **archivo de datos**, el cual es un archivo secuencial y un **índice.** El índice en sí es un archivo muy pequeño con sólo dos campos: la llave del archivo secuencial y la dirección del registro correspondiente en el disco. La figura 13.6 muestra la vista lógica de un archivo indexado. Para tener acceso a un registro de un archivo de este tipo, siga estos pasos:

1. Todo el archivo indexado se carga en la memoria principal (el archivo es pequeño y utiliza poca memoria).
2. Se buscan las entradas, utilizando un algoritmo de búsqueda eficiente tal como una búsqueda binaria, para encontrar la llave deseada.
3. La dirección del registro se recupera.
4. Utilizando la dirección. El registro de datos se recupera y se pasa al usuario.

ARCHIVOS INVERTIDOS

Una de las ventajas del archivo indexado es que usted puede tener más de un índice, cada uno con una llave distinta. Por ejemplo, un archivo de empleados puede recuperarse con base ya sea en el número del seguro social o en el apellido. Este tipo de archivo indexado por lo general se llama **archivo invertido**.

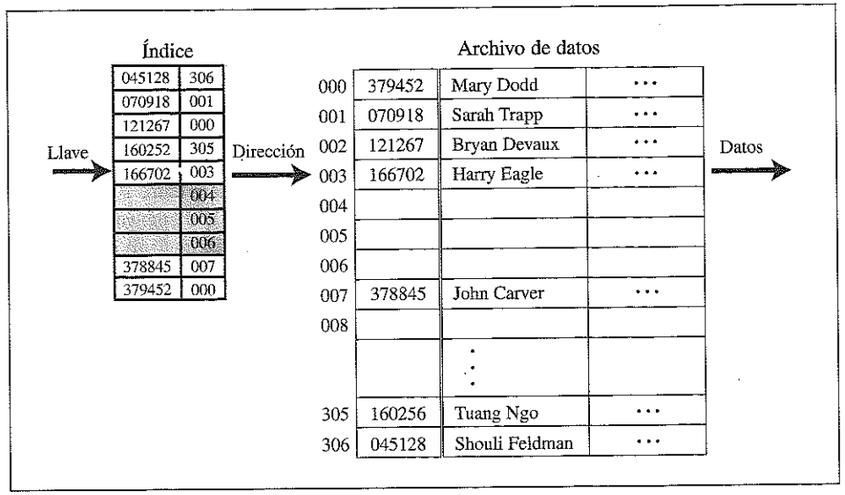


Figura 13.6 Vista lógica de un archivo indexado

13.4 ARCHIVOS HASHED

En un archivo indexado, el índice relaciona la llave con la dirección. Un **archivo hashed** utiliza una función para lograr esta relación. El usuario da la llave, la función relaciona la llave con la dirección y la pasa al sistema operativo y el registro se recupera (figura 13.7).

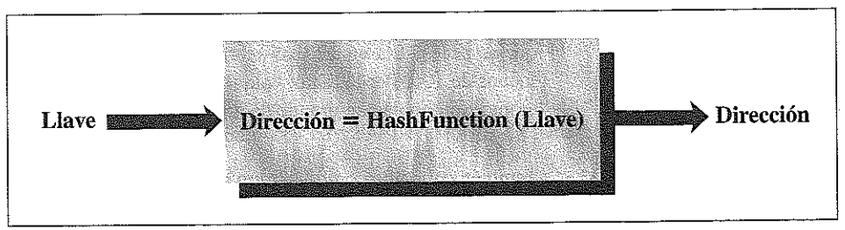


Figura 13.7 Relación en un archivo hashed

El archivo hashed elimina la necesidad de un archivo (índice) extra. En un archivo indexado usted debe mantener este índice en el disco y cuando necesita procesar el archivo de datos, primero debe cargar el índice en la memoria, buscarlo para encontrar la dirección del registro de datos y luego acceder al archivo de datos para tener acceso al registro. En un archivo hashed la dirección se encuentra mediante el uso de una función. No hay necesidad de un índice ni de toda la sobrecarga asociada con él. No obstante, verá que los archivos hashed tienen sus propios problemas.

MÉTODOS DE HASHING

Para la relación de llaves-direcciones, usted puede seleccionar uno de varios métodos de dispersión, también llamado hashing. Analizaremos unos cuantos de ellos en esta sección.

Método directo

En el **hashing directo**, la llave es la dirección sin ninguna manipulación algorítmica. El archivo debe, por tanto, contener un registro para cada llave posible. Aunque las situaciones para el hashing directo son limitadas, éste puede ser muy poderoso debido a que garantiza que no haya sinónimos o colisiones (los sinónimos o colisiones se analizan posteriormente en este capítulo) como sucede con otros métodos.

Veamos un ejemplo trivial. Imagine que una organización tiene menos de 100 empleados. A cada empleado se asigna un número entre 1 y 100 (ID del empleado). En este caso, si usted crea un archivo de registros de 100 empleados, el número de empleado puede utilizarse directamente como la dirección de un registro individual. Este concepto se muestra en la figura 13.8. El registro con la llave 025 (Vu Nguyen...) se relacionó a la dirección (sector) 025. Observe que no todos los elementos en el archivo contienen un registro de empleado. Una parte del espacio está desperdiciada.

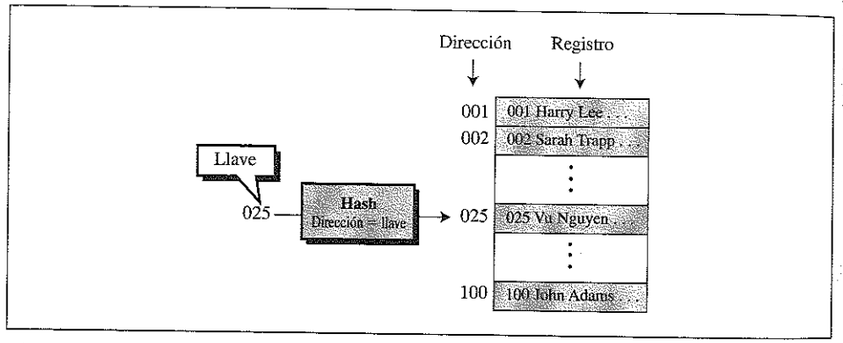


Figura 13.8 Hashing directo

Aun cuando éste es el método ideal, su aplicación está muy limitada. Por ejemplo, es muy ineficaz utilizar el número del seguro social como la llave porque éste tiene nueve dígitos. Así que se necesitaría un archivo enorme con 999 999 999 registros, de los cuales se utilizarían menos de 100. Por lo tanto, volquemos nuestra atención en las técnicas de hashing que relacionan una población grande de llaves posibles con pequeños espacios de dirección.

Método de módulo de división

También conocido como **hashing de residuo de división**, el método **de módulo** de **división** divide la llave entre el tamaño de archivo y utiliza el residuo más 1 para la dirección. Esto da el simple algoritmo de dispersión siguiente, donde `tamaño_lista` es el número de elementos en el archivo.

```
dirección = llave % tamaño_lista + 1
```

Aun cuando este algoritmo funciona con cualquier tamaño de lista, un número primo como tamaño de lista produce menos colisiones que otros tamaños en la lista. Por consiguiente, siempre que es posible, trate de hacer que el tamaño de archivo sea un número primo.

Conforme su pequeña compañía comience a crecer, se dará cuenta que pronto tendrá más de 100 empleados. Al planear para el futuro usted crea un nuevo sistema de numeración de empleados que manejará un millón de empleados. También decide que quiere proporcionar espacio de datos hasta para 300 empleados. El primer número primo mayor que 300 es 307. Por lo tanto, usted elige 307 como su tamaño de lista (archivo). Su nueva lista de empleados y algunas de sus direcciones relacionadas se muestran en la figura 13.9. En este caso, Vu Nguyen, con llave 121267, se relacionó a la dirección 003 porque 121267 % 307 = 2, y usted suma 1 al resultado para obtener la dirección (003).

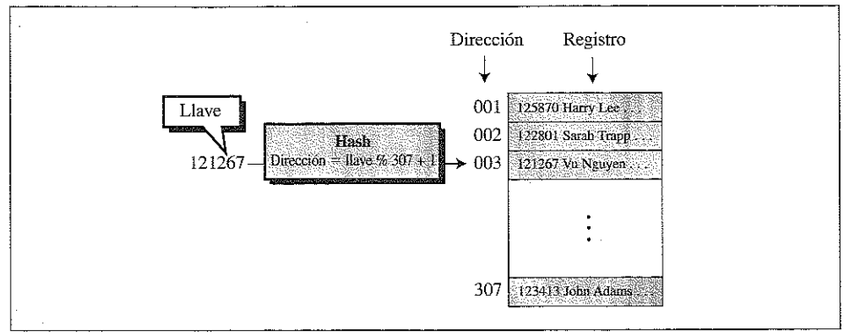


Figura 13.9 Módulo de división

Método de extracción de dígitos

Al utilizar el **hashing de extracción de dígitos,** los dígitos seleccionados se extraen de la llave y se utilizan como la dirección. Por ejemplo, al usar su número de empleado de seis dígitos para relacionar a una dirección de tres dígitos (000-999), usted puede seleccionar el primero, segundo, tercero y cuarto dígitos (a partir de la izquierda) y utilizar el término como la dirección. Usando las llaves de la figura 13.9, se relacionan con las direcciones como sigue:

```
125870 ⟶ 158
122801 ⟶ 128
121267 ⟶ 112
   .
123413 ⟶ 134
```

Otros métodos

Existen otros métodos populares, tales como el método de cuadrado medio, el método de doblado, el método rotacional y el método pseudoaleatorio. Dejamos la exploración de los mismos como ejercicios.

COLISIÓN

Generalmente, la población de llaves para una lista hashed es mayor que el número de registros en el archivo de datos. Por ejemplo, si usted tiene un archivo de 50 estudiantes para una clase en la cual los estudiantes se identifican mediante los últimos cuatro dígitos de su número de seguridad social, entonces hay 200 llaves posibles para cada elemento en el archivo (10 000 / 50). Debido a que hay muchas llaves para cada dirección en el archivo, existe la posibilidad de que más de una llave se relacione a la misma dirección en el archivo. Llamamos al conjunto de llaves que se direccionan a la misma dirección en nuestra lista **sinónimos**. El concepto de colisión se ejemplifica en la figura 13.10.

En la figura, cuando usted calcula la dirección para dos registros diferentes, obtiene la misma dirección (4). Evidentemente, los dos registros no pueden almacenarse en la misma dirección. Usted necesita resolver la situación como se explica en la siguiente sección.

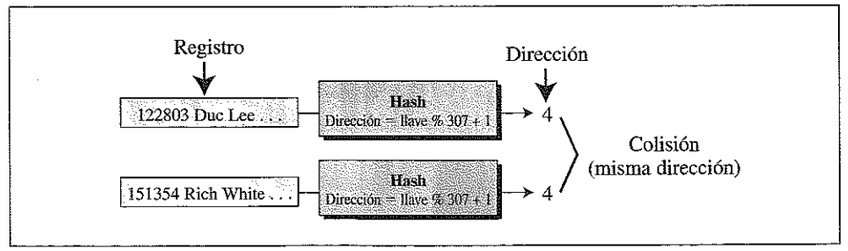


Figura 13.10 Colisión

Si los datos reales que usted inserta en su lista contienen dos o más sinónimos, tendrá colisiones. Una **colisión** es el evento que ocurre cuando un algoritmo de hashing produce una dirección para una llave de inserción y esa dirección ya está ocupada. La dirección producida por el algoritmo de hashing se conoce como **dirección base**. La parte del archivo que contiene todas las direcciones base se conoce como el **área principal**. Cuando dos llaves colindan en una dirección base, usted debe resolver la colisión al colocar una de las llaves y sus datos en otra localidad.

Resolución de colisiones

Con excepción del método directo, ninguno de los métodos que analizamos para hashing crea una relación uno a uno. Esto significa que cuando usted relaciona una llave nueva a una dirección, puede crear una colisión. Existen varios métodos para manejar colisiones, cada uno de ellos independiente del algoritmo de hashing. Es decir, cualquier método de hashing puede utilizarse con cualquier método de **resolución de colisiones**. En esta sección analizamos algunos de estos métodos.

Direccionamiento abierto El primer método de resolución de colisiones, la **resolución de direccionamiento abierto**, resuelve colisiones en el área principal. Cuando ocurre una colisión, las direcciones del área principal se buscan para un registro abierto o desocupado donde pueden colocarse los nuevos datos. Una estrategia simple para los datos que no pueden almacenarse en la dirección base es almacenarlos en la siguiente dirección (dirección base + 1). La figura 13.11 muestra cómo resolver la colisión de la figura 13.10 utilizando este método. El primer registro se almacena en la dirección 4 y el segundo se almacena en la dirección 5.

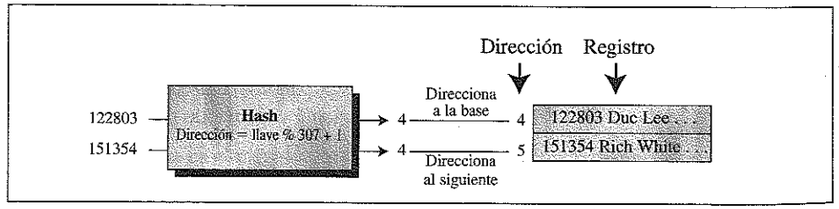


Figura 13.11 Resolución de direccionamiento abierto

Resolución de listas ligadas Una desventaja importante del direccionamiento abierto es que cada resolución de colisiones aumenta la probabilidad de colisiones futuras. Esta desventaja se elimina en otro método para resolución de colisiones, la **resolución de listas ligadas**. En este método, el primer registro se almacena en la dirección base, pero contiene un apuntador al segundo registro. La figura 13.12 muestra cómo resolver la situación que aparece en la figura 13.10.

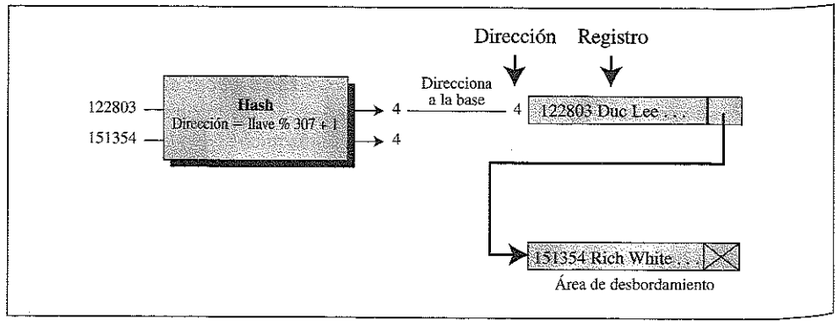


Figura 13.12 Resolución de listas ligadas

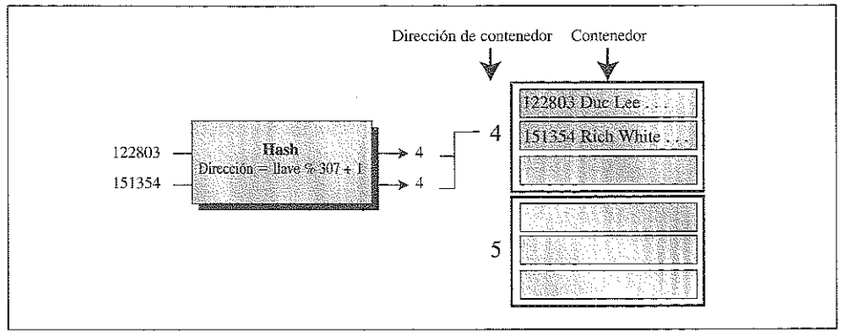


Figura 13.13 Resolución de hashing de contenedor

Hashing de contenedor Otro método para manejar el problema de la colisión es direccionar a **contenedores (buckets)**. La figura 13.13 muestra cómo resolver la colisión en la figura 13.10 utilizando el **hashing de contenedor**. Un contenedor es un nodo que puede acomodar más de un registro.

Métodos de combinación Hay varios métodos para resolver colisiones. Como se vio con los métodos de hashing, una implementación compleja a menudo utiliza múltiples métodos.

13.5 TEXTO *VERSUS* BINARIO

Antes de cerrar este capítulo, analizamos dos términos utilizados para clasificar los archivos: archivos de texto y archivos binarios. Un archivo almacenado en un dispositivo de almacenamiento es una secuencia de bits que puede ser interpretado por un programa de aplicación como un archivo de texto o un archivo binario según se muestra en la figura 13.14 y se explica a continuación.

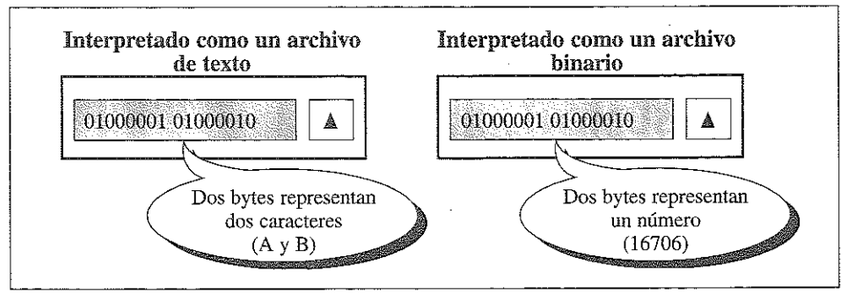


Figura 13.14 Interpretaciones de texto y binaria de un archivo

ARCHIVOS DE TEXTO

Un **archivo de texto** es un archivo de caracteres. No puede contener enteros, números de punto flotante o cualesquier otras estructuras de datos en su formato de memoria interna. Para almacenar estos tipos de datos, deben convertirse a sus formatos equivalentes de caracteres.

Algunos archivos sólo pueden utilizar tipos de datos de caracteres. Los más notables son cadenas de caracteres para teclados, monitores e impresoras. Ésta es la razón por la cual usted necesita funciones especiales para dar formato a los datos que entran a o salen de estos dispositivos.

Veamos un ejemplo. Cuando los datos (una cadena de caracteres) se envían a la impresora, la impresora toma ocho bits, los interpreta como un byte y los decodifica en el sistema de codificación de la impresora (ASCII o EBCDIC). Si el caracter pertenece a la categoría imprimible, éste se imprimirá; de lo contrario, se realiza alguna otra actividad, por ejemplo la impresión de un espacio. La impresora toma los siguientes ocho bits y repite el proceso. Esto se realiza hasta que una cadena de caracteres se agota.

ARCHIVOS BINARIOS

Un **archivo binario** es una colección de datos almacenados en el formato interno de la computadora. En esta definición, los datos pueden ser un entero, un número de punto flotante, un caracter o cualquier otra información estructurada (excepto un archivo).

A diferencia de los archivos de texto, los archivos binarios contienen datos que son significativos sólo si son interpretados adecuadamente por un programa. Si los datos son textuales, 1 byte se utiliza para representar un caracter. Pero si los datos son numéricos, 2 o más bytes se consideran un elemento de datos. Por ejemplo, suponga que está utilizando una computadora personal que usa dos bytes para almacenar un entero. En este caso, cuando usted lee o escribe un entero, dos bytes se interpretan como un entero.

13.6 TÉRMINOS LLAVE

acceso aleatorio
acceso secuencial
área principal
archivo
archivo binario
archivo de datos
archivo de informe de errores
archivo de texto
archivo de transacción
archivo hashed
archivo indexado
archivo invertido
archivo maestro nuevo
archivo maestro viejo
archivo secuencial
cinta
colisión
contenedor
disco
dispositivo de almacenamiento
dispositivo de almacenamiento auxiliar
dispositivo de almacenamiento secundario
dirección base
división de módulo
hashing de contenedor
hashing de extracción de dígitos
hashing de residuo (módulo) de división
hashing directo
índice
llave
lista ligada
método de acceso
resolución de colisiones
resolución de lista ligada
resolución de direccionamiento abierto
sinónimo

13.7 RESUMEN

- Un archivo es una colección de datos relacionados que se tratan como una unidad.
- Un registro en un archivo puede accederse en forma secuencial o aleatoria.
- En el acceso secuencial, cada registro debe accederse en secuencia, uno después de otro, de principio a fin.
- La actualización de un archivo secuencial requiere un archivo maestro nuevo, un archivo maestro viejo, un archivo de transacción y un archivo de informe de errores.
- En el acceso aleatorio, un registro puede accederse sin tener que recuperar ningún registro antes que él. La dirección del registro debe conocerse.
- Para el acceso aleatorio de un registro, puede utilizarse un archivo indexado que consiste en un archivo de datos y un índice.
- En el acceso aleatorio a archivos, el índice relaciona una llave con una dirección, la cual se utiliza después para recuperar el registro del archivo de datos.
- Un archivo hashed es un archivo de acceso aleatorio en el cual una función relaciona una llave con una dirección.
- En el hashing directo, la llave es la dirección y no se necesita ninguna manipulación de algoritmos.
- En el hashing de módulo de división, la llave se divide entre el tamaño de archivo. La dirección es el residuo más 1.
- En el hashing de extracción de dígitos, la dirección se compone de dígitos seleccionados desde la llave.
- Las llaves que se relacionan a la misma dirección se llaman sinónimos.
- Una colisión es un evento que ocurre cuando un algoritmo de hashing produce una dirección para una inserción y esa dirección ya está ocupada.
- Los métodos de resolución de colisiones mueven los datos direccionados que no pueden insertarse a una dirección nueva.
- El método de resolución de colisiones de direccionamiento abierto busca en el área principal para una dirección abierta (libre) para que se inserten los datos.
- El método de resolución de lista ligada utiliza un área separada para almacenar las colisiones y encadena a todos los sinónimos en una lista ligada.
- El hashing de contenedor es un método de resolución de colisiones que utiliza contenedores, nodos que acomodan múltiples ocurrencias de datos.
- Un archivo de texto es un archivo de caracteres.
- Un archivo binario son datos almacenados en el formato interno de la computadora.

13.8 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Qué es un archivo y cuál es su función?
2. ¿Cuáles son los dos tipos generales de métodos de acceso a archivos?
3. ¿Por qué se necesita un marcador EOF cuando se procesan los archivos secuencialmente?
4. ¿Cuál es la relación entre el archivo maestro nuevo y el archivo maestro viejo?
5. ¿Cuál es el propósito del archivo de transición en la actualización de un archivo secuencial?
6. Dé un ejemplo propio de una situación en la cual un archivo debe accederse en forma aleatoria.
7. Dé un ejemplo propio de una situación en la cual un archivo debe accederse en forma secuencial.
8. Describa la función de la dirección en un archivo al que se accedió en forma aleatoria.
9. ¿Cómo se relaciona el índice con el archivo de datos en los archivos indexados?
10. ¿Cuál es la relación entre la llave y la dirección en el hashing directo de un archivo?
11. ¿Cuál es la relación entre la llave y la dirección en el hashing de división de módulos de un archivo?
12. ¿Cuál es la relación entre la llave y la dirección en el hashing de extracción de dígitos de un archivo?
13. ¿Qué es una colisión?
14. Proporcione tres métodos de resolución de colisiones.
15. ¿Cómo funciona el método de resolución de colisiones de direccionamiento abierto?
16. Comente las dos áreas de almacenamiento requeridas para el método de resolución de colisiones de lista ligada.
17. ¿Cuál es la diferencia ante un archivo de texto y un archivo binario?

PREGUNTAS DE OPCIÓN MÚLTIPLE

18. Un archivo _____ puede accederse en forma aleatoria.
    a. secuencial
    b. indexado
    c. hashed
    d. b y c
19. Un archivo _____ puede accederse en forma secuencial.
    a. secuencial
    b. indexado
    c. hashed
    d. b y c
20. Cuando un archivo secuencial se actualiza, el archivo _____ obtiene la actualización más reciente.
    a. maestro nuevo
    b. maestro viejo
    c. de transacción
    d. de informe de errores
21. Cuando un archivo secuencial se actualiza, el archivo _____ contiene una lista de todos los errores que ocurren durante el proceso de actualización.
    a. maestro nuevo
    b. maestro viejo
    c. de transacción
    d. de informe de errores
22. Cuando un archivo secuencial se actualiza, el archivo _____ contiene los cambios a aplicar.
    a. maestro nuevo
    b. maestro viejo
    c. de transacción
    d. de informe de errores

23. Después de que se actualiza un archivo secuencial, el archivo _____ contiene los datos más actuales.
- **a.** maestro nuevo
- **b.** maestro viejo
- **c.** de transacción
- **d.** de informe de errores

24. Cuando un archivo secuencial necesita actualizarse, el archivo _____ en almacenamiento se vuelve el archivo _____.
- **a.** maestro nuevo; maestro viejo
- **b.** maestro viejo; maestro nuevo
- **c.** de transacción; maestro nuevo
- **d.** de transacción; maestro viejo

25. Si la llave del archivo de transacción es 20 y la llave del primer archivo maestro es 25, entonces usted _____.
- **a.** añade el nuevo registro al archivo maestro nuevo
- **b.** revisa el contenido del archivo maestro viejo
- **c.** elimina los datos
- **d.** escribe el registro del archivo maestro viejo en el archivo maestro nuevo

26. Si la llave del archivo de transacción es 20 con código de borrado y la llave del primer archivo maestro es 20, entonces usted _____.
- **a.** añade la transacción al archivo maestro nuevo
- **b.** revisa el contenido del archivo maestro viejo
- **c.** elimina los datos
- **d.** escribe el registro del archivo maestro viejo en el archivo maestro nuevo

27. Si un registro necesita accederse _____, un archivo indexado es el tipo de archivo más eficiente para usar.
- **a.** en forma secuencial
- **b.** en forma aleatoria
- **c.** en orden
- **d.** ninguno de los anteriores

28. Un archivo indexado consiste en _____.
- **a.** un archivo de datos secuencial
- **b.** un índice
- **c.** un archivo de datos aleatorio
- **d.** b y c

29. El índice de un archivo indexado tiene _____ campos.
- **a.** dos
- **b.** tres
- **c.** cuatro
- **d.** cualquier número de

30. Para acceder a un registro en forma aleatoria, se utiliza un(a) _____ en el índice para encontrar una dirección.
- **a.** dirección
- **b.** llave
- **c.** sinónimo
- **d.** a o b

31. En el método de dispersión o hashing _____, los dígitos seleccionados se extraen de la llave y se utilizan como la dirección.
- **a.** directo
- **b.** residuo de división
- **c.** módulo de división
- **d.** extracción de dígitos

32. En el método de dispersión o hashing _____, la llave se divide entre el tamaño de archivo y la dirección es el residuo más 1.
- **a.** directo
- **b.** módulo de división
- **c.** residuo de división
- **d.** extracción de dígitos

33. En el método de dispersión o hashing _____ ho hay sinónimos o colisiones.
- **a.** directo
- **b.** módulo de división
- **c.** residuo de división
- **d.** extracción de dígitos

34. Los(as) _____ son llaves que se direccionan a la misma localidad en el archivo de datos.
- **a.** colisiones
- **b.** contenedores
- **c.** sinónimos
- **d.** listas ligadas

35. Cuando un algoritmo de hashing produce una dirección para una llave de inserción y esa dirección ya está ocupada, se le llama _____.
- **a.** colisión
- **b.** sonda
- **c.** sinónimo
- **d.** lista ligada

36. La dirección producida por un algoritmo de hashing es la dirección _____.
- **a.** de sonda
- **b.** de sinónimo
- **c.** de colisión
- **d.** base

37. El área _____ es el área de archivo que contiene todas las direcciones base.
- **a.** de sonda
- **b.** ligada
- **c.** hash
- **d.** principal

38. En el método de resolución de colisiones _____, usted intenta poner en la localidad 124 los datos que no pueden colocarse en la localidad 123.
- **a.** direccionamiento abierto
- **b.** lista ligada
- **c.** hashing de contenedor
- **d.** a y b

39. En el método de resolución de colisiones _____, un nodo puede mantener múltiples piezas de datos.
- **a.** direccionamiento abierto
- **b.** lista ligada
- **c.** hashing de contenedor
- **d.** a y b

40. En el método de resolución de colisiones _____, tanto el área principal como el área de desbordamiento almacenan datos.
- **a.** direccionamiento abierto
- **b.** lista ligada
- **c.** hashing de contenedor
- **d.** a y b

41. Un archivo _____ es un archivo de caracteres.
- **a.** de texto
- **b.** binario
- **c.** de caracter
- **d.** hashed

EJERCICIOS

42. Dados el archivo maestro viejo y el archivo de transacción en la figura 13.15, encuentre el archivo maestro nuevo. Si hay algún error, también cree un archivo de error.

Archivo maestro viejo

14 John Wu	17.00
16 George Brown	18.00
17 Duc Lee	11.00
20 Li Nguyen	12.00
26 Ted White	23.00
31 Ioanne King	27.00
45 Bruce Wu	12.00
89 Mark Black	19.00
92 Betsy Yellow	14.00

Archivo de transacción

A 17	Martha Kent	17.00
D 20		
R 31		28.00
D 45		
A 89	Orva Gilbert	20.00

Figura 13.15 Ejercicio 42

43. Cree un archivo de índice para la tabla 13.1.

Llave	Nombre	Dep.
123453	John Adam	CIS
114237	Ted White	MAT
156734	Jimmy Lions	ING
093245	Sophie Grands	NEG
077654	Eve Primary	CIS
256743	Eva Lindens	ING
423458	Bob Bauser	ECO

Tabla 13.1 Ejercicio 43

44. Un archivo hashed utiliza un método de módulo de división con 41 como el divisor. ¿Cuál es la dirección para cada una de las llaves siguientes?
- **a.** 14232
- **b.** 12560
- **c.** 13450
- **d.** 15341

45. En el método de dispersión o hashing de cuadrado medio, la llave se eleva al cuadrado y la dirección se selecciona de la mitad del resultado. Utilice este método para seleccionar la dirección de cada una de las llaves siguientes. Use los dígitos 3, 4 y 5 (a partir de la izquierda).
- **a.** 142
- **b.** 125
- **c.** 134
- **d.** 153

46. En el método de dispersión o hashing de doblado, la llave se divide en partes. Las partes se suman para obtener la dirección. Utilice este método para encontrar la dirección de las llaves siguientes. Divida la llave en partes de dos dígitos y súmelas para encontrar la dirección.
- **a.** 1422
- **b.** 1257
- **c.** 1349
- **d.** 1532

47. En el método de dispersión o hashing de límite doblado, la llave se divide en partes. Las partes izquierda y derecha se invierten y se suman a la parte media para obtener la dirección. Utilice este método para encontrar la dirección de las llaves siguientes. Divida la llave en tres partes de dos dígitos, invierta los dígitos en la primera y tercera partes y luego sume las partes para obtener la dirección.
- **a.** 142234
- **b.** 125711
- **c.** 134919
- **d.** 153213

48. Encuentre la dirección de las llaves siguientes utilizando el método de módulo de división y un archivo de tamaño 411. Si hay una colisión, utilice el direccionamiento abierto para resolverlo. Dibuje una figura para mostrar la posición de los registros.
- **a.** 10278
- **b.** 08222
- **c.** 20553
- **d.** 17256

49. Repita el ejercicio 48 usando la resolución de lista ligada.

Bases de datos

Comenzamos por definir un sistema de administración de bases de datos (DBSM) y analizar sus componentes. Luego presentamos los tres niveles de arquitectura para un DBMS. Nos concentramos en el modelo de bases de datos relacionales con ejemplos de sus operaciones. Luego analizamos un lenguaje (el lenguaje de consultas estructurado o SQL) que opera en las bases de datos relacionales. Finalmente, estudiamos brevemente otros modelos de bases de datos.

Una **base de datos** es una colección de datos que de una manera lógica, pero no necesariamente física, es coherente. Normalmente, debe haber algún significado inherente a los datos para justificar la creación de la base de datos.

14.1 SISTEMA DE ADMINISTRACIÓN DE BASES DE DATOS

Un **sistema de administración de bases de datos** (**DBMS**: *Database management system*) define, crea y mantiene una base de datos. El DBMS además permite a los usuarios acceso controlado a los datos en la base de datos. Un DBMS es una combinación de cinco componentes: hardware, software, datos, usuarios y procedimientos (figura 14.1).

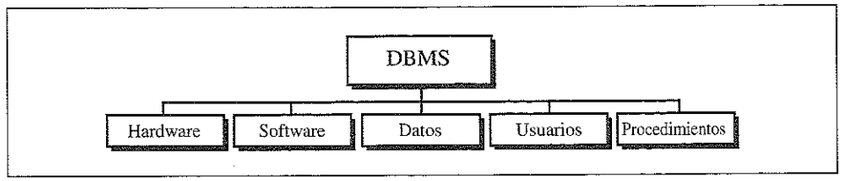


Figura 14.1 Componentes DBMS

Hardware

El **hardware** es el sistema de cómputo físico que permite acceso físico a los datos. Por ejemplo, las terminales de usuario, el disco duro, la computadora principal y las estaciones de trabajo se consideran parte del hardware en un DBMS.

Software

El **software** es el programa que permite a los usuarios acceder, mantener y actualizar los datos físicos. Además, el software controla cuál usuario puede tener acceso a qué parte de los datos en la base de datos.

Datos

Los datos en una base de datos se almacenan físicamente en los dispositivos de almacenamiento. En una base de datos los datos son una entidad separada del software que accede a los mismos. Esta separación permite la organización para cambiar el software sin tener que cambiar los datos físicos o la manera en que éstos se almacenan. Si una organización ha decidido utilizar un DBMS, entonces toda la información requerida por la organización debe mantenerse bajo una entidad, para que sea accesible por el software en el DBMS.

Usuarios

El término **usuarios** en un DBMS tiene un amplio significado. Podemos dividir a los usuarios en dos categorías: usuarios finales y programas de aplicación.

Usuarios finales Los **usuarios finales** son aquellas personas que pueden acceder a la base de datos directamente para obtener información. Hay dos tipos de usuarios finales: el administrador de bases de datos (DBA) y el usuario normal. El administrador de bases de datos tiene el nivel máximo de privilegios. Puede controlar a los otros usuarios y su acceso al DBMS. Puede otorgar algunos de sus privilegios a alguien más pero conserva la capacidad para revocarlas en cualquier momento. Un usuario normal, por otro lado, sólo puede utilizar parte de la base de datos y tiene acceso limitado.

Programas de aplicación Los demás usuarios de los datos en una base de datos son los **programas de aplicación**. Las aplicaciones necesitan tener acceso a los datos y procesarlos. Por ejemplo, un programa de aplicación de nómina necesita tener acceso a parte de los datos en una base de datos para crear cheques de pago a fin de mes.

Procedimientos

El último componente de un DBMS es una serie de procedimientos o reglas que deben definirse claramente y que deben seguir los usuarios de la base de datos.

14.2 ARQUITECTURA

El **Comité de planeación y requisitos de estándares** del Instituto Nacional Norteamericano de Estándares (ANSI/SPARC) ha establecido una arquitectura de tres niveles para un DBMS: interno, conceptual y externo (figura 14.2).

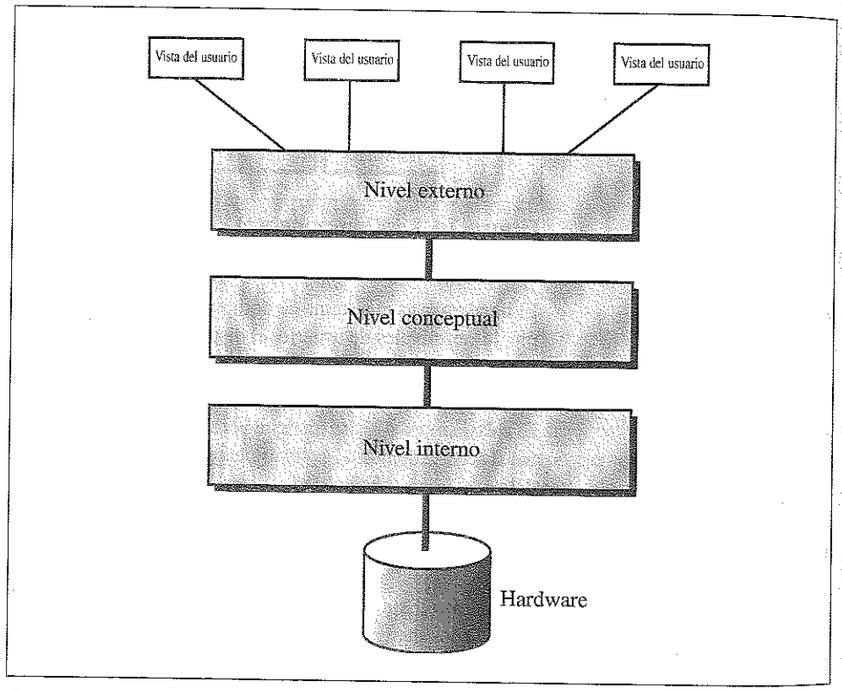


Figura 14.2 Arquitectura de bases de datos

NIVEL INTERNO

El **nivel interno** determina dónde se almacenan realmente los datos en el dispositivo de almacenamiento. Este nivel trata con métodos de acceso de bajo nivel y cómo se transfieren los bytes hacia y desde el dispositivo de almacenamiento. En otras palabras, el nivel interno interactúa directamente con el hardware.

NIVEL CONCEPTUAL

El **nivel conceptual**, o comunitario, define el punto de vista lógico de los datos. En este nivel se define el modelo de datos y los diagramas de esquemas. Las funciones principales del DBMS están en este nivel. El DBMS cambia la vista interna de los datos a la vista externa de los mismos que el usuario necesita ver. El nivel conceptual es un intermediario y libera a los usuarios del manejo del nivel interno.

NIVEL EXTERNO

El **nivel externo** interactúa directamente con el usuario (usuarios finales o programas de aplicación). Cambia los datos que llegan del nivel conceptual a un formato y vista que son conocidos por el usuario.

14.3 MODELOS DE BASES DE DATOS

Un modelo de base de datos define el diseño lógico de los datos. El modelo también describe las relaciones entre distintas partes de los datos. En la historia del diseño de bases de datos, tres modelos han estado en uso: el modelo jerárquico, el modelo de red y el modelo relacional.

MODELO JERÁRQUICO

En un modelo jerárquico, los datos están organizados como un árbol invertido. Cada entidad tiene sólo un padre, pero puede tener varios hijos. En la parte superior de la jerarquía, el árbol es una sola entidad, la cual se llama raíz. La figura 14.3 muestra una vista lógica del modelo jerárquico. Como el modelo jerárquico es obsoleto, no es necesario ningún análisis de este modelo.

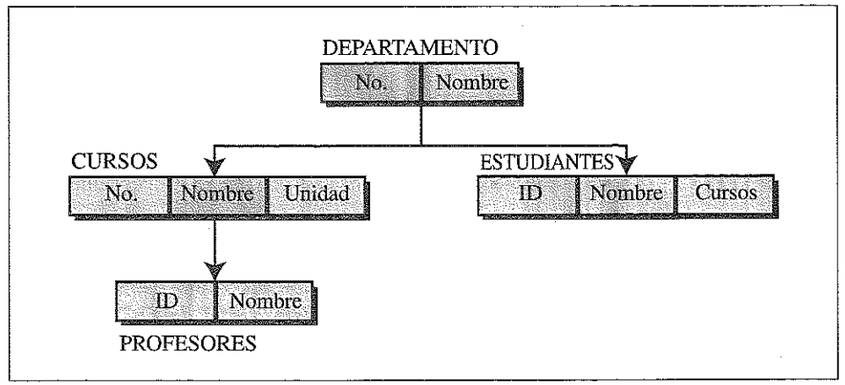


Figura 14.3 Modelo jerárquico

MODELO DE RED

En un **modelo de red**, las entidades se organizan en un grafo, donde se puede tener acceso a algunas entidades a través de varios caminos (figura 14.4). No hay una jerarquía. Este modelo también es obsoleto y no necesita un análisis posterior.

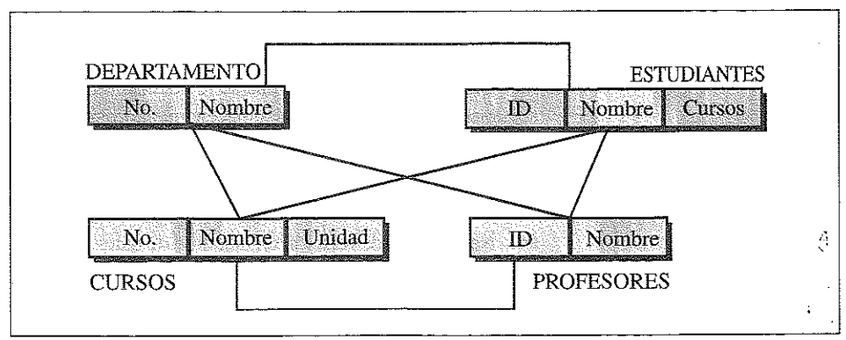


Figura 14.4 Modelo de red

MODELO RELACIONAL

En un **modelo relacional,** los datos se organizan en tablas bidimensionales llamadas relaciones. No hay una estructura jerárquica o de red impuesta en los datos. No obstante, las tablas o relaciones están relacionadas entre sí, como se verá en breve (figura 14.5).

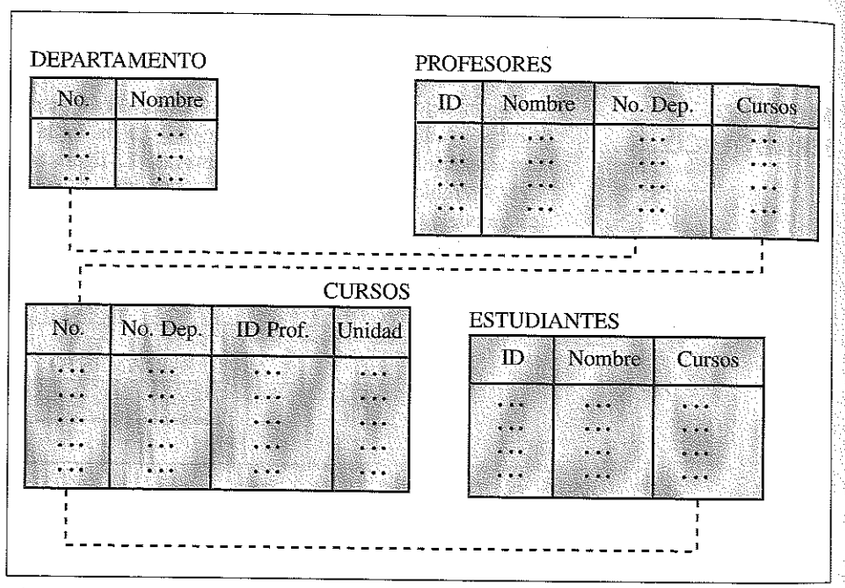


Figura 14.5 Modelo relacional

El modelo relacional es uno de los modelos más comunes en uso en la actualidad y dedicamos la mayor parte de este capítulo al mismo. En la última sección, estudiamos brevemente los otros dos modelos comunes que se derivan del modelo relacional: el modelo distribuido y el modelo orientado a objetos.

14.4 MODELO RELACIONAL

Continuamos nuestro análisis de las bases de datos con el modelo más popular, el **sistema de bases de datos relacionales** (**RDBMS**: *relational database management system*). En este modelo, los datos (universo del discurso) se representan mediante una serie de **relaciones**.

RELACIÓN

Una relación, en apariencia, es una tabla bidimensional. El RDBMS organiza los datos de manera que la vista externa sea una serie de relaciones o tablas. Esto no significa que los datos se almacenen como tablas; el almacenamiento físico de los datos es independiente de la forma en que éstos están lógicamente organizados. La figura 14.6 muestra un ejemplo de una relación.

Una relación en un RDBMS tiene las características siguientes:

- **Nombre.** Cada relación en una base de datos relacional debe tener un nombre único entre otras relaciones.
- **Atributos.** Cada columna en una relación se llama atributo. Los atributos son los encabezados de las columnas en la tabla. Cada atributo da significado a los datos almacenados bajo él.

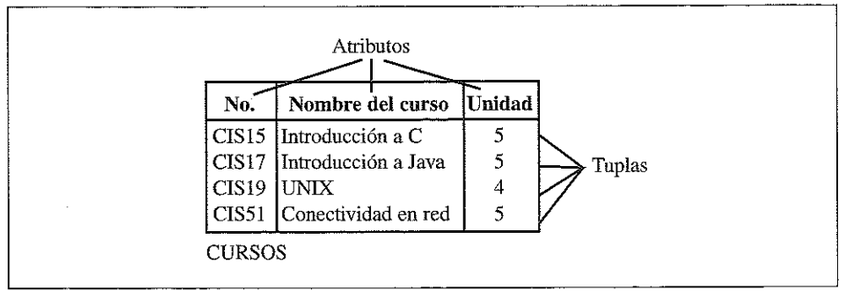


Figura 14.6 Relación

Cada columna en la tabla debe tener un nombre que sea único en el ámbito de la relación. El número total de atributos para una relación se conoce como el grado de la relación. Por ejemplo, en la figura 14.6, la relación tiene un grado de 3. Observe que los nombres de los atributos no se almacenan en la base de datos; el nivel conceptual utiliza los atributos para dar significado a cada columna.

- **Tuplas.** Cada fila en una relación se conoce como **tupla.** Una tupla define una colección de valores de atributos. El número total de filas en una relación se llama **cardinalidad** de la relación. Observe que la cardinalidad de una relación cambia cuando se añaden o eliminan tuplas. Esto vuelve dinámica a la base de datos.

14.5 OPERACIONES CON RELACIONES

En una base de datos relacional, podemos definir varias operaciones para crear nuevas relaciones además de las existentes. Definimos nueve operaciones en esta sección: inserción, eliminación, actualización, selección, proyección, juntura, unión, intersección y diferencia.

INSERCIÓN

La **operación de inserción** es una operación unaria; se aplica a una sola relación. La operación inserta una nueva tupla en la relación. La figura 14.7 muestra un ejemplo de la operación de inserción. Un nuevo curso (CIS52) se ha insertado (agregado) a la relación.

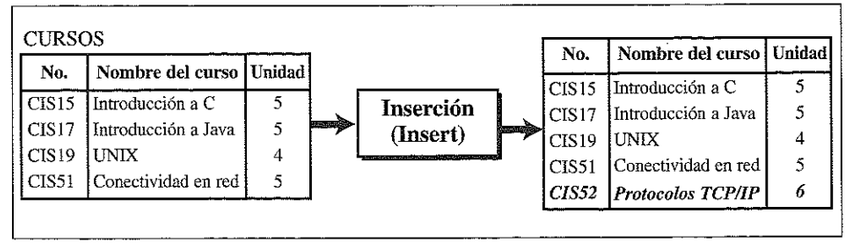


Figura 14.7 Operación de inserción

ELIMINACIÓN

La **operación de eliminación** también es una operación unaria. La operación elimina una tupla definido por un criterio de la relación. La figura 14.8 muestra un ejemplo de la operación de eliminación. El curso CIS19 se eliminó.

CURSOS

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

→ Eliminación (Delete) →

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

Figura 14.8 Operación de eliminación

ACTUALIZACIÓN

La **operación de actualización** también es una operación unaria; se aplica a una sola relación. La operación cambia el valor de algunos atributos de una tupla. La figura 14.9 muestra un ejemplo de la operación de actualización. El número de unidades para CIS51 se ha actualizado (cambiado) de 5 a 6.

CURSOS

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

→ Actualización (Update) →

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	**6**
CIS52	Protocolos TCP/IP	6

Figura 14.9 Operación de actualización

SELECCIÓN

La **operación de selección** es una operación unaria que se aplica a una sola relación y crea otra relación. Las tuplas (filas) de la relación resultante son un subconjunto de las tuplas de la relación original. Esta operación utiliza algunos criterios para seleccionar algunas de las tuplas de la relación original. El número de atributos (columnas) en esta operación permanece igual. La figura 14.10 muestra un ejemplo de la operación de selección. En esta figura, hay una relación que muestra los cursos ofrecidos por un pequeño departamento. La operación de selección permite al usuario seleccionar sólo los cursos de cinco unidades.

CURSOS

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

→ Selección (Select) →

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS51	Conectividad en red	5

Figura 14.10 Operación de selección

PROYECCIÓN

La **operación de proyección** también es una operación unaria; se aplica a una sola relación y crea otra relación. Los atributos (columnas) en la relación resultante son un subconjunto de los atributos de la relación original. La operación de proyección crea una relación en la cual cada tupla tiene menos atributos. El número de tuplas (filas) en esta operación sigue siendo el mismo. La figura 14.11 muestra un ejemplo de la operación de proyección que crea una relación con sólo dos columnas.

CURSOS

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

→ Proyección (Project) →

No.	Unidad
CIS15	5
CIS17	5
CIS19	4
CIS51	5
CIS52	6

Figura 14.11 Operación de proyección

JUNTURA

La **operación de juntura** es una operación binaria; toma dos relaciones y las combina con base en atributos comunes. Esta operación es muy compleja y tiene muchas variantes. En la figura 14.12 aparece un ejemplo muy simple en el cual la relación de CURSOS se combina con la relación de IMPARTIDOS POR para crear una relación que muestra toda la información sobre los cursos, incluyendo los nombres de los profesores que los imparten. En este caso, el atributo común es el número de curso (No.).

CURSOS

No.	Nombre del curso	Unidad
CIS15	Introducción a C	5
CIS17	Introducción a Java	5
CIS19	UNIX	4
CIS51	Conectividad en red	5
CIS52	Protocolos TCP/IP	6

IMPARTIDO POR

No.	Profesor
CIS15	Lee
CIS17	Lu
CIS19	Walter
CIS51	Lu
CIS52	Lee

→ Juntura (Join) →

No.	Nombre del curso	Unidad	Profesor
CIS15	Introducción a C	5	Lee
CIS17	Introducción a Java	5	Lu
CIS19	UNIX	4	Walter
CIS51	Conectividad en red	5	Lu
CIS52	Protocolos TCP/IP	6	Lee

Figura 14.12 Operación de juntura

UNIÓN

La **operación de unión** también es una operación binaria; toma dos relaciones y crea una nueva relación. Sin embargo, hay una restricción en las dos relaciones; éstas deben tener los mismos atributos. La operación de unión, según se define en la teoría de conjuntos, crea una nueva relación en la cual cada tupla está ya sea en la primera relación, en la segunda o en ambas. Por ejemplo, la figura 14.13 exhibe dos relaciones. En la esquina superior izquierda

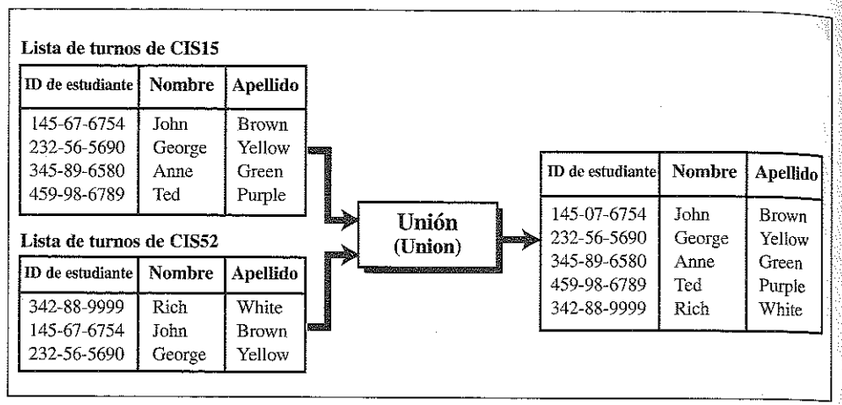


Figura 14.13 Operación de unión

está la lista de turnos para CIS15; en la esquina inferior izquierda está la lista para CIS52. El resultado es una relación con información sobre los estudiantes que toma ya sea CIS15, CIS52 o ambos.

INTERSECCIÓN

La **operación de intersección** también es una operación binaria; toma dos relaciones y crea una nueva relación. Al igual que la operación de unión, las dos relaciones deben tener los mismos atributos. La operación de intersección, como lo define la teoría de conjuntos, crea una nueva relación en la cual cada tupla es un miembro de ambas relaciones. Por ejemplo, la figura 14.14 presenta dos relaciones de entrada. El resultado de la operación de intersección es una relación con información sobre los estudiantes que toman tanto CIS15 como CIS52.

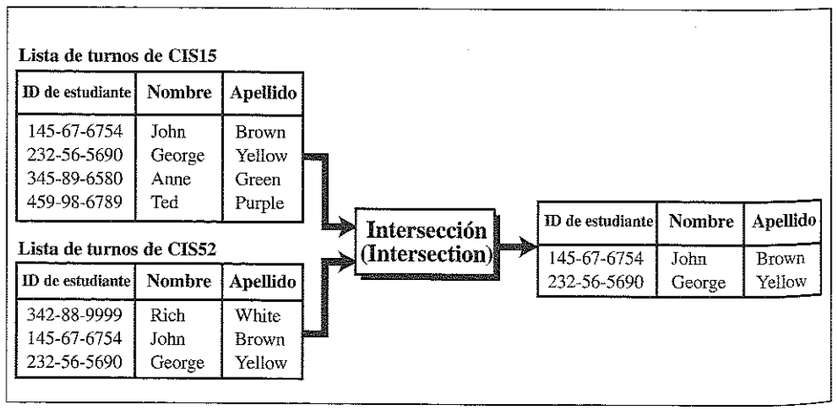


Figura 14.14 Operación de intersección

DIFERENCIA

La **operación de diferencia** también es una operación binaria. Se aplica a dos relaciones con los mismos atributos. Las tuplas en la relación resultante son aquellas que están en la primera relación pero no en la segunda. Por ejemplo, la figura 14.15 muestra dos relaciones de entrada. El resultado de la operación de diferencia es una relación con información sobre los estudiantes que toman CIS15 pero no CIS52.

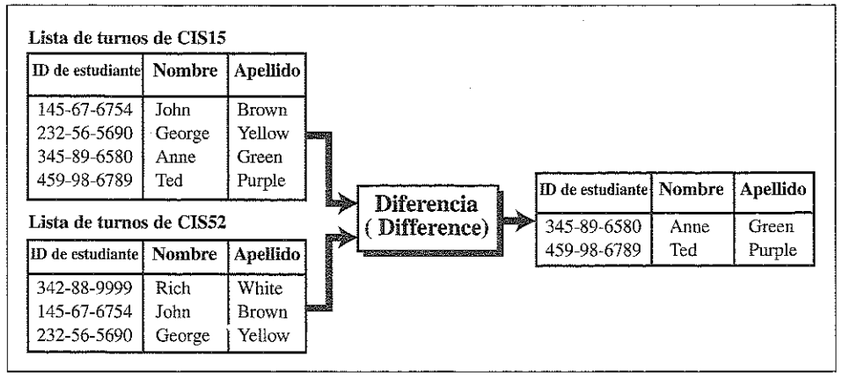


Figura 14.15 Operación de diferencia

14.6 LENGUAJE DE CONSULTAS ESTRUCTURADO

El **lenguaje de consultas estructurado** (SQL: *structured query language*) es el lenguaje estandarizado por el Instituto Nacional Norteamericano de Estándares (ANSI) y la Organización Internacional para la Estandarización (ISO) para usar en las bases de datos relacionales. Es un lenguaje declarativo (no de procedimientos), lo cual significa que los usuarios declaran lo que quieren sin tener que escribir un procedimiento paso a paso. El lenguaje SQL primero se implementó por Oracle Corporation en 1979; desde entonces se han liberado nuevas versiones de SQL.

En esta sección se definen algunas instrucciones comunes en el lenguaje SQL que se relacionan con las operaciones definidas en la sección anterior. De ninguna manera es un tutorial para el lenguaje SQL. Para obtener más información, consulte libros sobre SQL.

INSTRUCCIONES

Las instrucciones siguientes están relacionadas con las operaciones que definimos.

Insert

La operación insert (insertar) utiliza el formato siguiente. La cláusula de valores define todos los valores de atributos para la tupla correspondiente a insertar.

```
insert into    NOMBRE-RELACION
values         (. . . , . . . , . . .)
```

Por ejemplo, la operación de inserción en la figura 14.7 puede implementarse en SQL utilizando:

```
insert into    CURSOS
values         ("CIS52", "Protocolos TCP/IP", 6)
```

Observe que los valores de cadena están entrecomillados; los valores numéricos no lo están.

Delete

La operación delete (eliminar) utiliza el siguiente formato. Los criterios para la eliminación se definen en la cláusula where.

```
delete from NOMBRE-RELACION
where  criterios
```

Por ejemplo, la operación de eliminación en la figura 14.8 puede implementarse en SQL utilizando:

```
delete from CURSOS
where No ="CIS19"
```

Update

La operación update (actualizar) utiliza el formato siguiente. El atributo a cambiar se define en la cláusula set. Los criterios para la actualización se definen en la cláusula where.

```
update  NOMBRE-RELACION
set atributo1 = valor1  atributo2 = valor2, . . .
where   criterios
```

Por ejemplo, la operación de actualización en la figura 14.9 puede implementarse en SQL usando:

```
update     CURSOS
set    Unidad = 6
where No = "CIS51"
```

Select

La operación select (seleccionar) utiliza el formato siguiente. El asterisco significa que todos los atributos se eligen.

```
select      *
from   NOMBRE-RELACION
where    criterios
```

Por ejemplo, la operación de selección en la figura 14.10 puede implementarse en SQL usando:

```
select      *
from     CURSOS
where    Unidad = 5
```

Project

La operación project (proyectar) utiliza el formato siguiente. Los nombres de las columnas para la nueva relación se listan de manera explícita.

```
select  lista-atributos
from NOMBRE-RELACION
```

Por ejemplo, la operación de proyección en la figura 14.11 puede implementarse en SQL mediante:

```
select No. Unidad
from CURSOS
```

Join

La operación join (juntar) emplea el siguiente formato. La lista de atributos es la combinación de atributos de las dos relaciones de entrada; los criterios definen explícitamente los atributos utilizados como atributos comunes.

```
select     lista-atributos
from RELACION1,   RELACION2
where criterios
```


Por ejemplo, la operación de juntura en la figura 14.12 puede implementarse en SQL utilizando:

```
select No. Nombre del curso. Unidad. Profesor
from CURSOS. IMPARTIDOS-POR
where CURSOS.No = IMPARTIDOS-POR.No;
```

Union

La operación union (unir) utiliza el formato siguiente. Otra vez, el asterisco significa que se seleccionan todos los atributos.

```
select     *
from    RELACION1
union
select      *
from    RELACION2
```

Por ejemplo, la operación de unión de la figura 14.13 puede implementarse en SQL mediante:

```
select      *
from     Lista-turnos-CIS15
union
select      *
from     Lista-turnos-CIS52;
```

Intersection

La operación intersection (intersección) usa el formato siguiente. De nuevo, el asterisco significa que se seleccionan todos los atributos.

```
select     *
from    RELACION1
intersection
select     *
from    RELACION2
```

Por ejemplo, la operación de intersección de la figura 14.14 puede implementarse en SQL utilizando:

```
select     *
from     Lista-turnos-CIS15
intersection
select     *
from     Lista-turnos-CIS52
```

Difference

La operación difference (diferencia) utiliza el siguiente formato. Nuevamente, el asterisco significa que se seleccionan todos los atributos.

```
select     *
from    RELACION1
minus
select     *
from    RELACION2
```

Por ejemplo, la operación de diferencia de la figura 14.15 puede implementarse en SQL mediante:

```
select    *
from      Lista-turnos-CIS15
minus
select    *
from      Lista-turnos-CIS52
```

Combinación de instrucciones

El lenguaje SQL le permite combinar las instrucciones anteriores para extraer información más compleja desde una base de datos.

14.7 OTROS MODELOS DE BASES DE DATOS

La base de datos relacional es uno de los modelos más comunes de hoy en día. Los otros dos modelos comunes son las bases de datos distribuidas y las bases de datos orientadas a objetos. Analizamos brevemente estos dos modelos en esta sección y dejamos una exploración más profunda para libros sobre bases de datos.

BASES DE DATOS DISTRIBUIDAS

El modelo de **bases de datos distribuidas** en realidad no es un modelo nuevo. Está basado en el modelo relacional. Sin embargo, los datos se almacenan en varias computadoras que se comunican a través de Internet (o alguna red privada de área amplia). Cada computadora (o sitio, como se le llama) mantiene parte o toda la base de datos. En otras palabras, los datos están ya sea fragmentados, con cada fragmento almacenado en un sitio, o duplicados en cada sitio.

Bases de datos distribuidas fragmentadas

En una **base de datos distribuida fragmentada** los datos se localizan. Los datos utilizados localmente se almacenan en el sitio correspondiente. Sin embargo, esto no significa que un sitio no puede tener acceso a los datos almacenados en otro sitio. El acceso es principalmente local pero de vez en cuando es global. Aunque cada sitio tiene control completo sobre sus datos locales, hay un control global a través de Internet. Por ejemplo, una compañía farmacéutica puede tener múltiples sitios en muchos países. Cada sitio tiene una base de datos con información sobre sus propios empleados, pero un departamento central de personal tiene el control de todas las bases de datos.

Bases de datos distribuidas replicadas

En una **base de datos distribuida replicada**, cada sitio mantiene una réplica exacta de otro sitio. Cualquier modificación en los datos almacenados en un sitio se replica exactamente en cada sitio. La razón para tener una base de datos semejante es la seguridad. Si el sistema en un sitio falla, los usuarios en este sitio pueden tener acceso a los datos de otro sitio.

BASES DE DATOS ORIENTADAS A OBJETOS

La base de datos relacional tiene una vista específica de datos que se basa en la naturaleza de las bases de datos relacionales (tuplas y atributos). La unidad de datos más pequeña en una base de datos relacional es la intersección de una tupla y un atributo. Sin embrago, actualmente algunas aplicaciones necesitan buscar datos en otra forma. A algunas aplicaciones les gusta ver los datos como una estructura (capítulo 11), por ejemplo un registro hecho de campos.

Una **base de datos orientada a objetos** trata de mantener las ventajas del modelo relacional y al mismo tiempo permite que las aplicaciones accedan a los datos estructurados. En una base de datos orientada a objetos, se definen los objetos y sus relaciones. Además, cada objeto puede tener atributos que pueden expresarse como campos.

Por ejemplo, en una organización, uno puede definir tipos de objetos tales como empleados, departamentos y clientes. La clase empleados puede definir los atributos de un objeto empleado (nombre, apellido, número del seguro social, salario, etc.) y cómo se puede tener acceso a los mismos. El objeto departamento puede definir los atributos del departamento y cómo puede accederse a ellos. Asimismo, la base de datos puede crear una relación entre un objeto empleado y un objeto departamento (un empleado trabaja en un departamento).

14.8 TÉRMINOS CLAVE

base de datos
base de datos distribuida duplicada
base de datos distribuida fragmentada
base de datos distribuida (replicada)
base de datos orientada a objetos
cardinalidad
Comité de planeación y requisitos de estándares (SPARC)
datos (DBMS)
hardware
lenguaje de consultas estructurado (SQL)
modelo de red
modelo jerárquico
modelo relacional
nivel conceptual
nivel externo
nivel interno
operación difference (diferencia)
operación intersection (intersección)
operación join (juntura)
operación project (proyección)
operación select (seleccionar)
operación union (unión)
operación update (actualizar)
procedimiento
programa de aplicación
relación
sistema de administración de bases de operación difference
sistema de administración de bases de datos relacionales (RDBMS)
software
tupla
usuario
usuario final

14.9 RESUMEN

- Una base de datos es una colección de datos que de manera lógica, pero no necesariamente física, es coherente.
- Un sistema de administración de bases de datos (DBMS) define, crea y mantiene una base de datos y permite el acceso controlado a los usuarios.
- Un DBMS se compone de hardware, software, datos, usuarios y procedimientos.
- Los usuarios de DBMS pueden ser personas o programas de aplicación.
- Un DBMS tiene tres niveles: interno, conceptual y externo.
- El nivel interno de un DBMS interactúa directamente con el hardware y se ocupa de los métodos de acceso de bajo nivel y la transferencia de bytes hacia y desde el dispositivo de almacenamiento.
- El nivel conceptual de una DBMS define la vista lógica de los datos así como el modelo de datos y los diagramas de esquema.
- El nivel externo de un DBMS interactúa directamente con el usuario.
- El sistema de administración de bases de datos relacionales (RDBMS) es el único modelo de base de datos de amplio uso hoy en día. Los modelos jerárquico y de red son obsoletos.
- Una relación puede pensarse como una tabla bidimensional.
- Cada columna en una relación se llama atributo. El número de atributos en una relación es su grado.
- Cada fila en una relación se llama tupla. El número de filas en una relación es su cardinalidad.
- Nueve operaciones pueden realizarse con las relaciones.
- Una operación que actúa sobre una relación es un operador unitario. Los operadores unitarios incluyen las operaciones de inserción, eliminación, actualización, selección y proyección.
- Una operación que actúa sobre dos relaciones es un operador binario. Los operadores binarios incluyen las operaciones de juntura, unión, intersección y diferencia.
- El lenguaje de consultas estructurado (SQL) es el lenguaje estandarizado por el ANSI y la ISO para usarse en las bases de datos relacionales.

14.10 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuáles son los cinco componentes indispensables de un DBMS?
2. Mencione los dos tipos de usuarios de un DBMS.
3. ¿Cómo se relacionan los tres niveles de un DBMS entre sí?
4. ¿Cuáles son los tres modelos de bases de datos? ¿Cuáles de ellos son populares en la actualidad?
5. ¿Qué es una relación en un RDBMS?
6. En una relación, ¿qué es un atributo? ¿Qué es una tupla?
7. ¿Cuál es la diferencia entre una operación unaria y una operación binaria?
8. Mencione las operaciones unarias que se aplican a las relaciones en un RDBMS.
9. Mencione las operaciones binarias que se aplican a las relaciones en un RDBMS.
10. ¿Cuál es la diferencia entre las operaciones de inserción y eliminación?
11. ¿Cuál es la diferencia entre las operaciones de actualización y selección?
12. ¿Cuál es la función de la operación de proyección?
13. ¿Qué tiene en común la relación de salida de la operación de juntura con las relaciones de entrada?
14. ¿Cuál es la diferencia entre la operación de unión y la operación de intersección?
15. ¿Cuál es la función de la operación de diferencia?
16. ¿Qué es SQL?
17. En SQL, ¿cómo se sabe en cuáles relaciones están actuando los operadores?

PREGUNTAS DE OPCIÓN MÚLTIPLE

18. Un DBMS _____ una base de datos.
    a. define
    b. crea
    c. mantiene
    d. todos los anteriores
19. El código DBMS que permite al usuario tener acceso, mantener y actualizar es _____.
    a. el hardware
    b. los datos
    c. el software
    d. el usuario
20. Los componentes del DBMS tales como la computadora y los discos duros que permiten acceso físico a los datos se conocen como _____.
    a. hardware
    b. software
    c. usuarios
    d. programas de aplicación
21. El _____ de un DBMS puede ser un administrador de bases de datos o una persona que necesita tener acceso a la base de datos.
    a. usuario final
    b. programa de aplicación
    c. programador
    d. b y c
22. Tanto las personas como _____ pueden considerarse usuarios de una base de datos.
    a. los datos
    b. el software
    c. los programas de aplicación
    d. el hardware
23. En una arquitectura DBMS de tres niveles, la capa que interactúa directamente con el hardware es el nivel _____.
    a. externo
    b. conceptual
    c. interno
    d. físico
24. En una arquitectura DBMS de tres niveles, el nivel _____ determina dónde se almacenan los datos en realidad en el dispositivo de almacenamiento.
    a. externo
    b. conceptual
    c. interno
    d. físico
25. El nivel _____ de una arquitectura DBMS de tres niveles define la vista lógica de los datos.
    a. externo
    b. conceptual
    c. interno
    d. físico
26. El modelo de datos y el esquema de un DBMS a menudo se definen en el nivel _____.
    a. externo
    b. conceptual
    c. interno
    d. físico

27. En una arquitectura DBMS de tres niveles, el nivel _____ interactúa directamente con los usuarios.
    a. externo
    b. conceptual
    c. interno
    d. físico
28. De los diversos modelos de bases de datos, el modelo _____ es el que más prevalece hoy en día.
    a. jerárquico
    b. de red
    c. relacional
    d. de lista ligada
29. El modelo de bases de datos _____ acomoda sus datos en forma de un árbol invertido.
    a. jerárquico
    b. de red
    c. relacional
    d. de lista ligada
30. En el modelo de bases de datos _____, cada entidad puede accederse a través de múltiples caminos.
    a. jerárquico
    b. de red
    c. relacional
    d. de lista ligada
31. La relación es un conjunto de datos organizado lógicamente como una tabla _____.
    a. unidimensional
    b. bidimensional
    c. tridimensional
    d. de cualquier dimensión
32. Cada columna en una relación se llama _____.
    a. atributo
    b. tupla
    c. unión
    d. lista ligada
33. El grado de una relación es el número de _____ en la relación.
    a. atributos
    b. tuplas
    c. uniones
    d. posiciones
34. Cada fila en una relación se llama _____.
    a. atributo
    b. tupla
    c. unión
    d. posición
35. Si una relación tiene cinco filas, entonces su _____ es cinco.
    a. diferencia
    b. cardinalidad
    c. duplicidad
    d. relatividad
36. Un operador unario se aplica a la(s) relación(es) _____ y crea una salida de _____ relación(es).
    a. una; una
    b. una; dos
    c. dos; una
    d. dos; dos
37. Un operador binario se aplica a la(s) relación(es) _____ y crea una salida de _____ relación(es).
    a. una; una
    b. una; dos
    c. dos; una
    d. dos; dos
38. La operación unaria de _____ siempre da como resultado una relación que tiene exactamente una fila más que la relación original.
    a. inserción
    b. eliminación
    c. actualización
    d. selección
39. Si se quiere cambiar el valor de un atributo de una tupla, se utiliza la operación de _____.
    a. proyección
    b. juntura
    c. actualización
    d. selección
40. Si usted tiene tuplas en una relación que contiene información de los estudiantes y quiere sólo las tuplas de las estudiantes mujeres, puede utilizar la operación de _____.
    a. proyección
    b. juntura
    c. actualización
    d. selección
41. La operación que toma dos relaciones y las combina con base en atributos comunes es la operación de _____.
    a. juntura
    b. proyección
    c. unión
    d. intersección
42. Si se necesita eliminar un atributo en una relación, puede utilizarse la operación de _____.
    a. juntura
    b. proyección
    c. unión
    d. intersección

43. Usted quiere crear una relación llamada Nueva que contenga tuplas pertenecientes tanto a la relación A como a la relación B. Para hacerlo, puede utilizar la operación de _____.
- **a.** selección
- **b.** unión
- **c.** proyección
- **d.** intersección

44. ¿Cuál de las siguientes opciones es un operador unario?
- **a.** intersección
- **b.** unión
- **c.** juntura
- **d.** proyección

45. ¿Cuál de las siguientes opciones es un operador binario?
- **a.** selección
- **b.** actualización
- **c.** diferencia
- **d.** todas las anteriores

46. _____ es un lenguaje declarativo utilizado en las bases de datos relacionales.
- **a.** PDQ
- **b.** SQL
- **c.** LES
- **d.** PBJ

EJERCICIOS

47. Usted tiene las relaciones A, B y C, como se exhibe en la figura 14.16. Muestre la relación resultante si se aplican las siguientes instrucciones SQL.

```
select *
from A
where A2 = 16
```

48. Usted tiene las relaciones A, B y C, como se exhibe en la figura 14.16. Muestre la relación resultante si se aplican las siguientes instrucciones SQL.

```
select A1 A2
from A
where A2 = 16
```

A

A1	A2	A3
1	12	100
2	16	102
3	16	103
4	19	104

B

B1	B2
1	214
2	216
3	284
4	216

C

C1	C2	C3
31	401	1006
32	401	1025
33	405	1065

Figura 14.16 Ejercicios 47-53

49. Usted tiene las relaciones A, B y C, como se exhibe en la figura 14.16. Muestre la relación resultante si se aplican las siguientes instrucciones SQL.

```
select A3
from A
```

50. Usted tiene las relaciones A, B y C, según aparece en la figura 14.16. Muestre la relación resultante si se aplican las siguientes instrucciones SQL.

```
select B1
from B
where B2 = 216
```

51. Usted tiene las relaciones A, B y C, como se exhibe en la figura 14.16. Muestre la relación resultante si se aplican las siguientes instrucciones SQL.

```
update C
set C1 = 37
where C1 = 31
```

52. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo el número de curso y el número de unidades para cada curso.

53. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo el ID de estudiante y el nombre de estudiante.

54. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo el nombre de profesor.

55. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo el nombre de departamento.

56. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene los cursos tomados por el estudiante con el ID 2010.

57. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene los cursos impartidos por el profesor Blake.

58. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo los cursos que se componen de tres unidades.

59. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene sólo el nombre de los estudiantes que toman el curso CIS015.

60. Utilizando el diseño de la figura 14.5, muestre la instrucción SQL que crea una nueva relación que contiene el número de departamento del Departamento de Ciencias de la Computación.

61. Las bases de datos relacionales se basan entre entidades en una organización. Encuentre las entidades en el diseño de la figura 14.5. Por ejemplo, el estudiante y el curso son dos de las entidades de este diseño.

62. Se dice que las relaciones entre entidades son 1:1 (uno a uno, 1:M (uno a muchos) y M:N (muchos a muchos). Utilizando el diseño de la figura 14.5, encuentre el tipo de relación (1:1, 1:M o M:N) entre estudiante y curso.

63. Repita el ejercicio 62 para encontrar el tipo de relación (1:1, 1:M o M:N) entre estudiante y profesor.

64. Repita el ejercicio 62 para encontrar el tipo de relación (1:1, 1:M o M:N) entre departamento y profesor.

65. Repita el ejercicio 62 para encontrar una relación uno a uno entre dos entidades.

66. Diseñe una base de datos relacional para una biblioteca.

67. Diseñe una base de datos relacional para una compañía de bienes raíces.

V

Temas avanzados

Compresión de datos

En días recientes, la tecnología ha cambiado la forma de transmitir y almacenar los datos. Por ejemplo, el cable de fibra óptica nos permite transmitir los datos mucho más rápido y el DVD nos permite almacenar grandes cantidades de datos en un pequeño medio físico. Sin embargo, al igual que en otros aspectos de la vida, el índice de demandas del público siempre se está incrementando. Actualmente queremos descargar más y más información en menos y menos tiempo. También queremos almacenar más y más datos en un espacio pequeño.

La compresión de datos puede reducir la cantidad de datos enviados o almacenados al eliminar parcialmente la redundancia inherente. La redundancia se crea cuando producimos datos. A través de la compresión de datos, hacemos la transmisión y el almacenaje más eficientes, y al mismo tiempo conservamos la integridad de los datos (hasta cierto punto).

La **compresión de datos** significa el envío o almacenamiento de un número pequeño de bits. Aunque muchos métodos se utilizan para este propósito, en general estos métodos pueden dividirse en dos categorías generales: métodos sin pérdida y con pérdida. La figura 15.1 muestra las dos categorías y los métodos comunes utilizados en cada categoría.

Primero analizamos los métodos de compresión sin pérdida más simples y más fáciles de comprender. Después, presentamos los métodos de compresión con pérdida más complicados.

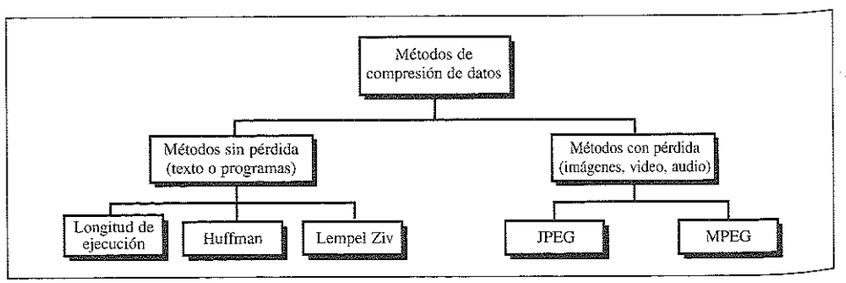


Figura 15.1 Métodos de compresión de datos

15.1 COMPRESIÓN SIN PÉRDIDA

En la **compresión de datos sin pérdida**, se conserva la integridad de los datos. Los datos originales y los datos después de la **compresión y descompresión** son exactamente iguales debido a que, en estos métodos, los algoritmos de compresión y de descompresión son exactamente inversos uno de otro. Ninguna parte de los datos se pierde en el proceso. Los datos redundantes se eliminan en la compresión y se añaden durante la descompresión.

Estos métodos por lo general se utilizan cuando uno no puede permitirse el lujo de perder un solo bit de datos. Por ejemplo, usted no quiere perder datos cuando comprime un archivo de texto o un programa.

En esta sección estudiamos tres métodos de compresión sin pérdida: la codificación de longitud de ejecución, la codificación de Huffman y el algoritmo de Lempel Ziv.

CODIFICACIÓN DE LONGITUD DE EJECUCIÓN

Probablemente el método más simple de compresión es la **codificación de longitud de ejecución**, la cual puede utilizarse para comprimir datos hechos de cualquier combinación de símbolos. No se necesita conocimiento de la frecuencia de ocurrencia de los símbolos (como lo requiere el método siguiente) y puede ser muy eficiente si los datos se representan como ceros y unos.

La idea general que sustenta este método es remplazar ocurrencias de un símbolo repetidas y consecutivas por una ocurrencia del símbolo y el número de ocurrencias. Por ejemplo, AAAAAAAA puede remplazarse por A08. La figura 15.2 presenta un ejemplo de este método simple de compresión. Observe que utilizamos un número fijo de dígitos (dos) para representar el conteo.

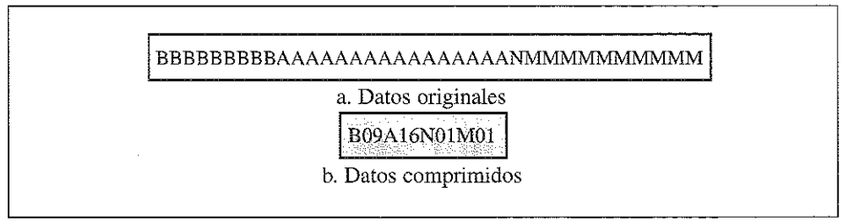


Figura 15.2 Ejemplo de codificación de longitud de ejecución

El método puede ser aún más eficiente si los datos utilizan sólo dos símbolos (por ejemplo, 0 y 1) en su patrón de bits y un símbolo es más frecuente que el otro. Por ejemplo, digamos que usted tiene una imagen representada principalmente por ceros y algunos unos. En este caso puede reducir el número de bits al enviar (o almacenar) el número de ceros que ocurren entre dos unos (figura 15.3).

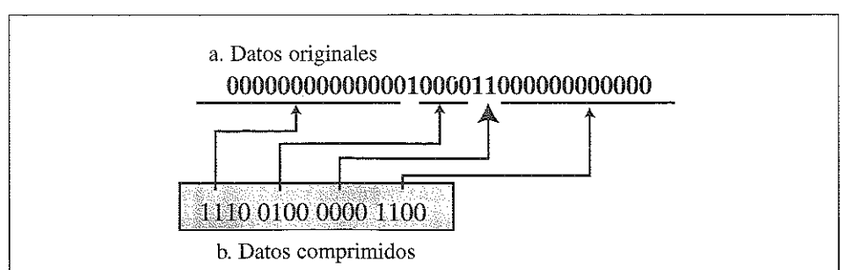


Figura 15.3 Codificación de longitud de ejecución para dos símbolos

Hemos representado los conteos como un número binario de cuatro bits (entero sin signo). En una situación real, usted encontraría un número óptimo de bits para evitar introducir redundancia extra. En la figura 15.3 hay catorce 0 antes del primer 1. Estos catorce 0 están comprimidos en el patrón binario 1110 (14 en binario). El siguiente conjunto de ceros está comprimido en 0100 debido a que hay cuatro 0. Enseguida usted tiene dos 1 en los datos originales, los cuales se representan mediante 0000 en los datos comprimidos. Finalmente, los últimos doce 0 en los datos están comprimidos en 1100.

Observe que, dada una compresión binaria de cuatro bits, si hay más de quince 0, éstos se dividen en dos o más grupos. Por ejemplo, una secuencia de veinticinco 0 se cifra como 1111 1010. Ahora la pregunta es cómo sabe el receptor que éstos son veinticinco 0 y no quince 0, luego un 1 y después diez 0. La respuesta es que si el primer conteo es 1111, el receptor sabe que el siguiente patrón de cuatro bits es una continuación de ceros. Ahora surge otra pregunta: ¿qué pasa si hay exactamente quince 0 entre dos 1? En este caso el patrón es 1111 seguido por 0000.

CODIFICACIÓN DE HUFFMAN

En la **codificación de Huffman**, usted asigna códigos más cortos a símbolos que ocurren con mayor frecuencia y códigos más largos a aquellos que ocurren con menor frecuencia. Por ejemplo, imagine que tiene un archivo de texto que utiliza sólo cinco caracteres (A, B, C, D, E). Elegimos sólo cinco caracteres para hacer el análisis más simple, pero el procedimiento es igualmente válido para un número de caracteres menor o mayor.

Antes de que pueda asignar patrones de bits a cada carácter, usted asigna a cada carácter un peso basado en su frecuencia de uso. En este ejemplo suponga que la frecuencia de los caracteres es la que se presenta en la tabla 15.1. El carácter A ocurre 17 por ciento de las veces, el carácter B ocurre 12 por ciento de las veces y así por el estilo.

Carácter	A	B	C	D	E
Frecuencia	17	12	12	27	32

Tabla 15.1 Frecuencia de caracteres

Una vez que se ha establecido el peso de cada carácter, se construye un árbol con base en aquellos valores. El proceso de construcción de este árbol se muestra en la figura 15.4. Sigue tres pasos básicos:

1. Se pone todo el conjunto de caracteres en una fila. Cada carácter es ahora un **nodo** en el nivel inferior de un árbol.
2. Se encuentran los dos nodos dentro de los pesos más pequeños y se juntan para formar un tercer nodo, el cual dé como resultado un árbol simple de dos niveles. El peso del nuevo nodo son los pesos combinados de los dos nodos originales. Este nodo, un nivel arriba de las hojas, es idóneo para combinarse con otros nodos. Recuerde, la suma de los pesos de los dos nodos elegidos debe ser más pequeña que la combinación de cualesquier otras opciones posibles.
3. Repita el paso 2 hasta que todos los nodos, de cada nivel, se combinen en un solo árbol.

Una vez que el árbol esté completo, utilícelo para asignar códigos a cada carácter. Primero asigne un valor de 1 bit a cada **rama**. Comenzando desde la raíz (nodo superior), asigne un 0 a la rama izquierda y un 1 a la rama derecha, y repita este patrón en cada nodo.

El código de un carácter se encuentra comenzando en la raíz y siguiendo las ramas que conducen a ese carácter. El código mismo es el valor de bits en cada rama del camino tomado en secuencia. La figura 15.5 muestra el árbol final con bits añadidos a cada rama. Observe que movimos los nodos de la hoja para hacer que el árbol pareciera un **árbol binario**.

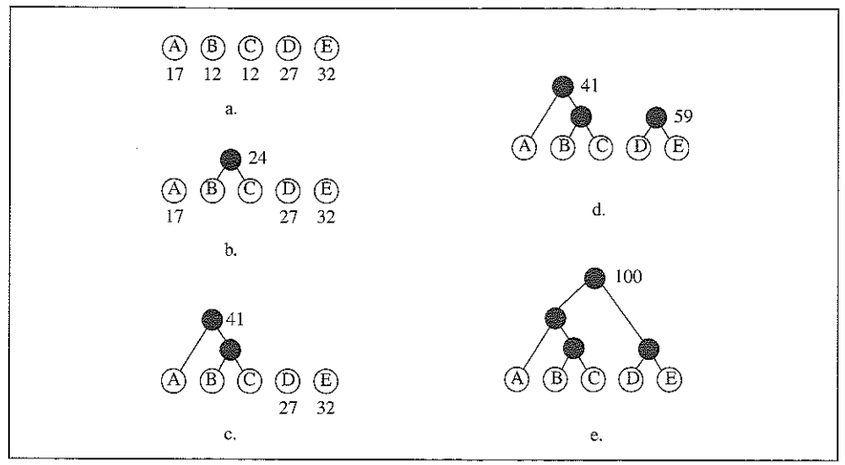


Figura 15.4 Codificación de Huffman

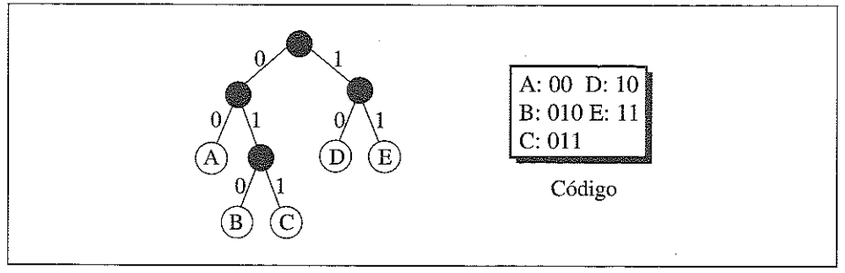


Figura 15.5 Árbol final y código

Observe estos puntos sobre los códigos. Primero, los caracteres con las frecuencias más altas reciben un código más corto (A, D y E) que los caracteres con frecuencias más bajas (B y C). Compare esto con un código que asigna a cada carácter longitudes de bits iguales. Segundo, en este sistema de codificación, ningún código es un prefijo de otro código. Los códigos de dos bits, 00, 10 y 11 no son el prefijo de ninguno de los otros dos códigos (010) y (011). En otras palabras, usted no tiene un código de tres bits que comience con 00, 10 u 11. Esta propiedad hace del código Huffman un código instantáneo. Explicaremos esta propiedad cuando analicemos la codificación y la **decodificación** en la codificación de Huffman.

Codificación

Veamos cómo codificar el texto utilizando el código para nuestros cinco caracteres. La figura 15.6 muestra el texto original y el texto codificado. Vale la pena mencionar dos puntos sobre esta figura. Primero, observe que hay un sentido de compresión incluso en este pequeño código ficticio. Si usted quiere enviar el texto sin utilizar la codificación de Huffman, necesita asignar un código de tres bits a cada carácter. Enviaría 30 bits, mientras que con la codificación de Huffman usted sólo enviaría 22 bits.

Segundo, observe que no hemos utilizado ningún delimitador entre los bits que codifican cada carácter. Escribimos los códigos uno después de otro. La belleza de la codificación Huffman radica en que ningún código es el prefijo de otro código. No hay ambigüedad en la codificación; además, el receptor puede decodificar los datos recibidos sin ambigüedad.

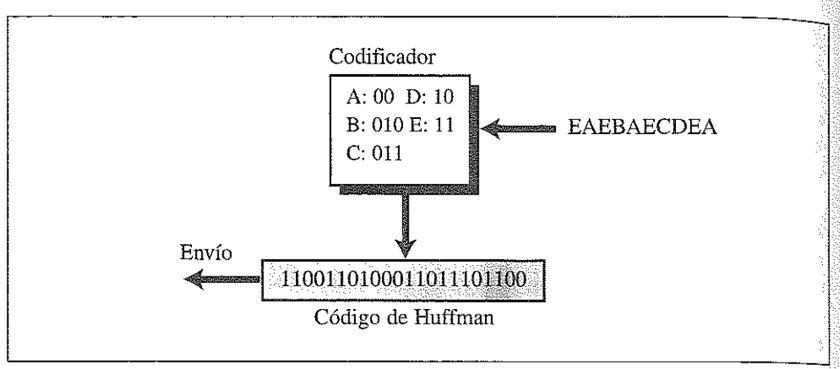


Figura 15.6 Codificación de Huffman

Decodificación

El receptor tiene una tarea muy fácil en la decodificación de los datos que recibe. La figura 15.7 muestra cómo ocurre la decodificación. Cuando el receptor recibe los primeros dos bits no tiene que esperar al siguiente bit para tomar la decisión. Sabe que estos dos se decodifican como E. La razón, como mencionamos antes, es que estos dos bits no son el prefijo de ningún código de tres bits (no hay código de tres bits que inicie con 11). Asimismo, cuando el receptor recibe los siguientes dos bits (00), también sabe que el carácter debe ser A. Los siguientes dos bits se interpretan de la misma manera (11 debe ser E). Sin embargo, cuando recibe los bits 7 y 8, sabe que debe esperar el siguiente bit porque este código (01) no está en la lista de códigos. Después de recibir el siguiente bit (0) interpreta los siguientes tres bits juntos (010) como B. Ésta es la razón por la cual el código de Huffman se llama código instantáneo; el decodificador puede decodificar inequívocamente los bits al instante (con el número mínimo de bits).

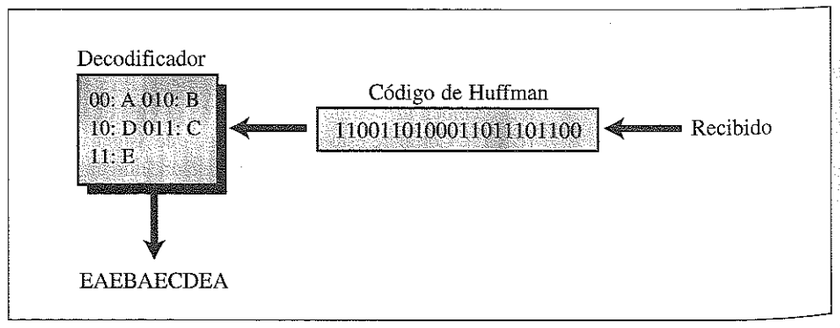


Figura 15.7 Decodificación de Huffman

CODIFICACIÓN DE LEMPEL ZIV

La **codificación de Lempel Ziv (LZ)** es un ejemplo de una categoría de algoritmos llamada **codificación basada en diccionario.** La idea es crear un diccionario (una tabla) de cadenas utilizado durante la sesión de comunicación. Si tanto el emisor como el receptor tienen una copia del diccionario, entonces las cadenas ya encontradas pueden sustituirse por su índice en el diccionario para reducir la cantidad de información transmitida.

Aunque la idea parece simple, varias dificultades afloran en la implementación. Primero, ¿cómo puede crearse un diccionario para cada sesión (no puede ser universal debido a su longitud)? Segundo, ¿cómo puede el receptor adquirir el diccionario hecho por el emisor (si

usted envía el diccionario está enviando información adicional, lo cual va en contra del propósito general de la compresión)?

Un algoritmo práctico que utiliza la idea de la codificación adaptativa basada en diccionario es el algoritmo de Lempel Ziv (LZ). Este algoritmo ha pasado por varias versiones (LZ77, LZ78, etc.). Presentamos la idea básica del algoritmo con un ejemplo pero no ahondamos en los detalles de diferentes versiones e implementaciones. En nuestro ejemplo, suponemos que la cadena siguiente está por enviarse. Hemos elegido esta cadena específica para simplificar el análisis.

BAABABBBAABBBBAA

Utilizando nuestra versión simple del algoritmo LZ, el proceso se divide en dos fases: la compresión de la cadena y la descompresión de la cadena.

Compresión

En esta fase, hay dos eventos concurrentes: la construcción de un diccionario indexado y la compresión de una cadena de símbolos. El algoritmo extrae la **subcadena** más pequeña que no puede encontrarse en el diccionario desde la cadena sin comprimir la restante. Luego almacena una copia de esta subcadena en el diccionario (como una nueva entrada) y asigna un valor de índice. La compresión ocurre cuando la subcadena, con excepción del último carácter, se remplaza con el índice encontrado en el diccionario. El proceso luego inserta el índice y el nuevo carácter de la subcadena en la cadena comprimida. Por ejemplo, si la subcadena es ABBB, usted busca ABB en el diccionario y encuentra que el índice para ABB es 4, por consiguiente la subcadena comprimida es 4B. La figura 15.8 muestra el proceso para nuestra cadena de ejemplo.

Sigamos unos cuantos pasos en esta figura:

PASO 1 El proceso extrae la subcadena más pequeña de la cadena original que no está en el diccionario. Debido a que el diccionario está vacío, el carácter más pequeño es un carácter (el primer carácter, B). El proceso almacena una copia de éste como la primera entrada en el diccionario. Su índice es 1. Ninguna parte de esta subcadena puede ser remplazada con un índice del diccionario original (es sólo un carácter). El proceso inserta B en la cadena comprimida. Hasta ahora, la cadena comprimida tiene sólo un carácter: B. La cadena restante sin comprimir es la cadena original sin el primer carácter.

PASO 2 El proceso extrae de la cadena restante la siguiente subcadena más pequeña que no está en el diccionario. Esta subcadena es el carácter A, el cual no está en el diccionario. El proceso almacena una copia de ésta como la segunda entrada en el diccionario. Ninguna parte de esta subcadena puede remplazarse con un índice desde el diccionario (sólo es un carácter). El proceso inserta A en la cadena comprimida. Hasta ahora, la cadena comprimida tiene dos caracteres: B y A (hemos colocado comas entre las subcadenas de la cadena comprimida para mostrar la separación).

PASO 3 El proceso extrae de la cadena restante la siguiente subcadena más pequeña que no está en el diccionario. Esta situación difiere de los dos pasos previos. El siguiente carácter (A) está en el diccionario, así que el proceso extrae dos caracteres (AB), los cuales no están en el diccionario. El proceso almacena una copia de AB como la tercera entrada en el diccionario. El proceso ahora encuentra el índice de una entrada en el diccionario que es la subcadena sin el último carácter (AB sin el último carácter es A). El índice para A es 2, así que el proceso remplaza A con 2 e inserta 2B en la cadena comprimida.

PASO 4 Enseguida el proceso extrae la subcadena ABB (debido a que A y AB ya están en el diccionario). Una copia de ABB se almacena en el diccionario con un índice de 4. El proceso encuentra el índice de la subcadena sin el último carácter (AB), el cual es 3. La combinación 3B se inserta en la cadena comprimida. Tal vez haya notado que en los tres pasos anteriores, en realidad no logramos ninguna compresión debido a que hemos remplazado un

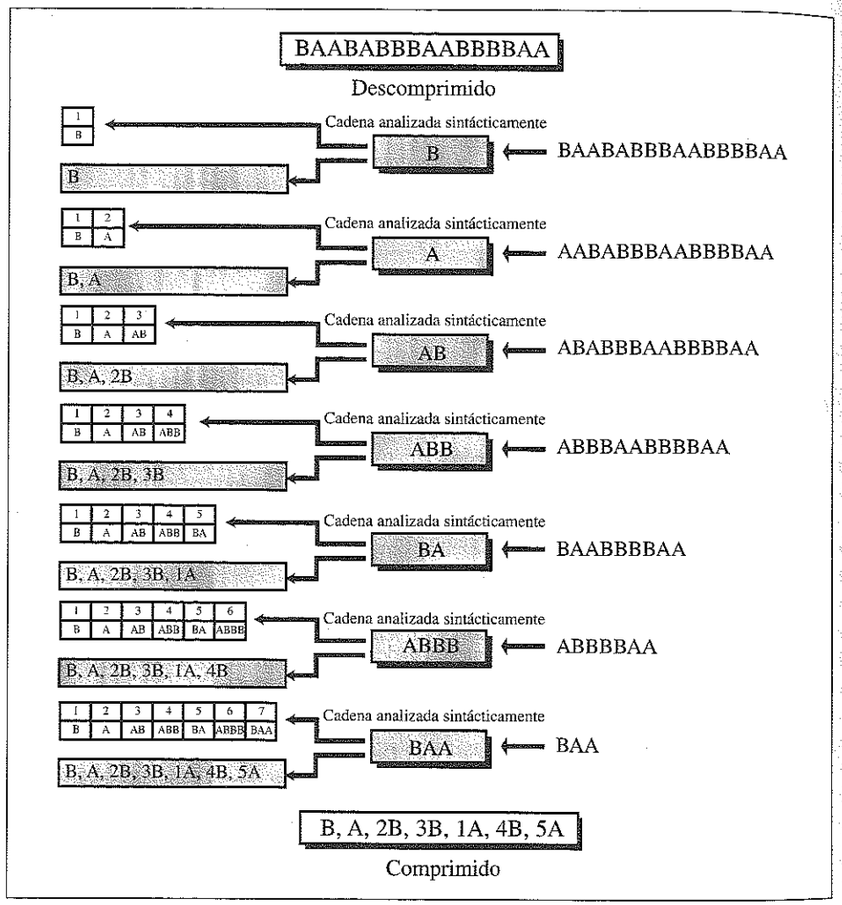


Figura 15.8 Ejemplo de decodificación Lempel Ziv

carácter por uno (A por A en el primer paso y B por B en el segundo paso) y dos caracteres por dos (AB por 2B en el tercer paso). Pero en este paso en realidad hemos reducido el número de caracteres (ABB se convierte en 3B). Si la cadena original tiene muchas repeticiones (lo cual es cierto en la mayoría de los casos), podemos reducir en gran medida el número de caracteres.

Los pasos restantes son similares a uno de los cuatro pasos precedentes, así que dejamos al lector seguir adelante. Observe que el diccionario sólo se utilizó por el emisor para encontrar los índices. Éste no se envía al receptor y el receptor debe crear el diccionario para sí mismo como veremos en la siguiente sección.

Descompresión

La descompresión es el proceso inverso de la compresión. Este proceso extrae las subcadenas de la cadena comprimida y trata de remplazar los índices con la entrada correspondiente en el diccionario, el cual está vacío al principio y se construye gradualmente. Toda la idea es que cuando se recibe un índice, ya hay una entrada en el diccionario que corresponde a ese índice. La figura 15.9 muestra el proceso de descompresión.

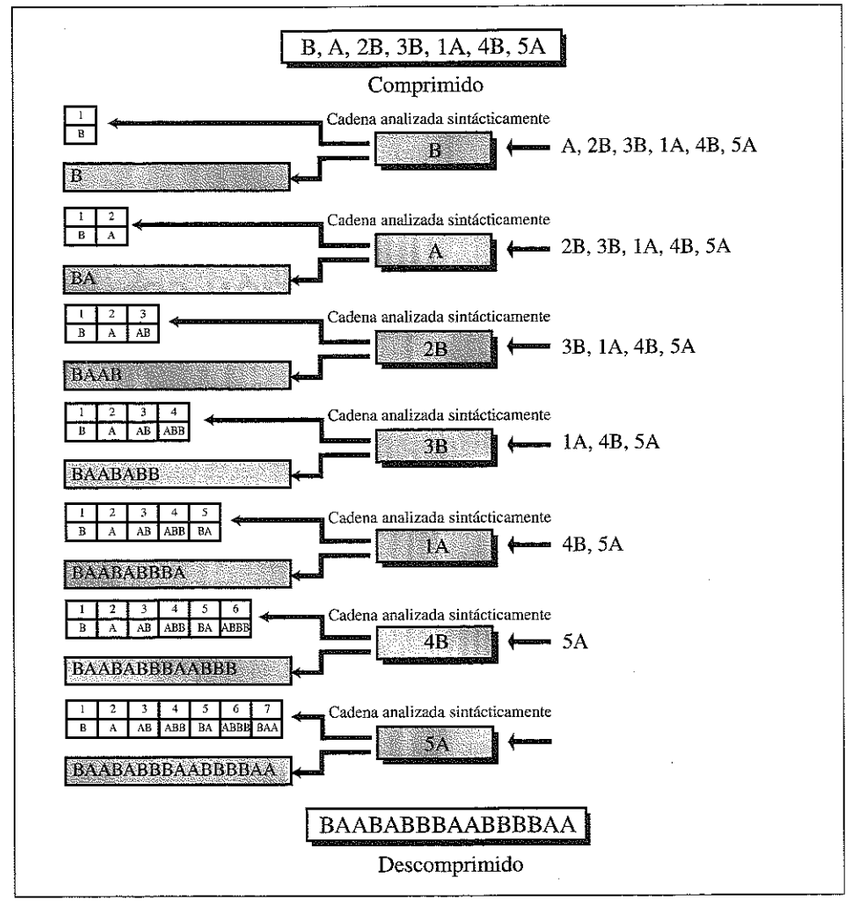


Figura 15.9 Ejemplo de decodificación Lempel Ziv

Veamos unos cuantos pasos en la figura.

PASO 1 La primera subcadena de la cadena comprimida se examina. Es B sin un índice. Debido a que la subcadena no está en el diccionario, ésta se añade al diccionario. La subcadena (B) se inserta en la cadena descomprimida.

PASO 2 La segunda subcadena (A) se examina; la situación es similar al paso 1. Ahora la cadena descomprimida tiene dos caracteres (BA) y el diccionario tiene dos entradas.

PASO 3 La tercera subcadena (2B) se examina. El proceso realiza una búsqueda en el diccionario y remplaza el índice 2 con la subcadena A. La nueva subcadena (AB) se añade a la cadena descomprimida y AB se añade al diccionario.

PASO 4 La cuarta subcadena (3B) se examina. El proceso busca en el diccionario y remplaza el índice 3 con la subcadena AB. La subcadena ABB ahora se añade a la cadena descomprimida y ABB se añade al diccionario.

Dejamos la exploración de los últimos tres pasos como un ejercicio. Como habrá notado, utilizamos un número como 1 o 2 para el índice. En realidad el índice es un patrón binario (posiblemente variable en longitud) para una mayor eficiencia. También advierta que la codificación LZ deja el último carácter sin comprimir (lo cual significa menor eficiencia). Una versión de la codificación LZ, llamada codificación de Lempel Ziv Welch (LZW) comprime incluso este solo carácter. No obstante, dejamos el análisis de este algoritmo a libros de texto más especializados.

15.2 MÉTODOS DE COMPRESIÓN CON PÉRDIDA

La pérdida de información no es aceptable en un archivo de texto o archivo de programa. Sin embargo es aceptable en una imagen o video. La razón es que nuestros ojos y oídos no pueden distinguir cambios sutiles. Para estos casos, usted puede usar un método de **compresión de datos con pérdida.** Estos métodos son más baratos y requieren menos tiempo y espacio cuando se trata de enviar millones de bits por segundo para imágenes y video.

Varios métodos se han desarrollado utilizando técnicas de compresión con pérdida. El **grupo unido de expertos en fotografía** (**JPEG:** *joint photographic experts group*) se usa para comprimir imágenes y gráficas. El **grupo de expertos en imágenes en movimiento** (**MPEG:** *motion picture experts group*) se utiliza para comprimir video.

COMPRESIÓN DE IMÁGENES: JPEG

Como se vio en el capítulo 2, una imagen puede representarse mediante un arreglo bidimensional (tabla) de elementos de imagen (pixeles); por ejemplo, 640 × 480 = 307 200 pixeles. Si la imagen es de escala de grises, cada pixel puede representarse por un entero de ocho bits (256 niveles). Si la imagen es a color, cada pixel puede representarse por 24 bits (3 × 8 bits), con cada ocho bits representando uno de los colores en el sistema de colores RBG (o YIQ). Para simplificar el análisis, nos concentramos en una imagen de escala de grises con 640 × 480 pixeles.

Usted puede ver por qué necesita compresión. Una imagen de escala de grises de 307 200 pixeles se representa mediante 2 457 600 bits y una imagen a color se representa por medio de 7 372 800 bits. En JPEG, una imagen de escala de grises se divide en bloques de 8 × 8 pixeles (figura 15.10).

El propósito de dividir la imagen en bloques es disminuir el número de cálculos, debido a que, como verá en breve, el número de operaciones matemáticas para cada imagen es el

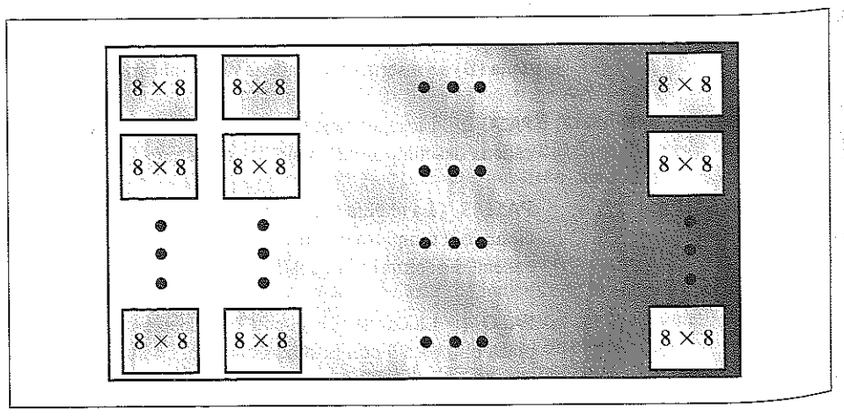


Figura 15.10 Ejemplo de JPEG en escala de grises, 640 × 480 pixeles

cuadrado del número de unidades. Es decir, para toda la imagen, se necesitan 307 200^2 operaciones (94 371 840 000 operaciones). Si se utiliza JPEG, se necesitan 64^2 operaciones para cada bloque, lo cual da un total de $64^2 \times 80 \times 60$ o 19 660 800 operaciones. Esto disminuye 4 800 veces el número de operaciones.

La idea de JPEG es cambiar la imagen en una serie de números lineales (vector) que revela las redundancias. Las redundancias (falta de cambios) pueden entonces eliminarse utilizando uno de los métodos de compresión sin pérdida que se estudiaron previamente. Una versión simplificada del proceso aparece en la figura 15.11.

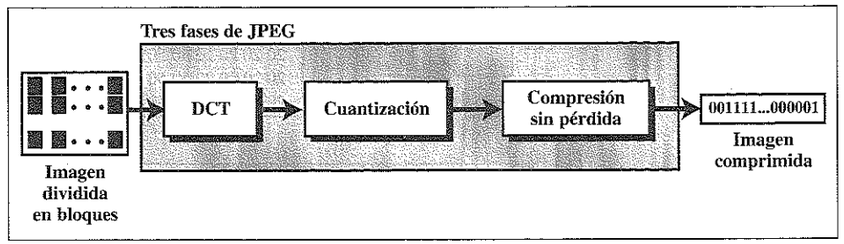


Figura 15.11 Proceso JPEG

Transformación discreta coseno (DCT)

En este paso, cada bloque de 64 pixeles atraviesa por una transformación llamada la **transformación discreta coseno** (**DCT:** *discrete cosine transform*). La transformación cambia los 64 valores de manera que se mantienen las relaciones relativas entre pixeles, pero las redundancias se revelan. La fórmula se da en el apéndice F. $P(x, y)$ define un valor en el bloque; $T(x, y)$ define el valor en el bloque transformado. El apéndice F muestra la fórmula matemática para una transformación.

Para comprender la naturaleza de esta transformación, permítanos mostrar el resultado de las transformaciones para tres casos.

Caso 1 En este caso, usted tiene un bloque de escala de grises uniforme y el valor de cada pixel es 20. Cuando se hacen las transformaciones se obtiene un valor diferente de cero para el primer elemento (esquina superior izquierda). El resto de los pixeles tiene un valor de cero debido a que, de acuerdo con la fórmula, el valor de $T(0, 0)$ es el promedio de los otros valores. Esto se conoce como el **valor DC** (tomado de las siglas en inglés de corriente directa, de la ingeniería eléctrica). El resto de los valores, llamados **valores AC**, en $T(x, y)$ representan los cambios en los valores de pixel. Pero como éstos no tienen cambios, el resto de los valores son ceros (figura 15.12).

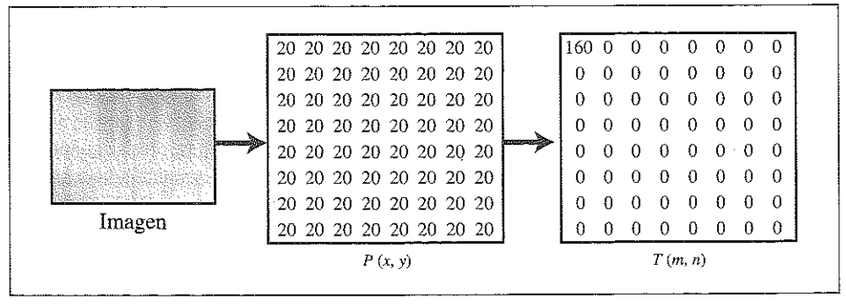


Figura 15.12 Caso 1: escala de grises uniforme

Caso 2 En el segundo caso, usted tiene un bloque con dos secciones diferentes de escala de grises uniforme. Hay un cambio brusco en los valores de los pixeles (de 20 a 50). Cuando se hacen las transformaciones se obtiene un valor DC así como valores AC diferentes de cero. No obstante, sólo hay unos cuantos valores diferentes de cero agrupados alrededor del valor DC. La mayoría de los valores son cero (figura 15.13).

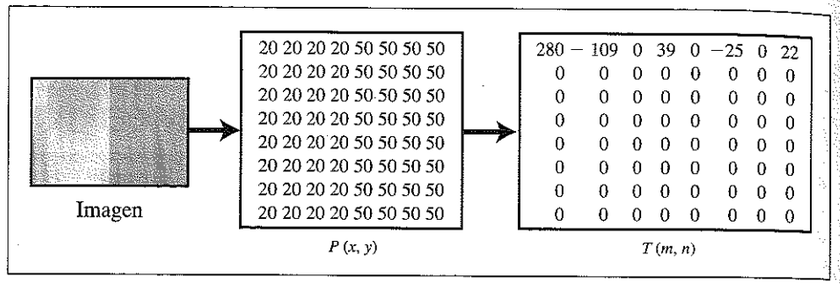


Figura 15.13 Caso 2: dos secciones

Caso 3 En el tercer caso, se tiene un bloque que cambia gradualmente. Esto significa que no hay un cambio brusco entre los valores de los pixeles vecinos. Cuando se hacen las transformaciones se obtiene un valor DC, también con muchos valores AC diferentes de cero (figura 15.14).

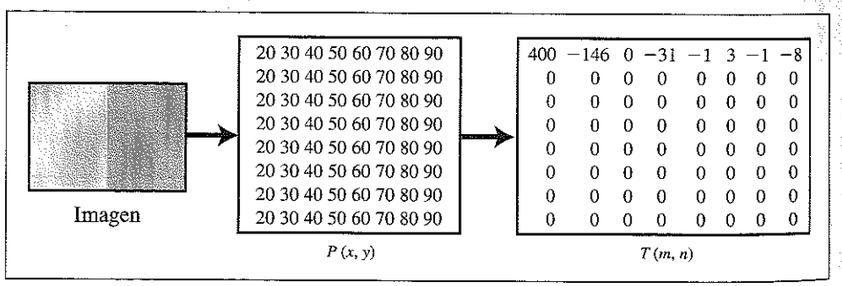


Figura 15.14 Caso 3: escala de grises gradiente

A partir de las figuras 15.12, 15.13 y 15.14, podemos establecer lo siguiente:

- La transformación crea la tabla *T* a partir de la tabla *P*.
- El valor DC da un valor promedio de los pixeles.
- Los valores AC dan los cambios.
- La falta de cambios en los pixeles vecinos crea ceros.

Antes de que cerremos el análisis de la DCT, observe que la transformación DCT es reversible. El apéndice F también muestra la fórmula matemática para una transformación inversa.

Cuantización

Después de que se crea la tabla *T*, los valores se cuantizan para reducir el número de bits necesarios para la codificación. Anteriormente a la cuantización, dejábamos la fracción de cada valor y manteníamos la parte entera. En la actualidad dividimos el número entre una constante y luego dejamos la fracción. Esto reduce el número requerido de bits todavía más. En la mayoría de las implementaciones, una tabla de cuantización (8 por 8) define cómo cuantizar cada valor. El divisor depende de la posición del valor en la tabla *T*. Esto se hace para optimizar el número de bits y el número de ceros para cada aplicación en particular.

Observe que la única fase en el proceso que no es reversible es la fase de cuantización. Usted pierde un poco de información aquí que no es recuperable. La verdad es que la única razón por la cual JPEG se llama compresión con pérdida es por la fase de cuantización.

Después de la cuantización, se leen los valores de la tabla y los ceros redundantes se eliminan. Sin embargo, para agrupar los ceros juntos, el proceso lee la tabla en forma diagonal a manera de zigzag en vez de fila por fila o columna por columna. La razón es que si la imagen no tiene cambios finos, la esquina inferior derecha de la tabla *T* son todos ceros. La figura 15.15 muestra el proceso. JPEG por lo general utiliza codificación de longitud de ejecución en la fase de compresión para comprimir el patrón de bits que resulta de la linealización de zigzag.

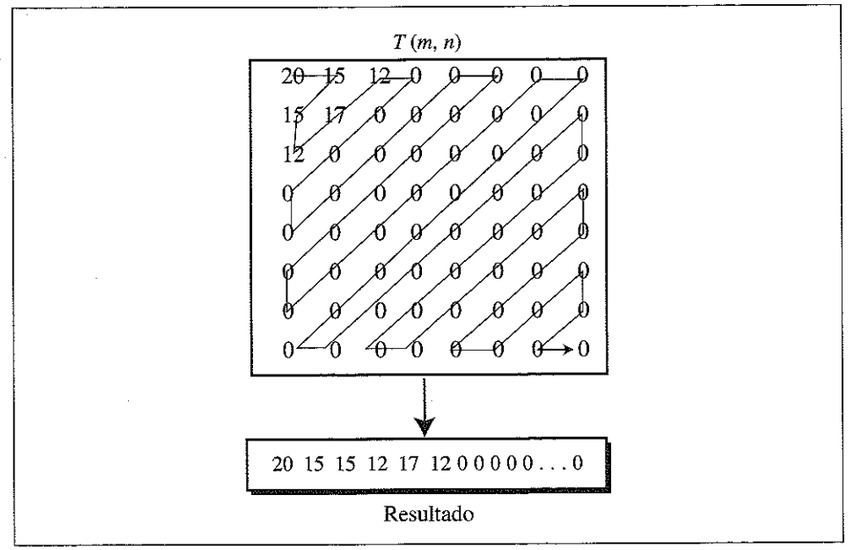


Figura 15.15 Lectura de la tabla

COMPRESIÓN DE VIDEO: MPEG

El método del grupo de expertos de imágenes en movimiento (MPEG) se utiliza para comprimir video. En un principio, una imagen en movimiento es un flujo rápido de una serie de cuadros (*frames*), donde cada cuadro es una imagen. En otras palabras, un cuadro es una combinación espacial de pixeles y un video es una combinación temporal de cuadros que se envían uno después de otro. La compresión de video, entonces, significa comprimir espacialmente cada cuadro y comprimir temporalmente una serie de cuadros.

Compresión espacial La **compresión espacial** de cada cuadro se realiza con JPEG (o una modificación de éste). Cada cuadro es una imagen que puede comprimirse de manera independiente.

Compresión temporal En la **compresión temporal,** los cuadros redundantes se eliminan. Cuando vemos televisión, recibimos 30 cuadros por segundo. Sin embargo, la mayoría de los cuadros consecutivos son casi iguales. Por ejemplo, cuando alguien está hablando, la mayor parte del cuadro es la misma que la del cuadro anterior, con excepción del segmento del cuadro alrededor de los labios, el cual cambia de un cuadro a otro.

Un cálculo aproximado apunta a la necesidad de compresión temporal para video. Una compresión JPEG 20:1 de un cuadro envía 368 640 bits por cuadro; a 30 cuadros por segundo, esto es 11 059 200 bits por segundo. Necesitamos reducir este número.

Para comprimir los datos temporalmente, el método MPEG primero divide los cuadros en tres categorías: cuadros I, cuadros P y cuadros B.

- **Cuadros I.** Un **cuadro intracodificado** (**cuadro I**) es un cuadro independiente que no se relaciona con ningún otro cuadro (ni con el cuadro enviado antes ni con el cuadro enviado después). Éste se presenta a intervalos regulares (por ejemplo, cada noveno cuadro es un cuadro I). Un cuadro I debe aparecer periódicamente debido a algún cambio repentino en el cuadro que los cuadros anterior y posterior no pueden mostrar. Además, cuando se transmite un video, un espectador o espectadora puede ajustar su receptor en cualquier momento. Si sólo hay un cuadro I al principio de la transmisión, el espectador que hace el ajuste más tarde no recibirá una imagen completa. Los cuadros I son independientes de otros cuadros y no pueden construirse a partir de otros cuadros.
- **Cuadros P.** Un **cuadro predecible** (**cuadro P**) se relaciona con el cuadro I o con el cuadro P precedente. En otras palabras, cada cuadro P contiene sólo los cambios respecto del cuadro precedente. Los cambios, sin embargo, no cubren un segmento grande. Por ejemplo para un objeto en movimiento rápido, los nuevos cambios pueden no registrarse en un cuadro P. Los cuadros P pueden construirse sólo a partir de los cuadros I o P previos. Los cuadros P transportan mucha menos información que otros tipos de cuadros y transportan incluso menos bits después de la compresión.
- **Cuadros B.** Un **cuadro bidireccional** (**cuadro B**) le conciernen los cuadros I o P anterior o posterior. En otras palabras cada cuadro B se relaciona con el pasado y con el futuro. Observe que un cuadro B nunca se relaciona con otro cuadro B.

La figura 15.16 muestra una secuencia de cuadros de muestra.

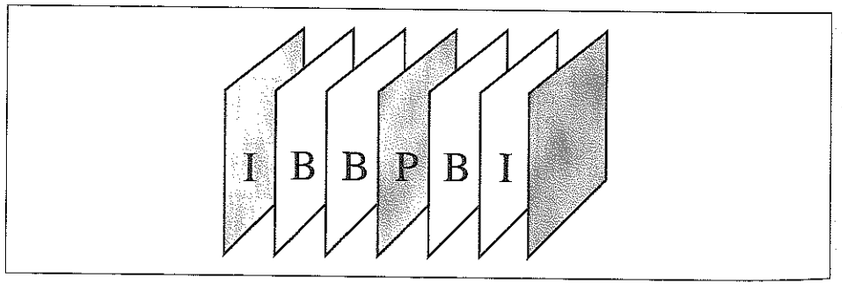


Figura 15.16 Cuadros MPEG

La figura 15.17 exhibe cómo los cuadros I, P y B se construyen a partir de una serie de siete cuadros.

MPEG ha pasado por muchas versiones. MPEG 1 se diseñó para un CD-ROM con un índice de datos de 1.5 Mbps. MPEG 2 se diseñó para DVD de alta calidad con un índice de datos de 3-6 Mbps. MPEG 3 (o MP3) es un estándar para compresión de audio.

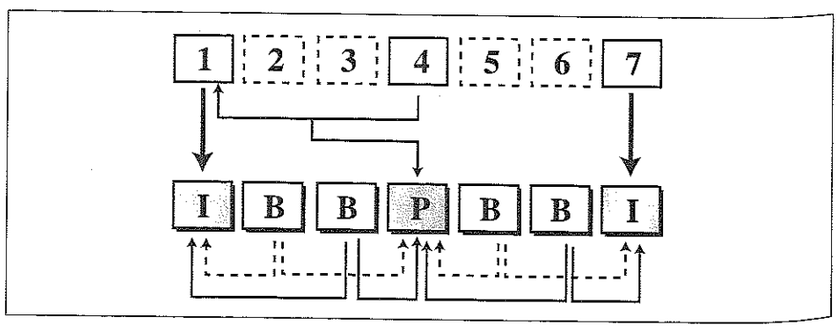


Figura 15.17 Construcción de cuadros MPEG

15.3 TÉRMINOS CLAVE

árbol binario
codificado basado en diccionario
codificación de Huffman
codificación de longitud de ejecución
codificado Lempel Ziv (LZ)
compresión
compresión de datos
compresión de datos con pérdida
compresión de datos sin pérdida
compresión espacial
compresión temporal
cuadro bidireccional (cuadro B)
cuadro intracodificado (cuadro I)
cuadro predecible (cuadro P)
cuantización
decodificado
descompresión
grupo de expertos de imágenes en movimiento (MPEG)
grupo unido de expertos en fotografía (JPEG)
nodo
rama
subcadena
transformación discreta coseno (DCT)
valor AC
valor DC

15.4 RESUMEN

- Los métodos de compresión de datos son ya sea sin pérdida (toda la información es recuperable) o con pérdida (alguna información se pierde).
- La compresión de datos toma el mensaje original y reduce el número de bits que se van a transmitir.
- En los métodos de compresión sin pérdida, los datos recibidos son la réplica exacta de los datos enviados.
- Los tres métodos de compresión sin pérdida son la codificación de longitud de ejecución, la codificación de Huffman y la codificación de Lempel Ziv (LZ).
- En la codificación de longitud de ejecución, las ocurrencias repetidas de un símbolo se remplazan por un símbolo y el número de ocurrencias del símbolo.
- En la codificación de Huffman, la longitud del código es una función de la frecuencia de símbolos; los símbolos más frecuentes tienen códigos más cortos que símbolos menos frecuentes.
- En la codificación LZ, las cadenas o palabras repetidas se almacenan en variables. Un índice a la variable remplaza la cadena o palabra.
- La codificación LZ requiere un diccionario y un algoritmo tanto en el emisor como en el receptor.
- En los métodos de compresión con pérdida, los datos recibidos no necesitan ser una réplica exacta de los datos enviados.
- El grupo unido de expertos en fotografía (JPEG) es un método para comprimir imágenes y gráficos.
- El proceso JPEG involucra la división en bloques, la transformación discreta de coseno, la cuantización y la compresión sin pérdida.
- El grupo de expertos en imágenes en movimiento (MPEG) es un método para comprimir video.
- MPEG involucra tanto la compresión espacial como la compresión temporal. La primera es similar a JPEG y la última elimina cuadros redundantes.

15.5 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuáles son las dos categorías de los métodos de compresión de datos?
2. ¿Cuál es la diferencia entre la compresión sin pérdida y la compresión con pérdida?
3. ¿Cuál es la codificación de longitud de ejecución?
4. ¿Cómo reduce la codificación de Lempel Ziv la cantidad de bits transmitidos?
5. ¿Qué es la codificación de Huffman?
6. ¿Cuál es el rol del diccionario en la codificación LZ?
7. ¿Cuál es la ventaja de la codificación LZ sobre la codificación de Huffman?
8. ¿Cuáles son dos métodos de compresión con pérdida?
9. ¿Cuándo utilizaría usted JPEG? ¿Cuándo utilizaría MPEG?
10. ¿Cómo se relaciona MPEG con JPEG?

11. En JPEG, ¿cuál es la función de dividir en bloques la imagen?

12. ¿Por qué la DCT se necesita en JPEG?

13. ¿Cómo contribuye la cuantización a la compresión?

14. ¿Qué es un cuadro en la compresión MPEG?

15. ¿Qué es la compresión espacial comparada con la compresión temporal?

16. Comente los tres tipos de cuadros utilizados en MPEG.

PREGUNTAS DE OPCIÓN MÚLTIPLE

17. Los datos se comprimen utilizando un diccionario con índices a las cadenas. Éste es la _____.
    a. codificación diferencial
    b. codificación de Lempel Ziv
    c. codificación Morse
    d. codificación con pérdida

18. Una cadena de cien ceros se remplaza con dos marcadores, un 0 y el número 100. Ésta es la _____.
    a. codificación de longitud de ejecución
    b. codificación Morse
    c. codificación diferencial
    d. codificación de Lempel Ziv

19. Un ejemplo de la compresión con pérdida es _____.
    a. la codificación diferencial
    b. la codificación de Lempel Ziv
    c. la codificación de longitud de ejecución
    d. JPEG

20. En un método de compresión de datos _____, los datos recibidos son una copia exacta del mensaje original.
    a. sin pérdida
    b. con pérdida
    c. de menor pérdida
    d. brillantes

21. En un método de compresión de datos _____, los datos recibidos no necesitan ser una copia exacta del mensaje original.
    a. sin pérdida
    b. con pérdida
    c. de menor pérdida
    d. brillantes

22. La codificación de _____ es un método de compresión de datos sin pérdida.
    a. Huffman
    b. longitud de ejecución
    c. LZ
    d. todos los anteriores

23. En la codificación de _____, los caracteres que ocurren con mayor frecuencia tienen códigos más cortos que los caracteres que ocurren con menos frecuencia.
    a. Huffman
    b. longitud de ejecución
    c. LZ
    d. todos los anteriores

24. En la codificación de _____, PPPPPPPPPPPPPP puede remplazarse por P15.
    a. Huffman
    b. longitud de ejecución
    c. LZ
    d. todos los anteriores

25. En la codificación de _____, una cadena se remplaza por un apuntador a la cadena almacenada.
    a. Huffman
    b. longitud de ejecución
    c. LZ
    d. todos los anteriores

26. La codificación LZ requiere _____.
    a. un diccionario
    b. un buffer
    c. un algoritmo
    d. todos los anteriores

27. La codificación JPEG involucra _____, un proceso que revela las redundancias en un bloque.
    a. división en bloques
    b. la DCT
    c. cuantización
    d. vectorización

28. En la codificación JPEG el proceso de _____ divide la imagen original en bloques más pequeños y asigna un valor a cada pixel en un bloque.
    a. división en bloques
    b. la DCT
    c. cuantización
    d. vectorización

29. El último paso en JPEG, la _____, elimina las redundancias.
    a. división en bloques
    b. cuantización
    c. compresión
    d. vectorización

30. _____ es un método de compresión con pérdida para imágenes y gráficos; mientras que _____ es un método de compresión con pérdida para video.
    a. DCT; MPEG
    b. MPEG; JPEG
    c. JPEG; MPEG
    d. JPEG; DCT

EJERCICIOS

31. Codifique el siguiente patrón de bits utilizando la codificación de longitud de ejecución con códigos de 5 bits:
    dieciocho ceros, 11, cincuenta y seis ceros, 1, quince ceros, 11

32. Codifique el siguiente patrón de bits utilizando codificación de longitud de ejecución con códigos de 5 bits:
    1, ocho ceros, 1, cuarenta y cinco ceros, 11

33. Codifique los caracteres siguientes usando codificación de Huffman con las frecuencias dadas:
    A(12), B(8), C(9), D(20), E(31), F(14), G(8)

34. Codifique los caracteres siguientes utilizando codificación de Huffman. Cada caracter tiene la misma frecuencia (1):
    A, B, C, D, E, F, G, H, I, J

35. ¿Puede ser la siguiente, codificación de Huffman? Explique.
    A: 0 B: 10 C: 11

36. ¿Puede ser el siguiente un código de Huffman? Explique.
    A: 0 B: 1 C: 00 D: 01 E: 10 F: 11

37. Codifique el mensaje BAABBBBAACAA utilizando el código Huffman siguiente:
    A: 0 B: 10 C: 11

38. Decodifique el mensaje 0101000011110 utilizando el código Huffman siguiente:
    A: 0 B: 10 C:11

39. Utilice una transformación para transformar una tabla de cuatro por cuatro. Las reglas son las siguientes:

$$T(0,0) = (1/16)\,[\,P(0,0) + P(0,1) + P(0,2) + \ldots\,]$$

$$T(0,1) = (1/16)\,[\,0.95\,P(0,0) + 0.9\,P(0,1) + 0.85\,P(0,2) + \ldots\,]$$

$$T(0,2) = (1/16)\,[\,0.90\,P(0,0) + 0.85\,P(0,1) + 0.80\,P(0,2) + \ldots\,]$$

    Compare y contraste este método con la DCT. ¿La DCT en realidad es un cálculo promedio con peso como la transformación precedente? De ser así, ¿cuál es el peso?

16 Seguridad

En la actualidad, la seguridad juega un papel muy importante en las ciencias de la computación. Con el crecimiento de Internet, más y más datos se están intercambiando y esos datos necesitan asegurarse. Por ejemplo, cuando usted realiza una compra en Internet, espera que la información que envía al vendedor se mantenga en secreto y que sólo la utilice el vendedor. Además, cuando recibe un mensaje, en ocasiones necesita autenticar al emisor. En este capítulo tratamos el tema de la **seguridad**. El tema es tan vasto que libros enteros se han dedicado a él. Los conceptos e ideas presentados aquí son motivadores para un estudio posterior.

Podemos decir que hay cuatro aspectos de la seguridad: privacidad (confidencialidad), autenticación de mensajes, integridad de los mensajes y no rechazo (figura 16.1).

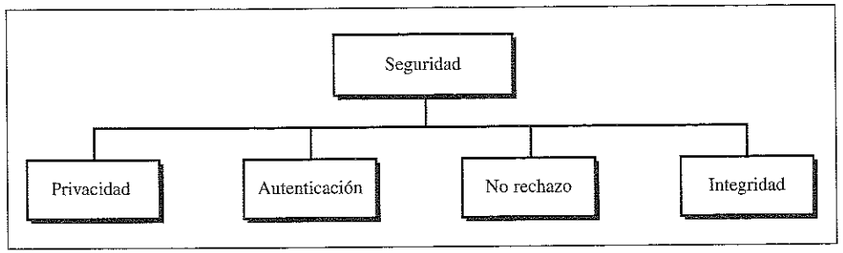


Figura 16.1 Aspectos de la seguridad

PRIVACIDAD

En una comunicación segura, el emisor y el receptor esperan privacidad, o confidencialidad. En otras palabras, sólo el emisor y el receptor del mensaje son capaces de comprender el contenido del mensaje.

AUTENTICACIÓN

La confidencialidad del mensaje no es suficiente en una comunicación segura; la **autenticación** también es necesaria. El receptor necesita estar seguro de la identidad del emisor.

INTEGRIDAD

La confidencialidad y la autenticación son sólo dos elementos de una comunicación segura. La integridad del mensaje también necesita conservarse. Ni el emisor ni el receptor están felices si el contenido del mensaje se modifica durante la transmisión. Por ejemplo, en una transacción bancaria, ni el cliente ni el banco están satisfechos si la transferencia de $1 000 del cliente cambia a $10 000 durante la transmisión. El cambio podría ocurrir ya sea de manera maliciosa, podría ser provocado por un intruso que se beneficia del cambio u ocurrir accidentalmente como resultado de un mal funcionamiento de hardware o software.

NO RECHAZO

Aun cuando no es muy obvio, uno de los elementos de una comunicación segura es el **no rechazo**, es decir, la prevención del rechazo (negación) del emisor. En otras palabras, un sistema seguro necesita probar que el emisor en realidad envió el mensaje. Por ejemplo, cuando un cliente envía un mensaje para transferir dinero de una cuenta a otra, el banco debe haber probado que el cliente en realidad solicitó esta transacción.

16.1 PRIVACIDAD

La privacidad requiere que el mensaje se cifre de alguna manera en el sitio del emisor y se descifre en el sitio del receptor de modo que un intruso potencial (una persona que escuche conversaciones ajenas) no pueda comprender su contenido.

CIFRADO/ DESCIFRADO

En la actualidad, la privacidad puede lograrse usando métodos de cifrado/descifrado. Los datos se cifran en el emisor y se descifran en el receptor. Dos categorías de métodos de cifrado/descifrado en uso hoy en día son la llave secreta y la llave pública.

Privacidad con cifrado de llave secreta

La manera más simple de cifrar los datos es utilizar una **llave secreta**. El emisor utiliza esta clave y un algoritmo de **cifrado** para cifrar los datos; el receptor utiliza la misma llave y el algoritmo de **descifrado** correspondiente para descifrar los datos (figura 16.2).

Los datos, cuando no se cifran, se llaman **texto plano**; una vez cifrados se llaman **texto cifrado**. Observe que tanto el usuario A como el usuario B utilizan la llave secreta, la cual es exactamente la misma. Sin embargo, los algoritmos de cifrado y descifrado son inversos

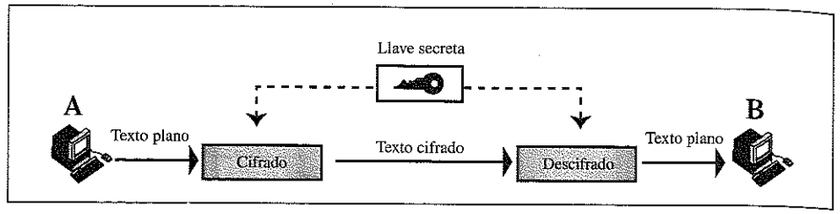


Figura 16.2 Cifrado de llave secreta

entre sí en el sentido de que, por ejemplo, si el algoritmo de cifrado añade algo a los datos, el algoritmo de descifrado sustrae lo mismo de los datos.

En el cifrado de llave secreta, la misma llave se utiliza en el cifrado y en el descifrado. Sin embargo, los algoritmos de cifrado y descifrado son inversos entre sí.

Observe que frecuentemente se hace referencia a los algoritmos de cifrado de llave secreta como algoritmos de cifrado simétrico debido a que la misma llave secreta puede utilizarse en la comunicación bidireccional.

Estándar de cifrado de datos (DES) El cifrado de llave secreta se ha utilizado por más de dos milenios. Al principio, los algoritmos eran muy simples y las llaves eran muy fáciles de adivinar. Hoy en día, usamos algoritmos muy sofisticados; el más común es el llamado **estándar de cifrado de datos (DES:** *data encryption standard*).

DES cifra y descifra en el nivel de bits. Los datos primero se transforman en una cadena de bits. Se dividen en segmentos de 64 bits (ceros adicionales se añaden a la última sección si ésta no es de 64 bits). Cada sección se cifra luego utilizando una llave de 56 bits (en realidad, la llave mide 64 bits, pero 8 bits son para el control de errores). La figura 16.3 muestra la disposición general de este método.

La idea es revolver los datos y la llave de tal manera que cada bit de texto cifrado dependa de cada bit de texto plano y de la llave. Esto vuelve muy difícil para un usuario adivinar los bits de texto plano a partir de los bits de texto cifrado.

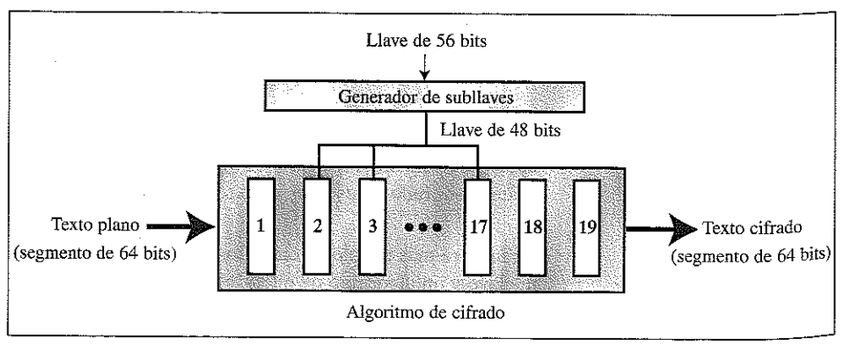


Figura 16.3 DES

La etapas 1, 18 y 19 del algoritmo son sólo operaciones de **permutación** (que no usan la llave). Las etapas 2 a 17 son etapas idénticas. Los 32 bits de la derecha de una etapa se convierten en los 32 bits de la izquierda de la etapa siguiente. Los 32 bits de la izquierda de una etapa se revuelven con la llave y se convierten en los 32 bits de la derecha de la etapa siguiente. La revoltura es compleja y está más allá del ámbito de este libro.

Ventajas y desventajas Los algoritmos de llave secreta tienen una gran ventaja: la eficiencia. Les toma menos tiempo cifrar o descifrar en comparación con los algoritmos de llave pública que estudiaremos en breve. Son muy buenos candidatos para los mensajes largos. Sin embargo, los algoritmos de llave secreta tienen dos grandes desventajas. Cada par de usuarios debe tener una llave secreta. Esto significa que si N personas en el mundo quieren utilizar este método, se necesita que haya $N\ (N\ -\ 1)\ /\ 2$ llaves secretas. Por ejemplo, para que un millón de personas se comuniquen, se requiere medio billón de llaves secretas. Asimismo, la distribución de las llaves entre dos partes puede ser difícil. En breve se verá cómo resolver este problema.

Privacidad con cifrado de llave pública

El segundo tipo de cifrado/descifrado es el **cifrado de llave pública**. En este método, hay dos llaves; una llave privada y una llave pública. La **llave privada** es mantenida por el receptor, y la **llave pública** se anuncia al público (tal vez por medio de Internet).

Cuando un usuario A quiere enviar un mensaje a un usuario B, A utiliza la llave pública para cifrar el mensaje. Cuando B recibe el mensaje, utiliza su llave privada para descifrarlo (figura 16.4).

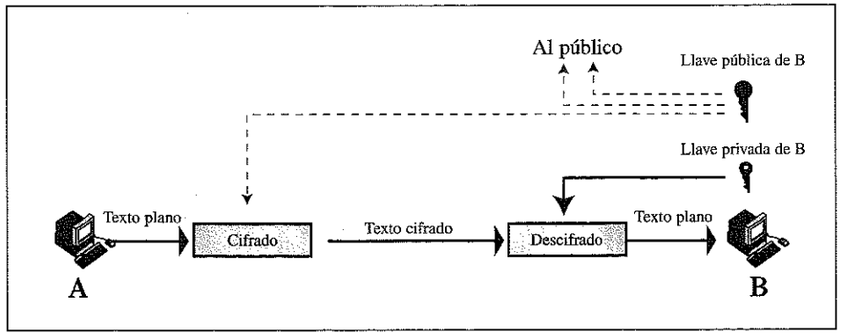


Figura 16.4 Cifrado de llave pública

La idea de este método es que los algoritmos de cifrado y descifrado no son inversos entre sí. Aunque un intruso tenga la llave pública y los algoritmos de cifrado y descifrado, no puede descifrar el mensaje sin la llave privada.

RSA El algoritmo de llave pública más común recibe su nombre en honor a sus inventores, **cifrado de Rivest-Shamir-Adleman (RSA)**. La llave privada es un par de números (N, d); la llave pública es también un par de números (N, e). Observe que N es común a las llaves pública y privada.

El emisor utiliza el siguiente algoritmo para cifrar el mensaje:

$$C = P^e \bmod N$$

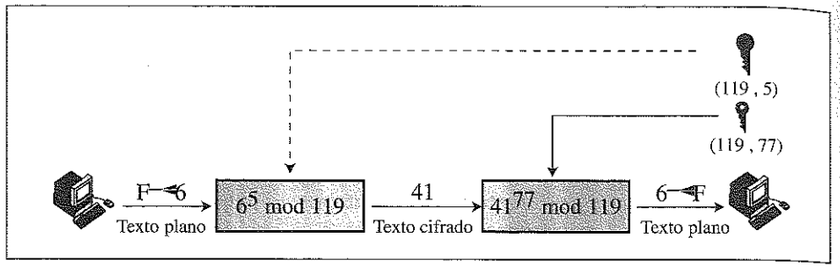


Figura 16.5 RSA

En este algoritmo, P es el texto plano, el cual se representa como un número. C es el número que representa el texto cifrado. Los dos números e y N son componentes de la llave pública. P se eleva a la potencia e y se divide entre N. El término mod indica que el residuo se envía como el texto cifrado.

El receptor utiliza el siguiente algoritmo para descifrar el mensaje:

$$C = P^d \bmod N$$

En este algoritmo, P y C son los mismos que antes. Los dos números d y N son componentes de la llave privada.

He aquí un ejemplo. Imagine que la llave privada es el par (119, 77) y la llave pública es el par (119, 5). La emisora necesita enviar el caracter F. Este caracter puede representarse como el número 6. (F es el sexto caracter en el alfabeto). El algoritmo de cifrado calcula $C = 6^5 \bmod 119 = 41$. Este número se envía al receptor como texto cifrado. El receptor utiliza el algoritmo de descifrado para calcular $P = 41^{77} \bmod 119 = 6$ (el número original). El número 6 luego se interpreta como F. La figura 16.5 muestra el proceso.

El lector puede poner en duda la efectividad de este algoritmo. Si un intruso conoce el algoritmo de cifrado y $N = 119$, lo único que falta es $d = 77$. ¿Por qué no podría un intruso utilizar la prueba y error para encontrar d? La respuesta es que en este ejemplo trivial, un intruso fácilmente podría adivinar el valor de d. Pero un concepto importante del algoritmo RSA es el uso de números muy grandes para d y e. En la práctica, los números son tan grandes (en la escala de decenas de dígitos) que el método de ensayo y error para forzar el código requiere mucho tiempo (meses, si no es que años) incluso con las computadoras más rápidas disponibles hoy en día.

Elección de claves públicas y privadas Una pregunta que viene a la mente es cómo elegir los tres números N, d y e para que el cifrado y el descifrado funcionen. Los inventores del algoritmo RSA probaron matemáticamente que utilizar el procedimiento siguiente garantiza que los algoritmos funcionarán. Aun cuando la prueba esté más allá del ámbito de este libro, damos una idea general del procedimiento:

- Se eligen dos números primos grandes, p y q.
- Se calcula $N = p \times q$.
- Se elige $\boldsymbol{e}$ (menor que N) tal que e y $(p - 1)(q - 1)$ sean relativamente primos (teniendo como factor sólo el uno).
- Se elige $\boldsymbol{d}$ tal que $(e \times d) \bmod [(p - 1)(q - 1)]$ sea igual a uno.

Ventaja y desventaja La ventaja del algoritmo de llave pública es el número de llaves. Cada entidad puede usar el procedimiento anterior para crear un par de llaves, mantener la llave privada (N, d) y distribuir públicamente la otra (N, e). Los individuos pueden incluso publicar su llave pública en su sitio web. Además, observe que en este sistema, para que un millón de usuarios se comuniquen, se necesitan sólo dos millones de llaves, no medio billón como en el caso del algoritmo de llave secreta. Sin embargo, la gran desventaja del método de llave pública es la complejidad del algoritmo. Si usted quiere que el método sea eficaz, necesita números grandes. Calcular el texto cifrado a partir del texto plano utilizando las llaves largas toma mucho tiempo. Ésta es la razón principal de que el cifrado de llave pública no sea recomendable para grandes cantidades de texto.

PRIVACIDAD MEDIANTE LA COMBINACIÓN

Usted puede combinar la ventaja del algoritmo de llave secreta (eficiencia) y la ventaja del algoritmo de llave pública (distribución fácil de las llaves). La llave pública se utiliza para cifrar la llave secreta; la llave secreta se utiliza para cifrar el mensaje. El procedimiento es el siguiente:

1. El emisor elige una llave secreta. Esta llave secreta se llama llave de una sesión; sólo se utiliza una vez.
2. El emisor utiliza la llave pública del receptor para cifrar la llave secreta (como texto) y envía la llave secreta cifrada al receptor. Observe que dijimos que el método de llave pública es bueno para un mensaje corto. Una llave secreta es un mensaje corto.
3. El receptor usa la llave privada para descifrar la llave secreta.
4. El emisor utiliza la llave secreta para cifrar el mensaje actual.

La figura 16.6 muestra la idea de la combinación.

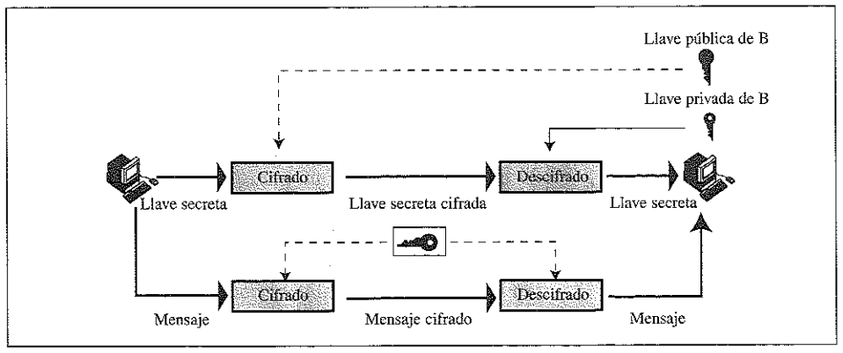


Figura 16.6 Combinación

16.2 FIRMA DIGITAL

Sorprendentemente, los otros tres aspectos de la seguridad (integridad, autenticación y no rechazo) pueden lograrse utilizando un solo concepto. El concepto viene de la firma (autenticación) de un documento por su autor o creador. Cuando un autor firma un documento, éste no puede cambiarse. Como una analogía, si usted tacha algo en un documento como un cheque, debe poner su firma o iniciales por el cambio que hizo. De esta manera, no puede negarlo posteriormente (no rechazo). Cuando usted envía un documento en forma electrónica, puede además firmarlo. A esto se le llama **firma digital**.

Hay dos maneras de hacer una firma digital. Usted puede firmar todo el documento o puede firmar un compendio del documento.

FIRMA DE TODO EL DOCUMENTO

El cifrado de llave pública puede utilizarse para firmar todo el documento. Sin embargo, el uso de la llave pública aquí es diferente de aquel para la llave privada. Aquí el emisor utiliza su llave pública o privada (no la llave privada del receptor) para cifrar el mensaje. El receptor, por otra parte, utiliza la llave pública del emisor (no su llave privada) para descifrar el mensaje. En otras palabras, la llave privada se utiliza para cifrado y la llave pública para descifrado. Esto es posible debido a que los algoritmos de cifrado y descifrado usados actualmente, como RSA, son fórmulas matemáticas y sus estructuras son las mismas. La figura 16.7 muestra cómo se hace esto.

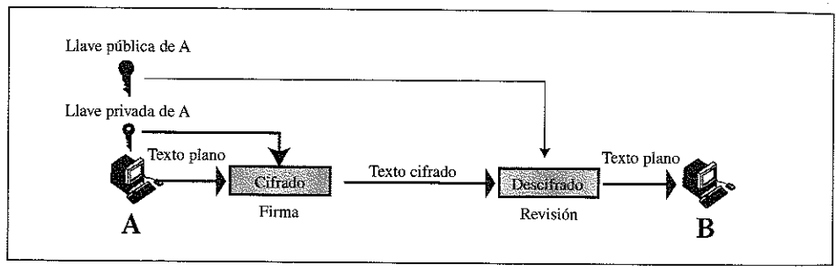


Figura 16.7 Firma de todo el documento

Veamos cómo este tipo de cifrado puede proporcionar integridad, autenticación y no rechazo. Dijimos que la integridad del mensaje se conserva debido a que, si un intruso intercepta el mensaje y lo cambia total o parcialmente, el mensaje descifrado sería (con una alta probabilidad) ilegible e incluso no parecería un mensaje. El mensaje también puede autenticarse debido a que si un intruso envía un mensaje (fingiendo que viene del verdadero autor), utiliza su propia clave privada. El mensaje entonces no se descifra correctamente mediante la llave pública del verdadero autor (es ilegible). El método también proporciona no rechazo. Aunque el emisor puede negar el envío del mensaje, debe revelar (en la corte) su llave privada, la cual debe corresponder con la llave pública. Si usted cifra y descifra el mensaje recibido, obtiene el mensaje enviado.

Observe dos puntos importantes. Primero, no puede proporcionar estos aspectos de la seguridad usando la llave secreta (dejamos el razonamiento para un ejercicio). Segundo, el método no proporciona confidencialidad; cualquiera puede usar la llave pública del emisor para leer el mensaje. Para añadir confidencialidad a esta técnica, se necesita otro nivel de cifrado (ya sea con el cifrado de llave secreta o de llave pública).

FIRMA DEL COMPENDIO

Mencionamos anteriormente que una de las desventajas del cifrado de llave pública es que es muy ineficiente. Esto es verdad cuando se utiliza el cifrado de llave pública para firmar todo el documento (todo el documento debe cifrarse y descifrarse).

Para hacer el proceso más eficiente, usted puede dejar que el emisor firme un compendio del documento en lugar de todo el documento. En otras palabras, el emisor hace una miniatura del documento y la firma (la cifra con su llave privada); el receptor luego revisa la firma de la miniatura (la descifra con la llave pública del emisor).

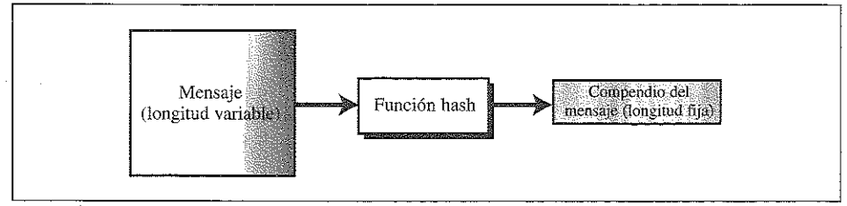


Figura 16.8 Firma del compendio

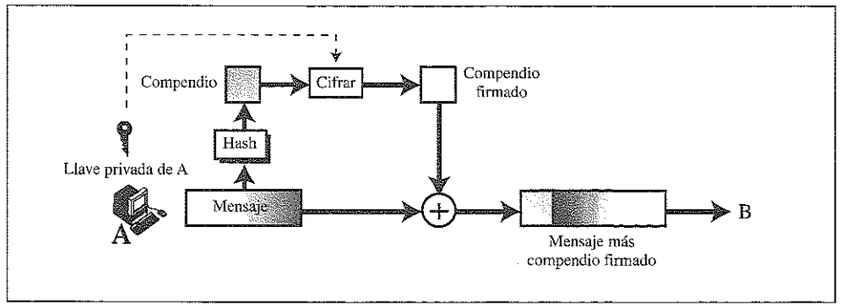


Figura 16.9 Sitio del emisor

El método utiliza una técnica llamada función hash para crear un compendio del mensaje. No importa qué longitud tenga el mensaje, el compendio es de tamaño fijo (por lo general, 128 bits) como se aprecia en la figura 16.8.

Las dos funciones hash más comunes son Message Digest 5 (MD5) y Secure Hash Algoritm 1 (SHA-1). El primero produce un compendio de 128 bits. El segundo produce un compendio de 160 bits.

Observe que la función hash tiene dos propiedades para garantizar su éxito. Primero, el hashing debe ser en un sentido; el compendio sólo puede crearse a partir del mensaje, pero no viceversa. Segundo, el hashing debe ser uno a uno; debe ser muy difícil encontrar dos mensajes que crean el mismo compendio. La razón de esta condición se verá en breve.

Después de que se ha creado el compendio, éste se cifra (firma) usando la llave privada del emisor. El compendio cifrado se adjunta al mensaje original y se envía al receptor. La figura 16.9 muestra el sitio del emisor.

El receptor recibe el mensaje original y el compendio cifrado, y los separa. Aplica la misma función hash al mensaje para crear un segundo compendio. También descifra el compendio recibido usando la llave pública del emisor. Si los dos compendios son iguales, es obvio que los tres aspectos de la seguridad se conserven. La figura 16.10 muestra el sitio del receptor.

Sabemos que estas tres propiedades se conservan para la copia del compendio recibido por el receptor. Ahora veamos por qué estas propiedades se conservan para el mensaje.

1. El compendio no ha cambiado y el mensaje crea una réplica del compendio. Así que el mensaje no ha cambiado (recuerde, dos mensajes no pueden crear el mismo compendio).
2. El compendio viene del verdadero emisor, así que el mensaje también viene del verdadero emisor. Si un intruso hubiera iniciado el mensaje, el mensaje no habría creado el mismo compendio.
3. El emisor no puede negar el mensaje debido a que no puede negar el compendio; el único mensaje que puede crear ese compendio es el mensaje recibido.

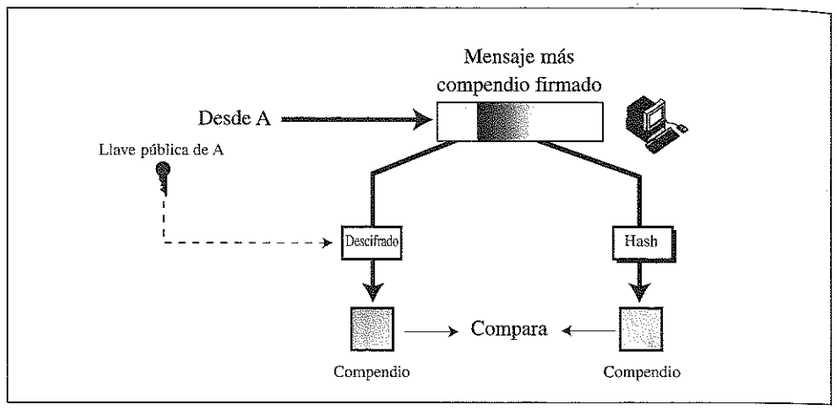


Figura 16.10 Sitio del receptor

De nuevo observe que este método de firma digital no proporciona privacidad; si la privacidad se requiere, debe añadirse usando otro nivel de cifrado ya sea con cifrado de llave secreta, con cifrado de llave pública o con ambos.

16.3 TÉRMINOS CLAVE

autenticación
cifrado
cifrado de llave pública
cifrado de Rivest-Shamir-Adleman (RSA)
descifrado
estándar de cifrado de datos (DES)
firma digital
llave privada
llave pública
llave secreta
no rechazo
permutación
seguridad
texto cifrado
texto plano

16.4 RESUMEN

- La seguridad involucra los temas de privacidad, autenticación, integridad y no rechazo.
- La privacidad se logra a través del cifrado. El cifrado traduce un mensaje (texto plano) ininteligible para el personal no autorizado.
- Un método de cifrado/descifrado puede clasificarse ya sea como un método de llave secreta o un método de llave pública.
- En el cifrado de llave secreta, sólo el emisor y el receptor conocen la llave.
- DES es un método de cifrado de llave secreta popular.
- En el cifrado de llave pública, la llave pública es conocida por todos, pero la llave privada es conocida sólo por el receptor.
- Un método de cifrado de llave pública de uso común se basa en el algoritmo RSA.
- La autenticación, la integridad y el no rechazo se logran a través de un método llamado firma digital.
- Usted puede usar la firma digital en todo el documento o en un compendio del documento.

16.5 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Cuáles son las cuatro condiciones necesarias para la seguridad de los datos transmitidos?
2. ¿Cómo se puede asegurar la privacidad de un mensaje?
3. ¿Cómo se puede autenticar al emisor de un mensaje?
4. ¿Cómo puede usted conservar la integridad de un mensaje?
5. Dé un ejemplo de no rechazo.
6. ¿Cuáles son las dos categorías principales de métodos de cifrado?
7. ¿Cómo se relaciona el texto plano con el texto cifrado?
8. ¿Qué es DES?
9. Comente la llave secreta, la llave pública y la llave privada. ¿Quién tiene posesión de cada llave? ¿Cuál es el tipo de cifrado que utiliza cada llave?
10. ¿Por qué el algoritmo RSA es tan poderoso?
11. ¿Cuáles son las desventajas del cifrado de llave secreta?
12. ¿Cuál es la desventaja del cifrado de llave pública?
13. ¿Cuál es la diferencia entre firmar todo el documento y firmar el compendio?
14. ¿Cómo se relaciona la firma digital con los asuntos de privacidad?

PREGUNTAS DE OPCIÓN MÚLTIPLE

15. En el cifrado/descifrado, la llave _____ es conocida por todos.
    a. secreta
    b. privada
    c. pública
    d. reducida
16. La (el) _____ se logra a través del cifrado/descifrado.
    a. autenticación
    b. integridad
    c. privacidad
    d. no rechazo
17. En el método de cifrado y descifrado de llave secreta, _____ posee(n) la llave secreta.
    a. sólo el emisor
    b. sólo el receptor
    c. tanto el emisor como el receptor
    d. el público en general
18. Una de las ventajas del cifrado de llave pública es _____.
    a. el poco tiempo requerido para el cifrado/descifrado
    b. que se requiere un número pequeño de llaves
    c. que se conserva la integridad
    d. que todos conocen todas las llaves
19. El algoritmo RSA es la base de un método de cifrado _____.
    a. llave pública
    b. llave secreta
    c. llave privada
    d. todas las anteriores
20. Para crear un compendio de un documento, usted puede utilizar _____.
    a. una firma digital
    b. un número primo
    c. DES
    d. una función hash
21. En el método de firma digital, el emisor utiliza su llave _____ para cifrar el mensaje.
    a. pública
    b. privada
    c. secreta
    d. reducida
22. En el método de firma digital, el receptor utiliza la llave _____ para descifrar el mensaje.
    a. pública
    b. privada
    c. secreta
    d. reducida
23. ¿Qué representa 10 mod 3?
    a. 1
    b. 3
    c. 3.33
    d. 10
24. En una firma digital que involucra un compendio, la función hash se necesita _____.
    a. sólo en el receptor
    b. sólo en el emisor
    c. tanto en el emisor como en el receptor
    d. por el público en general

25. El método de firma digital no proporciona _____.
    a. privacidad
    b. autenticación
    c. integridad
    d. no rechazo

EJERCICIOS

26. Uno de los primeros métodos de llave secreta se llamaba sustitución monoalfabética (o la Cifra de César, atribuido a Julio César). En este método, cada caracter en texto plano se mueve hacia delante *n* caracteres. Los caracteres se envuelven de ser necesario. Por ejemplo, si *n* es 5, el caracter A se remplaza por F, el caracter B por G y así sucesivamente. ¿Cuál es la llave aquí? ¿Cuál es el algoritmo de cifrado? ¿Cuál es el algoritmo de descifrado?

27. Usando la Cifra de César y 6 como la llave, cifre el mensaje Hola.

28. Comente la efectividad de la Cifra de César. ¿Puede un intruso adivinar la llave buscando sólo en el texto cifrado? De ser así, ¿cómo?

29. Una de las operaciones utilizadas en un algoritmo de llave secreta es la permutación de los bits. Un texto plano de 8 bits se permuta (revuelve). El bit 1 se convierte en el bit 3, el bit 2 se convierte en el bit 7 y así por el estilo. Dibuje un diagrama que muestre el cifrado y el descifrado. Elija su propia revoltura. ¿Cuál es la llave aquí? ¿Cuál es el algoritmo de cifrado? ¿Cuál es el algoritmo de descifrado?

30. Una operación en un algoritmo de llave secreta es la operación XOR (consulte el capítulo 4). A un patrón de bits de tamaño fijo (texto plano) se le aplica XOR con el mismo tamaño de patrón de bits (llave) para crear un texto cifrado de tamaño fijo. ¿Cuál es el algoritmo de cifrado aquí? ¿Cuál es el algoritmo de descifrado? Considere el hecho de que un algoritmo XOR es un algoritmo reversible.

31. Utilice la llave pública (15, 3) para cifrar el número 7. Use la llave privada (15, 11) para descifrar el resultado del cifrado anterior. Haga un diagrama que muestre el flujo de información entre el emisor y el receptor.

32. Pruebe que el papel de la llave pública y el papel de la llave privada pueden cambiar al repetir el ejercicio previo, pero cifre el número 7 con la llave privada (15, 11) y descífrelo con la llave pública (15, 3). Haga un diagrama para mostrar el flujo de información entre el emisor y el receptor.

33. Comente por qué el cifrado/descifrado de llave secreta no puede usarse para el no rechazo.

34. Comente por qué el cifrado/descifrado de llave secreta no puede usarse para la autenticación.

35. Añada un nivel de cifrado/descifrado de llave secreta a la figura 16.7 para proporcionar privacidad.

36. Añada un nivel de cifrado/descifrado de llave pública a la figura 16.7 para proporcionar privacidad.

Teoría de la computación

Varias preguntas inquietaron a los científicos de la computación al principio de la era de las computadoras:

- ¿Cuáles problemas pueden resolverse mediante una computadora? ¿Cuáles no?
- ¿Cuánto tiempo se lleva la solución de un problema usando un lenguaje en particular?
- ¿Un lenguaje es superior a otro? Es decir, ¿un programa de computadora escrito en un lenguaje puede resolver un problema que otro no puede?
- ¿Cuál es el número mínimo de instrucciones necesarias para que un lenguaje resuelva un problema?
- Antes de ejecutar un programa, ¿puede determinarse si el programa se parará (terminará) o se ejecutará por siempre?

Para responder estas preguntas, pasamos a una disciplina llamada *teoría de la computación*. Primero, presentamos un lenguaje, llamado el Lenguaje simple, para mostrar que el número mínimo de instrucciones requeridas para resolver cualquier problema que tiene solución mediante una computadora es tres. En otras palabras, probamos que todos los lenguajes que tienen estas tres instrucciones básicas son iguales. Segundo, presentamos otra herramienta, un modelo de computadora llamado la máquina de Turing. Mostramos que un problema que puede resolverse mediante nuestro Lenguaje simple, también puede resolverse por la máquina de Turing. Tercero, probamos que ningún programa puede decir si otro programa se detiene o no. Esta prueba es en sí misma una indicación de que hay problemas que no pueden resolverse mediante una computadora. Finalmente, comentamos brevemente la complejidad de los algoritmos.

17.1 LENGUAJE SIMPLE

Podemos definir un lenguaje con sólo tres instrucciones: de *incremento*, de *decremento* y el ciclo *while* (figura 17.1). En este lenguaje, usted utiliza sólo el tipo de datos de enteros. No hay necesidad de ningún otro tipo de datos porque usted puede simular otros tipos de datos con el tipo entero. El lenguaje utiliza sólo unos cuantos símbolos como { y }.

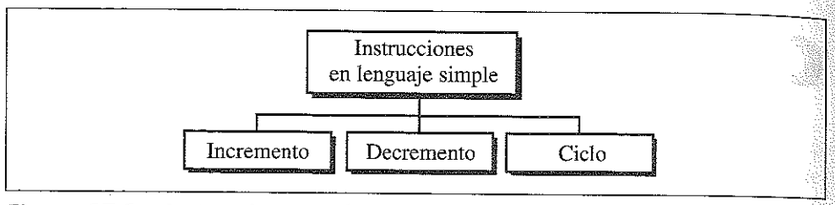


Figura 17.1 Instrucciones en lenguaje simple

INSTRUCCIÓN DE INCREMENTO

La **instrucción de incremento** añade un 1 a la variable (por ejemplo, X). El formato es:

```
incr X
```

INSTRUCCIÓN DE DECREMENTO

La **instrucción de decremento** sustrae un 1 de la variable (por ejemplo, X). El formato es:

```
decr X
```

INSTRUCCIÓN DE CICLO

La **instrucción de ciclo** repite una acción (o una serie de acciones) cuando el valor de la variable (por ejemplo, X) no es cero. El formato es:

```
while X
{
  Acción(es)
}
```

EL PODER DEL LENGUAJE SIMPLE

De manera inductiva, podemos probar que este lenguaje de programación simple con sólo tres instrucciones es tan poderoso (aunque no necesariamente tan eficiente) como cualquier lenguaje sofisticado en uso hoy en día, por ejemplo C. Para hacerlo, mostramos cómo podemos simular varias instrucciones encontradas en algunos lenguajes populares.

Macros en el Lenguaje simple

Llamamos a cada simulación una **macro** y la utilizamos en otras simulaciones sin la necesidad de repetir el código.

Primera macro: X ⟵ 0 El código siguiente muestra cómo utilizar las instrucciones en este lenguaje para asignar 0 a la variable X. La notación de macros para esto es X ⟵ 0. A veces se le llama limpiar una variable.

```
while X
{
  decr X
}
```


Segunda macro: X ⟵ n El código siguiente muestra cómo utilizar las instrucciones en este lenguaje para asignar un entero positivo a la variable X. La notación de macros para esto es X ⟵ n. Primero limpie la variable X; luego incremente la X *n* veces.

```
X ⟵ 0
incr X
incr X
. . .
. . .
. . .
incr X
```

Tercera macro: Y ⟵ X Todo lenguaje de programación tiene una instrucción que copia el valor de una variable a otra sin perder el valor de la variable original. Esta macro puede simularse en el Lenguaje simple utilizando la siguiente serie de instrucciones. Observe que el segundo ciclo utiliza una variable temporal (TEMP) para restaurar el valor de X.

```
Y    ⟵ 0
TEMP ⟵ 0
while X
{
  incr Y
  decr X
  incr TEMP
}
while TEMP
{
  decr TEMP
  incr X
}
```

Cuarta macro: Z ⟵ X + Y Esta macro suma los valores de X y Y y almacena el resultado en Z. Esto se hace en dos pasos. Primero almacena el valor de X en Z (tercera macro) y luego incrementa la Z Y veces.

```
Z    ⟵ X
TEMP ⟵ Y
while TEMP
{
  incr Z
  decr TEMP
}
```

Quinta macro: Z ⟵ X * Y Esta macro multiplica los valores de X y Y y almacena el resultado en Z. La cuarta macro se utiliza porque la multiplicación es una suma repetida.

```
Z    ⟵ 0
TEMP ⟵ Y
while TEMP
{
  Z ⟵ X * Y
  decr TEMP
}
```

Sexta macro: `Z ⟵ X ** Y` Esta macro eleva `X` a la potencia `Y` y almacena el resultado en `Z`. Esto se hace utilizando la multiplicación porque la exponenciación es una multiplicación repetida.

```
Z    ⟵ 1
TEMP ⟵ Y
while TEMP
{
  Z ⟵ Z * X
  decr TEMP
}
```

Séptima macro: `comp (X)` Esta macro complementa el valor de `X`. Si el valor de `X` es 0 (falso), lo cambia a 1 (verdadero). Si no es 0 (verdadero), lo cambia a 0 (falso). El primer ciclo cambia el valor de `X` a 0 si éste no es 0 (positivo). Si usted introduce este ciclo (`X` no es 0), entonces el valor de `TEMP` se establece en 0, lo cual significa que usted nunca introduce el segundo ciclo. Si usted nunca introduce el primer ciclo (`X` es 0), entonces definitivamente introduce el segundo ciclo y cambia el valor de `X` a 1 (el ciclo se itera sólo una vez porque el valor de `TEMP` es 1).

```
TEMP ⟵ 1
while X
{
  X    ⟵ 0
  TEMP ⟵ 0
}
while TEMP
{
  incr (X)
  decr TEMP
}
```

Octava macro: `if X then A1 else A2` Esta macro simula la instrucción de toma de decisiones (if-then-else) de los lenguajes modernos. Si el valor de `X` no es 0, A1 (una acción o una serie de acciones) se ejecuta en el primer ciclo. Sin embargo, el ciclo se ejecuta sólo una vez porque, después de la primera iteración, el valor de `TEMP` se vuelve 0 y usted sale del ciclo. Si el valor de `X` es 0, el primer ciclo se omite. El valor de `TEMP`, el cual es lo mismo que el valor de `X`, se complementa (se vuelve no 0) y el segundo ciclo se ejecuta sólo una vez.

```
TEMP ⟵ X
while TEMP
{
  A1
  TEMP ⟵ 0
}
  TEMP ⟵ X
  comp (TEMP)
  while TEMP
{
  A2
  TEMP ⟵ 0
}
```

Otras macros Tal vez haya adivinado que necesita más macros para hacer el Lenguaje simple compatible con los lenguajes contemporáneos. La creación de otras macros es posible, aunque no trivial. Hemos dejado algunas macros desafiantes como ejercicios.

Entrada y salida

Tal vez se haya planteado preguntas sobre las instrucciones de entrada/salida. Por ejemplo, ¿cómo puede leer datos en una variable y cómo puede imprimir el resultado de un programa? Para un lenguaje como éste, no hay necesidad de entrada o salida. Usted puede simular la entrada, como `read X`, mediante una instrucción de asignación (`X ⟵ n`). También puede simular la salida al suponer que la última variable utilizada en un programa aloja lo que debería imprimirse. Recuerde que éste no es un lenguaje práctico, está diseñado para probar algunos teoremas en las ciencias de la computación.

CONCLUSIÓN

El Lenguaje simple es tan poderoso como cualquier otro lenguaje analizado en el capítulo 9. Esto significa que si usted no puede resolver un problema en este lenguaje, no podrá resolverlo en ningún otro lenguaje. Posteriormente, mostramos que el *problema de paro* (*halting*) no tiene solución en ningún lenguaje porque no tiene solución en el Lenguaje simple.

17.2 MÁQUINA DE TURING

La **máquina de Turing** se introdujo en 1936 por Alan M. Turing para resolver problemas computables. Es la base de las computadoras modernas. En esta sección, presentamos una versión muy simplificada de esta máquina para mostrar cómo funciona. Luego mostramos cómo implementa las instrucciones en el Lenguaje simple.

COMPONENTES DE LA MÁQUINA DE TURING

Una máquina Turing está formada por tres componentes: una cinta, un controlador y una cabeza de lectura/escritura (figura 17.2).

Cinta

Aun cuando las computadoras modernas utilizan un dispositivo de acceso aleatorio con capacidad finita, la memoria de la máquina de Turing es infinita. La **cinta**, en cualquier momento mantiene una secuencia de caracteres del conjunto de caracteres aceptado por la máquina. Para nuestro propósito, suponemos que la máquina puede aceptar sólo unos cuantos símbolos: el signo de número (#), el ampersand (&), el dígito 1 y el espacio en blanco. La figura 17.3 muestra un ejemplo de los datos en una cinta en esta máquina. El # define el principio del número, el número almacenado en la cinta se representa mediante 11111 y el & define el final del número. El resto de la cinta contiene caracteres en blanco.

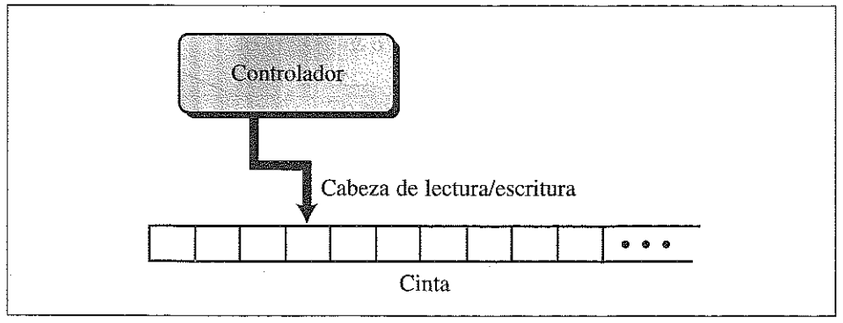


Figura 17.2 Máquina de Turing

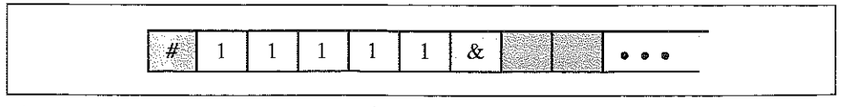


Figura 17.3 Cinta

También suponemos que la cinta procesa sólo los datos enteros positivos representados en la aritmética unaria. En esta aritmética, un entero positivo se forma sólo por unos. Por ejemplo, el entero 4 se representa como 1111 y el entero 7 se representa como 1111111. La ausencia de unos representa el 0.

Cabeza de lectura/escritura

La cabeza de lectura/escritura en cualquier momento señala a un símbolo en la cinta. Llamamos a este símbolo el símbolo actual. La cabeza de lectura/escritura lee y escribe un símbolo a la vez desde la cinta. Después de leer y escribir se mueve a la izquierda, a la derecha o permanece en su lugar. La lectura, la escritura y el desplazamiento, todos se realizan bajo instrucciones del controlador.

Controlador

El **controlador** es la contraparte teórica de la unidad central de procesamiento (CPU) en las computadoras modernas. Es un autómata de estado finito, una máquina que tiene un número finito predeterminado de estados y se mueve de un estado a otro con base en la entrada. En cualquier momento puede estar en uno de estos estados.

La figura 17.A exhibe el diagrama de transición para un controlador que tiene un autómata de estado finito. En esta figura el autómata tiene cuatro estados (A, B, C, D). El diagrama muestra el cambio de estado como una función de la lectura de caracteres. Lo siguiente describe la figura:

- Si el controlador está en el estado A y lee 1 o un espacio en blanco, pasa al estado B. Si lee # o &, pasa al estado C.
- Si el controlador está en el estado B y lee 1, pasa al estado C. Si lee cualquier otro carácter pasa al estado A.
- Si el controlador está en el estado C y lee 1, pasa al estado B. Si lee cualquier otro carácter pasa al estado D.
- Si el controlador está en el estado D y lee cualquier carácter excepto un espacio en blanco, pasa al estado B. Pero si lee un espacio en blanco, permanece en el estado D.

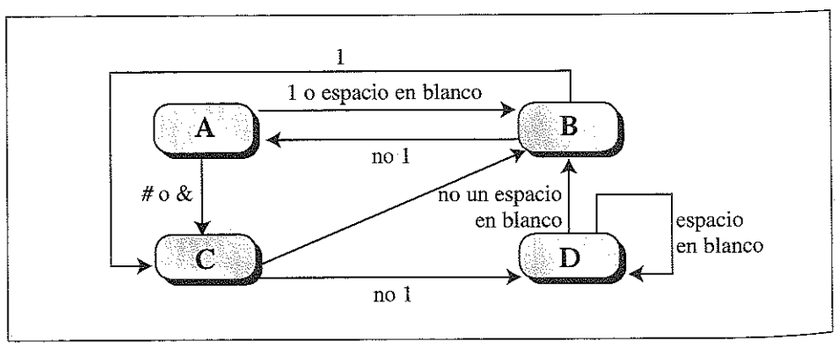


Figura 17.4 Estado de transición

Para cada lectura de un símbolo, el controlador escribe un carácter, define la siguiente posición de la cabeza de lectura/escritura y cambia el estado. En el diagrama de transición, mostramos sólo uno de estos tres; una tabla de transición puede mostrar los tres. La tabla de transición, como se puede ver en la tabla 17.1, tiene cinco columnas: el estado actual, el caracter leído, el caracter escrito, la siguiente posición de la cabeza de lectura/escritura y el nuevo estado.

Para cada problema, debemos definir la tabla correspondiente. Esto es similar a un programa escrito en un lenguaje de computadora. Un programa es una implementación moderna de la tabla de transición.

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
A	1 o espacio en blanco	#	⟶	B
A	# o &	&	⟵	C
B	1	1	⟵	C
B	no 1	lo mismo que lee		A
C	1	espacio en blanco	⟶	B
C	no 1	1	⟶	D
D	no un espacio en blanco	lo mismo que lee	⟶	B
D	espacio en blanco	1	⟵	D

Tabla 17.1 Tabla de transición

SIMULACIÓN DE LENGUAJE SIMPLE

Veamos si podemos escribir programas (crear tablas de transición) que implementen las instrucciones del Lenguaje simple.

Instrucción de incremento

Implementemos la instrucción (`incr X`) utilizando la máquina de Turing. La figura 17.5 muestra el diagrama de transición para esta instrucción. Interpretamos la `X` como los datos que ya están en la cinta delimitados por el signo # (al principio) y el signo & (al final). Por razones de simplicidad, en la figura 17.5 hemos omitido algunos estados (por ejemplo, el estado de error).

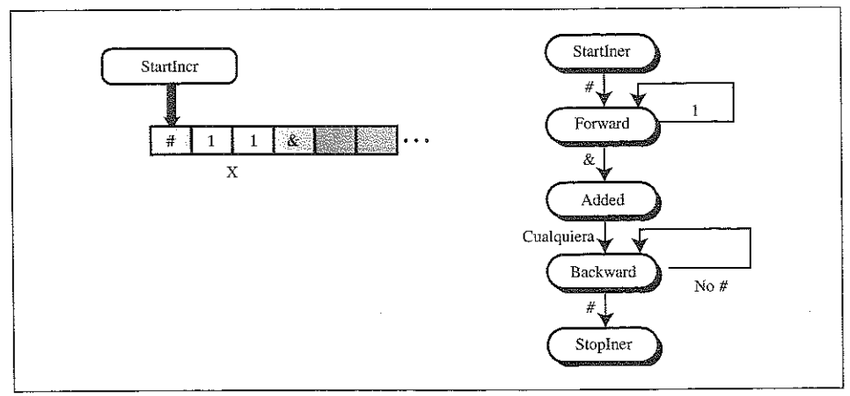


Figura 17.5 Diagrama de transición para `incr X`

La tabla 17.2 muestra la tabla de transición para esta instrucción.

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
StartIncr	#	#	⟶	Forward
Forward	1	1	⟶	Forward
Forward	&	1	⟶	Added
Added	cualquiera	&	⟵	Backward
Backward	no #	lo mismo que lee	⟵	Backward
Backwar	#	#	a ningún lugar	**StopIncr**

Tabla 17.2 Tabla de transición para la instrucción `incr X`

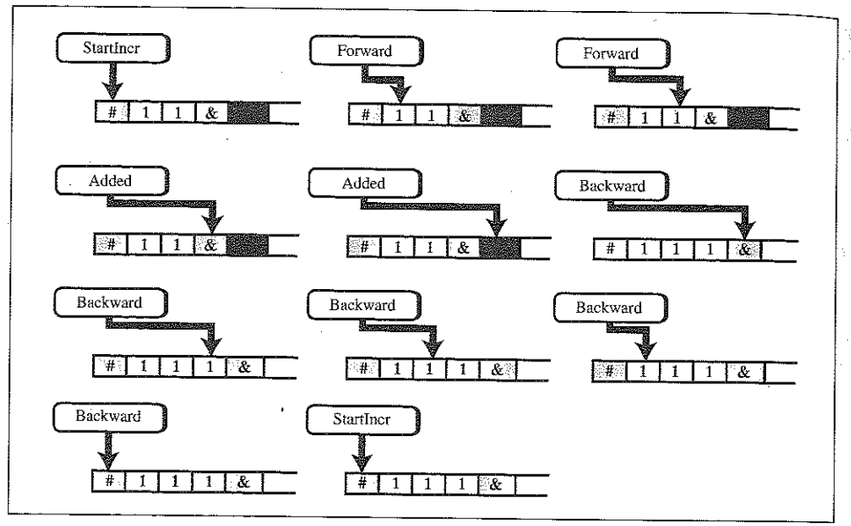


Figura 17.6 Pasos en la instrucción `incr X`

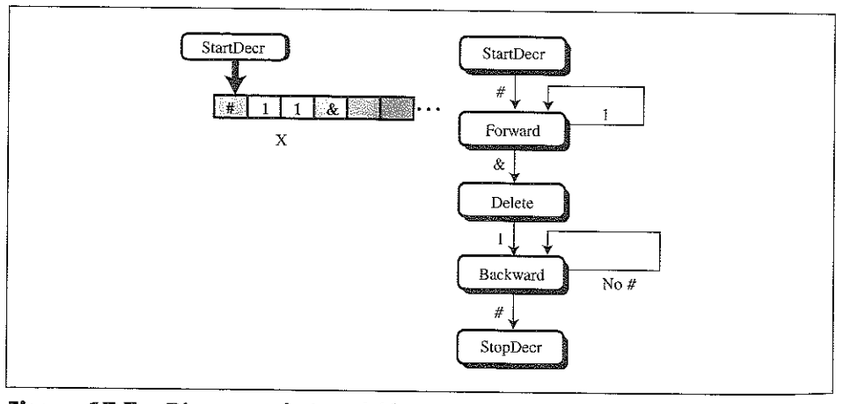


Figura 17.7 Diagrama de transición para `decr X`

La figura 17.6 muestra cómo el estado del controlador cambia y cómo se mueve la cabeza de lectura/escritura.

Instrucción de decremento

La instrucción de decremento (`decr X`) es similar a la instrucción de incremento. La figura 17.7 muestra el diagrama de transición.

La tabla 17.3 muestra la tabla de transición para esta instrucción.

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
StartDecr	#	#	→	Forward
Forward	1	1	→	Forward
Forward	&	blanco	←	Delete
Delete	1	&	←	Backward
Backward	no #	lo mismo que lee	←	Backward
Backwar	#	#	a ningún lugar	**StopDecr**

Tabla 17.3 Tabla de transición para la instrucción `decr X`

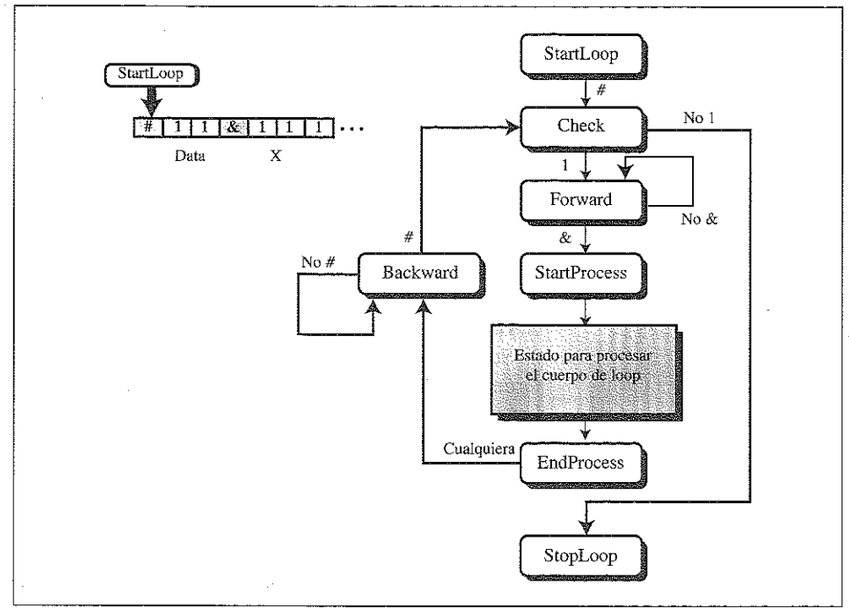


Figura 17.8 Diagrama de transición para la instrucción loop

Instrucción loop

Para simular el ciclo, suponga que X se almacena después del símbolo #. La & marca el final de X y el principio de los datos que se procesan en el cuerpo del ciclo (éste puede ser más que un solo elemento de datos). La figura 17.8 muestra el diagrama de transición.

La tabla 17.4 muestra la tabla de transición.

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
StartLoop	#	#	→	Check
Check	No 1	Lo mismo que se lee	←	**StartLoop**
Check	1	1	→	Forward
Forward	No &	Lo mismo que se lee	→	Forward
Forward	&	&	Ninguno	StartProcess
...	...	...	...	...
...	...	...	...	...
EndProcess	Cualquiera	Lo mismo que se lee	←	Backward
Backward	No #	Lo mismo que se lee	←	Backward
Backward	#	#	Ninguno	Check

Tabla 17.4 Tabla de transición para la instrucción loop

CONCLUSIÓN

La máquina de Turing es tan poderosa como nuestro Lenguaje simple. Cualquier problema que puede resolverse mediante el Lenguaje simple también puede resolverse por medio de la máquina de Turing. Pero ¿qué hay del otro lado de la moneda? ¿Hay un problema que pueda resolverse mediante la máquina de Turing que no se pueda resolver mediante el Lenguaje simple? Aunque no podemos probarlo, durante las décadas posteriores, los científicos de la computación se convencieron de que esto no puede ocurrir. A esto se le conoce como la tesis de Church (en honor a Alonzo Church), la cual establece que los lenguajes simples como el Lenguaje simple y la máquina de Turing son equivalentes. Todos los problemas que pueden resolverse mediante el Lenguaje simple también pueden resolverse por medio de la máquina de Turing y viceversa.

17.3 NÚMEROS DE GÖDEL

En la ciencia de la computación teórica, un número sin asignar se asigna a cada programa que puede escribirse en un lenguaje específico. Por lo general, a esto se le llama el **número de Gödel** (en honor a Kurt Gödel).

Esta asignación tiene muchas ventajas. Primero, los programas pueden utilizarse como un solo elemento de datos que sirva como entrada a otro programa. Segundo, se puede hacer referencia a los programas simplemente mediante sus representaciones de enteros. Tercero, la numeración puede usarse para probar que algunos problemas no pueden resolverse mediante una computadora al mostrar que el número total de problemas en el mundo es mucho más grande que el número total de programas que puedan escribirse.

Se han concebido distintos métodos para numerar los programas. Utilizamos una transformación muy simple para numerar programas escritos en nuestro Lenguaje simple. El Lenguaje simple usa sólo 15 símbolos (tabla 17.5). Observe que en este lenguaje usted utiliza sólo `X`, `X1`, `X2`, . . . , `X9` como variables. Para codificar estas variables, se maneja `Xn` como dos símbolos `X` y `n` (`X3` es `X` y `3`). Si usted tiene una macro con otras variables, éstas necesitan cambiarse a `Xn`.

Símbolo	Código hexadecimal	Símbolo	Código hexadecimal
1	1	9	9
2	2	**incr**	A
3	3	decr	B
4	4	while	C
5	5	{	D
6	6	}	E
7	7	X	F
8	8		

Tabla 17.5 Código para símbolos utilizados en el Lenguaje simple

REPRESENTACIÓN DE UN PROGRAMA

Utilizando la tabla, usted puede representar cualquier programa escrito en nuestro Lenguaje simple mediante un entero positivo único. Siga estos pasos:

1. Remplace cada símbolo con el código hexadecimal que le corresponde en la tabla.
2. Interprete el número hexadecimal resultante como un entero sin asignar.

EJEMPLO 1

¿Cuál es el número de Gödel para el programa `incr X`?

SOLUCIÓN

Se remplaza cada símbolo mediante su código hexadecimal:

```
incr    X
A       F
175 (en decimal)
```

De modo que este programa puede representarse mediante el número 175. ■

INTERPRETACIÓN DE UN NÚMERO

Para mostrar que el sistema de numeración es único, utilice los pasos siguientes para interpretar un número de Gödel:

1. Convierta el número a hexadecimal.
2. Interprete cada **dígito hexadecimal** como un símbolo utilizando la tabla 17.5 (ignore un 0).

Observe que mientras cualquier programa escrito en Lenguaje simple puede representarse mediante un número, no todos los números pueden interpretarse como un programa válido. Después de la conversión, si los símbolos no siguen la sintaxis del lenguaje, el número no es un programa válido.

EJEMPLO 2

Interprete 3058 como un programa.

SOLUCIÓN

El número se cambia a hexadecimal y se remplaza cada dígito con el símbolo correspondiente:

```
3058
B  F 2
decr X 2
```

Observe que en nuestro Lenguaje simple cada programa incluye entrada y salida. Esto significa que la combinación de un programa y sus entradas define el número de Gödel. ■

17.4 PROBLEMA DE PARO

Casi todo programa escrito en un lenguaje de programación implica la repetición (ciclos o funciones recursivas). Un constructor de repetición puede no terminar (pararse o detenerse); es decir, un programa puede ejecutarse por siempre si éste tiene un ciclo infinito. Por ejemplo, el programa siguiente en Lenguaje simple nunca termina.

```
X ⟵ 1
while X
{
}
```

Una pregunta de programación clásica es:

> **¿Puede escribirse un programa que evalúe si un programa, representado por su número de Gödel, terminará o no?**

La existencia de este programa ahorraría a los programadores mucho tiempo. La ejecución de un programa sin saber si éste se detiene o no, es una tarea tediosa. Lamentablemente, ahora se ha probado que este programa no puede existir (una gran decepción para los programadores).

EL PROBLEMA DE PARO NO TIENE SOLUCIÓN

En vez de decir que el programa evaluador no existe y nunca podrá existir, el científico de la computación dice: "El problema de paro no tiene solución".

Prueba

Demos una prueba informal sobre la inexistencia de este programa evaluador. Nuestro método a menudo se utiliza en las matemáticas: Suponga que éste existe y luego muestre que su existencia crea una contradicción. Por consiguiente, no puede existir. En este método usamos tres pasos para mostrar la prueba.

PASO 1 En este paso, suponemos que existe un programa, llamado Evaluar. Éste puede aceptar cualquier programa como *P*, representado por su número de Gödel, como entrada y produce una salida ya sea de 1 o 0. Si *P* termina, la salida de Evaluar es 1; si *P* no termina, la salida de Evaluar es 0 (figura 17.9).

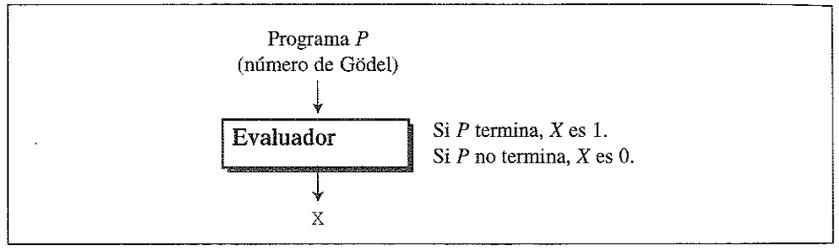


Figura 17.9 Paso 1 en prueba

PASO 2 En este paso, creamos otro programa llamado Extraño que se compone de dos partes: una copia de Evaluar al principio y un ciclo vacío (ciclo con un cuerpo vacío) al final. El ciclo utiliza *X* como la variable de prueba, la cual en realidad es la salida del programa Evaluar. Este programa también utiliza *P* como la entrada. Llamamos a este programa Extraño por la siguiente razón. Si *P* termina, la primera parte de Extraño, la cual es una copia de Evaluar, produce una salida de 1. Este 1 es la entrada del ciclo. El ciclo no termina (ciclo infinito) y, en consecuencia, Extraño no termina. Si *P* no termina, la primera parte de Extraño, la cual es una copia de Evaluar, produce una salida de 0. Este 0 es la entrada del ciclo; el ciclo termina (ciclo finito) y en consecuencia Extraño termina.

En otras palabras, tenemos dos situaciones extrañas:

Si *P* termina, Extraño no termina.
Si *P* no termina, Extraño termina.

La figura 17.11 muestra el paso dos en la prueba.

PASO 3 Ahora, al haber hecho el programa Extraño, probamos este programa consigo mismo (su número de Gödel) como entrada. Esto es legítimo porque no ponemos ninguna restricción a *P*. La figura 17.10 muestra la situación.

Contradicción ¿Ve alguna contradicción?

Si suponemos que Evaluar existe, tenemos las siguientes contradicciones:

Extraño no termina si Extraño termina.

Extraño termina si Extraño no termina.

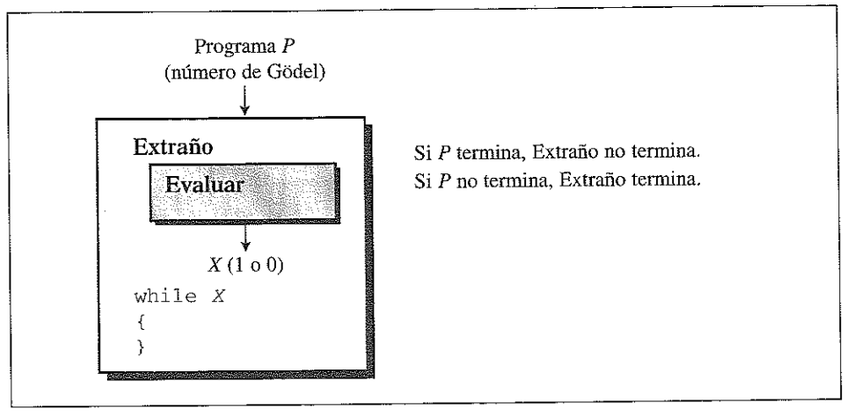


Figura 17.10 Paso 2 en la prueba

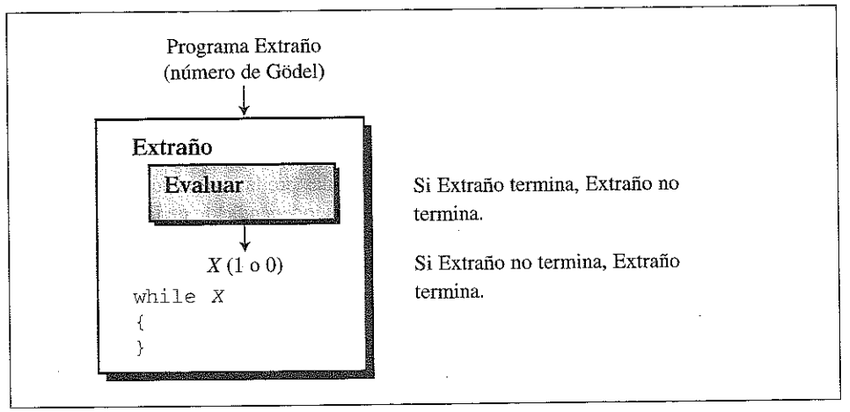


Figura 17.11 Paso 3 en prueba

Esto prueba que el programa Evaluar no puede existir y que debemos dejar de buscarlo. El problema de paro no tiene solución.

17.5 PROBLEMAS CON SOLUCIÓN Y SIN SOLUCIÓN

Ahora que hemos demostrado que al menos un problema no tiene solución mediante una computadora, tratemos este tema importante un poco más. En las ciencias de la computación podemos decir que, en general, los problemas se dividen en dos categorías: **problemas con solución** y **problemas sin solución**. Los problemas con solución se dividen a su vez en dos categorías: problemas polinomiales y no polinomiales (figura 17.12).

PROBLEMAS SIN SOLUCIÓN

Existe un número infinito de problemas que no pueden resolverse mediante una computadora, uno de ellos es el problema de paro. Un método para probar que un problema no tiene solución es mostrar que si ese problema tiene solución, el problema de paro también tiene solución. En otras palabras, probar que la capacidad de solución de un problema da como resultado la capacidad de solución del problema de paro.

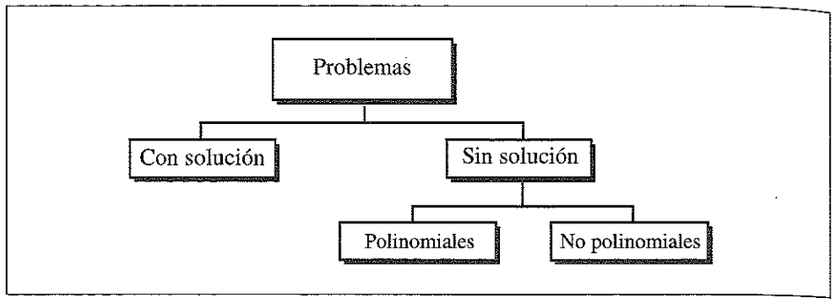


Figura 17.12 Taxonomía de los problemas

PROBLEMAS CON SOLUCIÓN

Existen muchos problemas que pueden resolverse mediante una computadora. Con frecuencia, queremos saber cuánto tiempo le toma a la computadora resolver ese problema. En otras palabras, ¿qué tan complejo es el programa?

La complejidad del programa puede medirse de varias maneras distintas, tales como el tiempo de ejecución, la memoria requerida y así por el estilo. Un método es el tiempo de ejecución: ¿cuánto le toma al programa ejecutarse?

Complejidad de los problemas con solución

Una manera de medir la complejidad de un problema con solución es encontrar el número de operaciones realizadas por la computadora cuando ésta ejecuta el programa. De esta manera, la complejidad es independiente de la velocidad de la computadora que ejecuta el programa. Esta medida de la complejidad puede depender del número de entradas. Por ejemplo, si un programa está procesando una lista (ordenando una lista), la complejidad depende del número de elementos en la lista.

Notación de orden O Con la velocidad de las computadoras actuales, no estamos tan preocupados con los números exactos como con los órdenes de magnitud generales. Por ejemplo, si el análisis de dos programas muestra que uno ejecuta 15 operaciones (o una serie de operaciones) mientras que el otro ejecuta 25, ambos son tan rápidos que usted no puede ver la diferencia. Por otra parte, si los números son 15 contra 1 500, usted debe preocuparse.

Esta simplificación de eficiencia se conoce como **notación de orden O**. Damos la idea de esta notación sin ahondar en su definición y cálculo formales. En esta notación, el número de operaciones (o una serie de operaciones relacionadas) se da como una función del número de entradas. La notación O (n) significa que un programa realiza n operaciones para n entradas; la notación O (n^2) significa que un programa realiza n^2 operaciones para n entradas.

EJEMPLO 3

Imagine que ha escrito tres programas distintos para resolver el mismo problema. El primero tiene una complejidad de O ($\log_{10} n$), el segundo O (n) y el tercero O (n^2). Suponiendo una entrada de un millón, ¿cuánto tiempo se llevaría ejecutar cada uno de estos programas en una computadora que realiza una instrucción en un milisegundo (un millón por segundo)?

SOLUCIÓN

A continuación se muestra el análisis:

```
Primer programa:   n=1 000 000 O(log10 n) → 6      Tiempo → 6 µs
Segundo programa:  n=1 000 000 O(n) → 1 000 000    Tiempo → 1 sec
Tercer programa:   n=1 000 000 O(n²) → 10^12       Tiempo → 277 hrs
```


Problemas polinomiales Si un programa tiene una complejidad de O (log n), O (n), O (n^2), O (n^3), O (n^4) o O (n^k) (siendo k una constante) se le llama *polinomial*. Con la velocidad de las computadoras actuales, usted puede obtener soluciones para los problemas polinomiales con un número razonable de entradas (por ejemplo, de 1 000 a un millón).

Problemas no polinomiales Si un programa tiene una complejidad mayor que un polinomial –por ejemplo, O (10^n) o O ($n!$)– éste puede resolverse si el número de entradas es muy pequeño (menor que 100). Si el número de entradas es grande, uno podría sentarse enfrente de la computadora durante meses para ver el resultado de un **problema no polinomial**. Pero ¿quién sabe? Al ritmo con el que la velocidad de las computadoras está aumentando, usted tal vez sea capaz de obtener un resultado para este tipo de problema en menos tiempo.

17.6 TÉRMINOS CLAVE

cinta
controlador
dígito hexadecimal
instrucción de decremento
instrucción de incremento
instrucción loop
macro
máquina de Turing
número de Gödel
problema con solución
problema no polinomial
problema polinomial
problema sin solución

17.7 RESUMEN

- La teoría de la computación puede ayudar a los científicos de la computación a responder preguntas intrínsecas.
- Tres instrucciones (incremento, decremento y ciclo) son necesarias para simular todos los otros tipos de instrucciones en un lenguaje de computadora. Por ejemplo, usted puede limpiar una variable, asignar un valor a una variable, copiar el valor de una variable a otra y sumar los valores de dos variables usando las tres instrucciones básicas.
- La máquina de Turing puede implementar instrucciones en nuestro Lenguaje simple.
- Una máquina de Turing tiene una cinta, un controlador y una cabeza de lectura/escritura.
- La cinta en una máquina de Turing mantiene una secuencia de caracteres de un conjunto de caracteres aceptable.
- La cabeza de lectura/escritura en una máquina de Turing en cualquier momento señala a un caracter. Después de leer y escribir, la cabeza puede moverse a la izquierda, a la derecha o permanecer inmóvil.
- El controlador en una máquina de Turing controla la cabeza de lectura/escritura. Es la contraparte teórica del CPU en las computadoras actuales.
- Un diagrama de transición es una representación pictórica de los estados del controlador.
- Una tabla de transición es una representación de una matriz de información respecto a los estados del controlador.
- Usted puede asignar un número de Gödel a cada programa en un lenguaje de computadora específico.
- No hay un programa que pueda predecir si un programa terminará o no.
- La notación de orden O se utiliza para denotar la eficiencia de un programa.
- Los problemas son ya sea con solución o sin solución. Los problemas con solución pueden clasificarse como polinomiales y no polinomiales.
- Los problemas no polinomiales por lo general toman más tiempo en resolverse que los problemas polinomiales si el número de entradas es muy grande.

17.8 PRÁCTICA

PREGUNTAS DE REPASO

1. ¿Por qué se necesita la teoría de la computación?
2. Mencione y proporcione las funciones de las tres instrucciones básicas que son la base de otras instrucciones en un lenguaje de computadora.
3. Muestre cómo la asignación del valor de una variable a otra (con la variable original conservando su valor) utiliza las tres instrucciones básicas.
4. ¿Cuál es la relación entre la máquina de Turing y nuestro Lenguaje simple?
5. ¿Cuáles son los componentes de la máquina de Turing y cuál es la función de cada componente?
6. ¿Cuál es una forma de delimitar los datos en una cinta de máquina de Turing?
7. Cuando una cabeza de lectura/escritura termina la lectura y escritura de un símbolo, ¿cuáles son sus siguientes opciones?
8. ¿Cómo se relaciona un diagrama de transición con un controlador de máquina de Turing?
9. ¿Cómo se relaciona un diagrama de transición con una tabla de transición? ¿Tienen la misma información? ¿Cuál tiene más información?
10. ¿Qué es un número de Gödel?
11. ¿Cómo utilizaría un número de Gödel para probar que el problema de paro no tiene solución?
12. ¿Cómo se puede indicar la eficiencia de un programa?
13. Compare y contraste la complejidad de un problema polinomial con solución y un problema no polinomial con solución.

PREGUNTAS DE OPCIÓN MÚLTIPLE

14. Un lenguaje de computadora simple puede diseñarse con sólo _____ instrucciones.
    a. una
    b. dos
    c. tres
    d. cuatro

15. La instrucción de _____ suma un 1 a la variable.
    a. incremento
    b. decremento
    c. ciclo
    d. complemento

16. La instrucción de _____ repite una o más acciones.
    a. incremento
    b. decremento
    c. ciclo
    d. complemento

17. La instrucción de _____ resta un 1 de la variable.
    a. incremento
    b. decremento
    c. ciclo
    d. complemento

18. Para limpiar una variable utilice las instrucciones de _____.
    a. incremento
    b. decremento
    c. ciclo
    d. b y c

19. Para asignar un número a una variable, utilice las instrucciones de _____.
    a. incremento
    b. decremento
    c. ciclo
    d. todas las anteriores

20. Para copiar el valor de una variable a otra variable y que la primera variable mantenga su valor, utilice las instrucciones de _____.
    a. incremento
    b. decremento
    c. ciclo
    d. todas las anteriores

21. La macro que cambia los ceros en unos y un entero positivo a 0 se llama macro _____.
    a. decr X
    b. Y X
    c. comp (X)
    d. switch (X)

22. Una máquina de Turing tiene los siguientes componentes: _____.
    a. cinta, memoria y cabeza de lectura/escritura
    b. disco, controlador y cabeza de lectura/escritura
    c. cinta, controlador y cabeza de lectura/escritura
    d. disco, memoria y controlador

23. En una máquina de Turing, la (el) _____ mantiene una secuencia de caracteres.
    a. disco
    b. cinta
    c. controlador
    d. cabeza de lectura/escritura

24. Después de leer un símbolo, la cabeza de lectura/escritura _____.
    a. se mueve a la izquierda
    b. se mueve a la derecha
    c. permanece en su sitio
    d. cualquiera de las opciones anteriores

25. El (la) _____ es la contraparte téorica del CPU.
    a. disco
    b. cinta
    c. controlador
    d. cabeza de lectura/escritura

26. El controlador tiene _____ estados.
    a. tres
    b. cuatro
    c. un número finito de
    d. un número infinito de

27. Un(a) _____ es una representación gráfica de los estados y sus relaciones entre sí.
    a. diagrama de transición
    b. diagrama de flujo
    c. tabla de transición
    d. máquina de Turing

28. Un(a) _____ muestra, entre otras cosas, el movimiento de la cabeza de lectura/escritura, el caracter leído y el caracter escrito.
    a. diagrama de transición
    b. diagrama de flujo
    c. tabla de transición
    d. máquina de Turing

29. El número de Gödel es un número _____ asignado a un programa en un lenguaje específico.
    a. binario
    b. entero
    c. con signo
    d. sin signo

30. El número de Gödel para `decr X` en decimal es _____.
    a. 367
    b. 175
    c. 174
    d. 191

31. El número de Gödel para `decr X` en hexadecimal es _____.
    a. B C
    b. C B
    c. B F
    d. A F

32. Usted utiliza _____ para denotar la complejidad de un programa.
    a. el número de Turing
    b. la notación de orden O
    c. factoriales
    d. el Lenguaje simple

33. Si la complejidad de $O(n^3)$ es 8, entonces el número de entradas es _____.
    a. uno
    b. dos
    c. tres
    d. cuatro

34. Si la complejidad de $O(n^1)$ es 24, entonces el número de entradas es _____.
    a. uno
    b. dos
    c. tres
    d. cuatro

35. La complejidad de `O(log*10 n)` y la computadora ejecuta un millón de instrucciones por segundo. ¿Cuánto tiempo se toma la ejecución del programa si el número de entradas es 10 000?
    a. un microsegundo
    b. dos microsegundos
    c. tres microsegundos
    d. cuatro microsegundos

EJERCICIOS

36. Simule la macro siguiente utilizando las instrucciones o macros previamente definidas en el Lenguaje Simple:

```
Z ⟵ X - Y
```

37. Simule la macro siguiente utilizando las instrucciones o macros previamente definidas en el Lenguaje simple:

```
if X < Y then A1 else A2
```

38. Simule la macro siguiente utilizando las instrucciones o macros previamente definidas en el Lenguaje simple:

```
if X > Y then A1 else A2
```

39. Simule la macro siguiente utilizando las instrucciones o macros previamente definidas en el Lenguaje simple:

```
while X > Y
{
  acciones
}
```

40. Simule la siguiente macro usando las instrucciones o macros previamente definidas en el Lenguaje simple:

```
while X < Y
{
  acciones
}
```

41. Simule la macro siguiente utilizando las instrucciones o macros previamente definidas en el Lenguaje simple:

```
while X == Y
 {
   acciones
 }
```

42. Muestre el diagrama de transición para la máquina de Turing que simula `X` ⟵ `0`.

43. Muestre el diagrama de transición para la máquina de Turing que simula `X` ⟵ `n`.

44. Muestre el diagrama de transición para la máquina de Turing que simula `Y` ⟵ `X`.

45. Muestre el diagrama de transición para la máquina de Turing que simula la macro `Z` ⟵ `X + Y`.

46. Muestre el diagrama de transición para la máquina de Turing que simula la macro `Z` ⟵ `X * Y`.

47. Muestre el diagrama de transición para la máquina de Turing que simula la macro comp `(X)`.

48. Muestre el diagrama de transición para la máquina de Turing que simula la macro if `X` then `A1 else A2`.

49. ¿Cuál es el número de Gödel para la macro `X1` ⟵ `0`?

50. ¿Cuál es el número de Gödel para la macro `X2` ⟵ `n`?

51. ¿Cuál es el número de Gödel para la macro `X3` ⟵ `X1 + X2`?

Código ASCII

El código norteamericano de estándares para intercambio de información (ASCII) es un código de siete bits que representa 128 caracteres como se muestra en la tabla 1.

Decimal	Hexadecimal	Binario	Caracter	Descripción
0	00	0000000	NUL	Nulo
1	01	0000001	SOH	Inicio de encabezado
2	02	0000010	STX	Inicio de texto
3	03	0000011	ETX	Fin de texto
4	04	0000100	EOT	Fin de transmisión
5	05	0000101	ENQ	Indagación
6	06	0000110	ACK	Reconocimiento
7	07	0000111	BEL	Campana
8	08	0001000	BS	Retroceso
9	09	0001001	HT	Tabulador horizontal
10	0A	0001010	LF	Avance de línea
11	0B	0001011	VT	Tabulador vertical
12	0C	0001100	FF	Salto de página
13	0D	0001101	CR	Retorno de carro
14	0E	0001110	SO	Desplazamiento hacia afuera
15	0F	0001111	SI	Desplazamiento hacia dentro
16	10	0010000	DLE	Escape de enlace de datos
17	11	0010001	DCI	Control de dispositivo 1
18	12	0010010	DC2	Control de dispositivo 2
19	13	0010011	DC3	Control de dispositivo 3
20	14	0010100	DC4	Control de dispositivo 4
21	15	0010101	NAK	Reconocimiento negativo
22	16	0010110	SYN	Inactividad síncrona
23	17	0010111	ETB	Fin del bloque de transmisión
24	18	0011000	CAN	Cancelar
25	19	0011001	EM	Fin del medio
26	1A	0011010	SUB	Sustituto
27	1B	0011011	ESC	Escape
28	1C	0011100	FS	Separador de archivos
29	1D	0011101	GS	Separador de grupos
30	1E	0011110	RS	Separador de registros
31	1F	0011111	US	Separador de unidades
32	20	0100000	SP	Espacio
33	21	0100001	!	Signo de exclamación
34	22	0100010	”	Comillas dobles
35	23	0100011	#	Signo de números
36	24	0100100	$	Signo monetario
37	25	0100101	%	Signo de porcentaje
38	26	0100110	&	Ampersand
39	27	0100111	’	Apóstrofe
40	28	0101000	(	Paréntesis abierto
41	29	0101001	)	Paréntesis cerrado
42	2A	0101010	*	Asterisco
43	2B	0101011	+	Signo más
44	2C	0101100	,	Coma
45	2D	0101101	-	Guión
46	2E	0101110	.	Punto
47	2F	0101111	/	Diagonal

Tabla A.1 Tabla ASCII

Decimal	Hexadecimal	Binario	Caracter	Descripción
48	30	0110000	0	
49	31	0110001	1	
50	32	0110010	2	
51	33	0110011	3	
52	34	0110100	4	
53	35	0110101	5	
54	36	0110110	6	
55	37	0110111	7	
56	38	0111000	8	
57	39	0111001	9	
58	3A	0111010	:	Dos puntos
59	3B	0111011	;	Punto y coma
60	3C	0111100	<	Signo menor que
61	3D	0111101	=	Signo igual
62	3E	0111110	>	Signo mayor que
63	3F	0111111	?	Signo de interrogación
64	40	1000000	@	Arroba
65	41	1000001	A	
66	42	1000010	B	
67	43	1000011	C	
68	44	1000100	D	
69	45	1000101	E	
70	46	1000110	F	
71	47	1000111	G	
72	48	1001000	H	
73	49	1001001	I	
74	4A	1001010	J	
75	4B	1001011	K	
76	4C	1001100	L	
77	4D	1001101	M	
78	4E	1001110	N	
79	4F	1001111	O	
80	50	1010000	P	
81	51	1010001	Q	
82	52	1010010	R	
83	53	1010011	S	
84	54	1010100	T	
85	55	1010101	U	
86	56	1010110	V	
87	57	1010111	W	
88	58	1011000	X	
89	59	1011001	Y	
90	5A	1011010	Z	
91	5B	1011011	[	Corchete de apertura
92	5C	1011100	\	Diagonal invertida
93	5D	1011101	]	Corchete de cierre
94	5E	1011110	^	Circunflejo
95	5F	1011111	_	Guión bajo

Tabla A.1 Tabla ASCII (continuación)

Decimal	Hexadecimal	Binario	Caracter	Descripción
96	60	0110000	`	Acento grave
97	61	0110001	a	
98	62	0110010	b	
99	63	0110011	c	
100	64	0110100	d	
101	65	0110101	e	
102	66	0110110	f	
103	67	0110111	g	
104	68	0111000	h	
105	69	0111001	i	
106	6A	0111010	j	
107	6B	0111011	k	
108	6C	0111100	l	
109	6D	0111101	m	
110	6E	0111110	n	
111	6F	0111111	o	
112	70	1000000	p	
113	71	1000001	q	
114	72	1000010	r	
115	73	1000011	s	
116	74	1000100	t	
117	75	1000101	u	
118	76	1000110	v	
119	77	1000111	w	
120	78	1001000	x	
121	79	1001001	y	
122	7A	1001010	z	
123	7B	1001011	{	Llave de apertura
124	7C	1001100	|	Barra
125	7D	1001101	}	Llave de cierre
126	7E	1001110	~	Tilde
127	7F	1001111	DEL	Eliminar

Tabla A.1 Tabla ASCII (continuación)

B Unicode

Unicode es un código de 16 bits que puede representar hasta 65 536 símbolos. Utilizando la notación hexadecimal, el código puede variar de 0000 a FFFF. Observe que se requieren decenas, si no es que cientos, de páginas para mostrar todos los símbolos individuales (caracteres). Aquí mostramos sólo algunos intervalos.

ALFABETOS

Los códigos de 0000 a 1FFF definen diferentes alfabetos. Algunos de ellos se muestran en la tabla B.1. Observe lo siguiente respecto a esta tabla:

1. Los códigos 0000 a 007F (Latín básico) son exactamente los mismos que se definieron para el código ASCII.
2. Los códigos 0080 a 00FF (complemento de Latín 1) son los mismos que los caracteres Latín 1 definidos por la ISO. El sistema operativo Windows utiliza la variación de estos símbolos. Los caracteres de Latín básico se complementan con caracteres de acento, diéresis, el signo de interrogación de cierre y así por el estilo.

Intervalo	Descripción
0000–007F	Latín básico
0080–00FF	Complemento de Latín 1
0100–017F	Latín extendido A
0180–024F	Latín extendido B
0250–02AF	Extensión del alfabeto fonético internacional (IPA)
02B0–02FF	Espaciado de letras modificadoras
0300–036F	Combinación de marcas diacríticas
0370–03FF	Griego
0400–04FF	Cirílico
0530–058F	Armenio
0590–05FF	Hebreo
0600–06FF	Árabe
0700–074F	Sirio
0780–07BF	Thaana
0900–097F	Deviagari
0980–09FF	Bengalí
0A00–0A7F	Gumurkhi
0A80–0AFF	Gujaratí
0B00–0B7F	Oriya
0B80–0BFF	Tamul
0C00–0C7F	Teluga
0C80–0CFF	Kannada
0D00–0D7F	Malayalam
0D80–0DFF	Cingalés
0E00–0E7F	Tailandés
0E80–0EFF	Lao
0F00–0FFF	Tibetano
1000–109F	Myanmar
10A0–10FF	Georgiano
1100–11FF	Hangul Jamo
1200–137F	Etíope
13A0–13FF	Cheroquí
1400–167F	Silábico unificado de los aborígenes canadienses
1680–169F	Ogham
16a0–16FF	Rúnico
1780–17FF	Jemer
1800–18AF	Mongol
1E00–1EFF	Latín extendido adicional
1F00–1FFF	Griego extendido

Tabla B.1 Alfabetos (0000-1FFF)

SÍMBOLOS Y MARCAS DE PUNTUACIÓN

Los códigos 2000 a 2FFF definen los símbolos y las marcas de puntuación. Algunos de ellos se muestran en la tabla B.2.

Intervalo	Descripción
2000–2067	Puntuación general
2070–209F	Subíndices y superíndices
20A0–20CF	Símbolos de moneda
20D0–20FF	Combinación de marcas para símbolos
2100–214F	Símbolos tipo carta
2150–218F	Formatos de números
2190–21FF	Flechas
2200–22FF	Operadores matemáticos
2300–23FF	Miscelánea técnica
2400–243F	Imágenes de control
2440–245F	Reconocimiento óptico de caracteres
2460–24FF	Alfanumérico cerrado
2500–257F	Bordes de cuadros
2580–259F	Elementos de bloque
25A0–25FF	Formas geométricas
2600–26FF	Símbolos misceláneos
2700–27BF	Símbolos decorativos
2800–28FF	Patrones Braile
2E80–2EFF	Complemento radical CJK
2F00–2FDF	Kanji radical
2FF0–2FFF	Caracteres de descripción ideográfica

Tabla B.2 Símbolos y puntuación (2000 a 2FFF)

AUXILIARES CJK

Los códigos 3000 a 33FF definen los auxiliares chinos, japoneses y coreanos (CJK: *Chinese, Japanese and Korean*), algunos de los cuales se muestran en la tabla B.3.

Intervalo	Descripción
3000–303F	Símbolos y puntuación de CJK
3040–309F	Hiragana
30A0–30FF	Katakana
3100–312F	Bopomofo
3130–318F	Jamo con compatibilidad Hangul
3190–319F	Kanbun
31A0–31BF	Bopomofo extendido
3200–32FF	Letras y meses en CJK cerrados
3300–33FF	Compatibilidad con CJK

Tabla B.3 Auxiliares CJK (3000 a 33FF)

IDEOGRAMAS CJK UNIFICADOS

Los códigos 4000 a 9FFF definen los ideogramas CJK unificados.

SUSTITUTOS Los códigos D800 a DFFF definen los sustitutos.

USO PRIVADO Los códigos E000 a F8FF son para uso privado.

CARACTERES Y SÍMBOLOS MISCELÁNEOS Los códigos F900 a FFFF definen caracteres y símbolos misceláneos.

C Diagramas de flujo

Una herramienta muy efectiva para mostrar el flujo lógico de un programa es el diagrama de flujo. En un entorno de programación, puede utilizarse para diseñar un programa completo o sólo parte de un programa.

El propósito principal de un diagrama de flujo es mostrar el diseño de un algoritmo. Al mismo tiempo, libera a los programadores de la sintaxis y los detalles de un lenguaje de programación mientras que les permite concentrarse en los detalles del problema a resolver.

Un diagrama de flujo proporciona una representación pictórica de un algoritmo. Esto es en contraposición a otra herramienta de diseño de programación, el pseudocódigo (ver apéndice D), que proporciona una solución de diseño textual. Ambas herramientas tienen sus ventajas, pero un diagrama de flujo tiene la capacidad pictórica que a otras herramientas les falta.

C.1 SÍMBOLOS AUXILIARES

Un diagrama de flujo es una combinación de símbolos. Algunos símbolos se usan para mejorar la legibilidad o funcionalidad del diagrama de flujo. No se utilizan directamente para mostrar instrucciones o comandos. Muestran los puntos de inicio y fin, el orden y la secuencia de acciones y cómo una parte de un diagrama de flujo se conecta con otra. Estos símbolos auxiliares se muestran en la figura C.1.

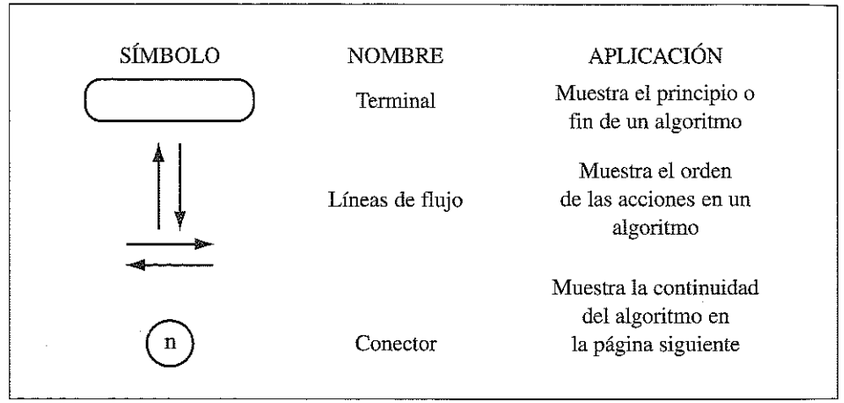


Figura C.1 Símbolos auxiliares en el diagrama de flujo

INICIO Y FIN

Un óvalo se utiliza para mostrar el inicio y fin de un algoritmo. Cuando lo utilice para mostrar el principio de un algoritmo, escriba la palabra START (iniciar) en el óvalo; cuando lo utilice para indicar el fin de un algoritmo, escriba la palabra STOP (detener) en el óvalo.

Una de las primeras reglas de la programación estructurada es que cada algoritmo debe tener sólo un punto de entrada y uno de salida. Esto significa que un diagrama de flujo bien estructurado debe tener uno y sólo un inicio START, y uno y sólo un fin STOP. Los óvalos deben estar alineados para mostrar claramente el flujo de la acción en un algoritmo. Por ejemplo, en la figura C.2 se muestra un diagrama de flujo para un programa que no hace nada. Este programa inicia y termina sin hacer nada.

Un óvalo también puede usarse para indicar el inicio y el fin de un módulo. Cuando lo utilice al principio escriba dentro de él el nombre del módulo (en lugar de la palabra START), cuando al final escriba la palabra RETURN (regreso), en vez de STOP.

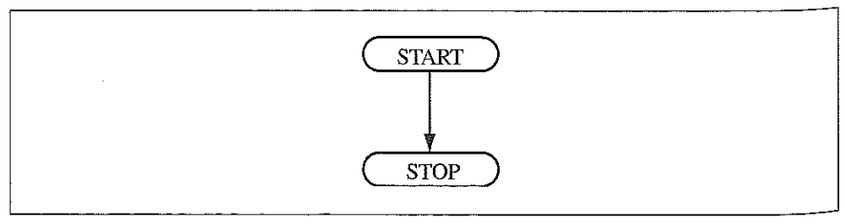


Figura C.2 Símbolos de inicio y fin

LÍNEAS DE FLUJO

Las líneas de flujo se utilizan para mostrar el orden o la secuencia de las acciones en un programa. Estas líneas conectan los símbolos. Por lo general, un símbolo tiene algunas líneas de entrada y algunas líneas de salida. El óvalo START tiene sólo una línea de salida. El óvalo de STOP tiene sólo una línea de entrada. Acabamos de mostrar el uso de las líneas de flujo en la figura C.2. Mostraremos otros flujos en los ejemplos que siguen.

CONECTORES

Usted utiliza sólo un símbolo, un círculo con un número dentro de él, para mostrar la conectividad. Se utiliza cuando se llega al final de la página y el diagrama de flujo aún no termina. En la parte inferior de la página utilice un conector para mostrar que el flujo lógico continúa en la parte superior de la página siguiente. El número en el conector puede ser un número serial simple o puede ser una combinación de una página y un símbolo en la forma *página.número*. La figura C.3 muestra un conector fuera de página.

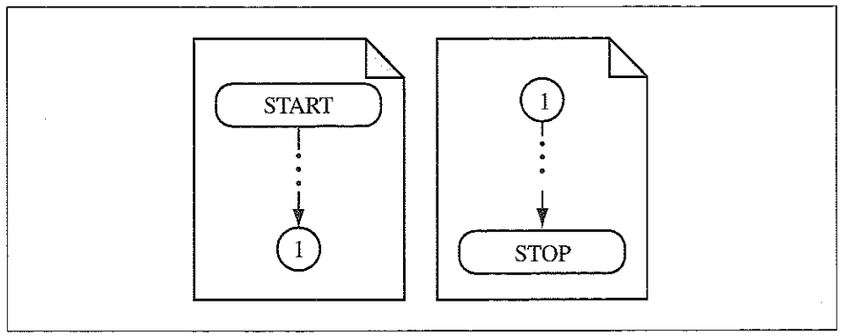


Figura C.3 Conectores

C.2 SÍMBOLOS PRINCIPALES

Los símbolos principales se utilizan para mostrar las instrucciones o acciones necesarias para resolver el problema presentado en el algoritmo. Con estos símbolos, es posible representar los cinco constructores de programación estructurada: secuencia, decisión, ciclo `while`, ciclo `for` y ciclo `do-while`.

INSTRUCCIONES EN SECUENCIA

Las instrucciones en secuencia simplemente representan una serie de acciones que deben continuar en un orden lineal. Aun cuando las acciones representadas en el símbolo de secuencia pueden ser muy complejas, por ejemplo una operación de entrada o una operación de salida, el flujo lógico debe introducir el símbolo en la parte superior y fluir hacia fuera en la parte inferior. Los flujos de secuencia no permiten que se tome ninguna decisión o haya cambios de flujo dentro del símbolo.

Existen cinco símbolos de secuencia: instrucción nula, de asignación, de entrada/salida, llamada de módulo e instrucción compuesta. Los últimos cuatro se muestran en la figura C.4.

Instrucción nula

Vale la pena resaltar que *do nothing* (hacer nada) es una instrucción válida. Comúnmente se le conoce como instrucción nula. La instrucción nula se considera una instrucción de secuencia debido a que no puede cambiar la dirección del flujo de un programa. No hay un símbolo para una instrucción nula. Es simplemente una línea de flujo. La figura C.2 es un ejemplo de una instrucción nula.

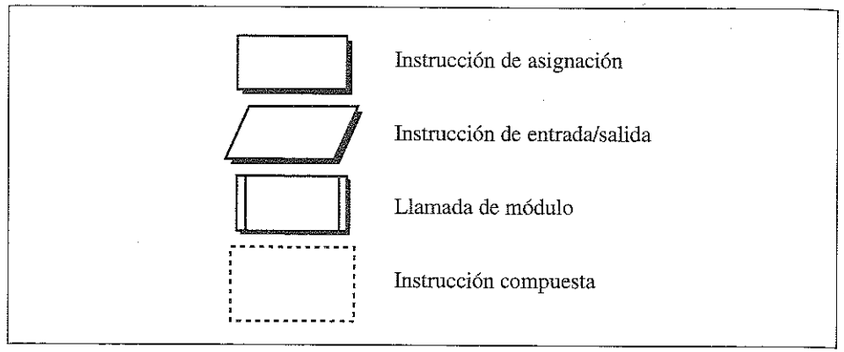


Figura C.4 Símbolos de secuencia

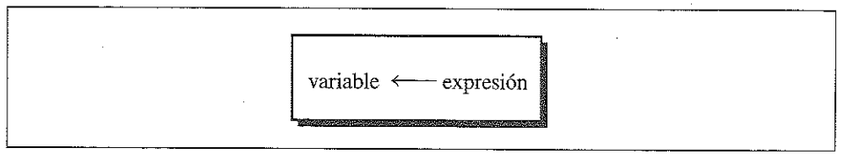


Figura C.5 Instrucción de asignación

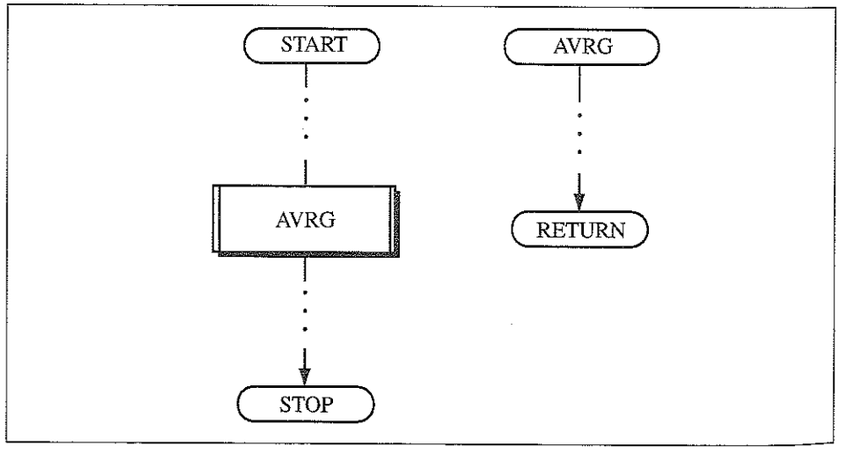


Figura C.6 Instrucción de llamada de módulo

Instrucción de asignación

La instrucción de asignación se muestra utilizando un rectángulo. Dentro del rectángulo, el operador de asignación se muestra como una flecha que apunta a la izquierda. Al lado derecho de la flecha hay una expresión cuyo valor debe almacenarse en la variable del lado izquierdo. La figura C.5 muestra una instrucción de asignación.

Instrucción de entrada/salida

Un paralelogramo se utiliza para mostrar cualquier entrada o salida, por ejemplo, leer de un teclado o escribir en la consola del sistema.

Instrucción de llamada de módulo

El símbolo para llamar a un módulo es un rectángulo con dos barras verticales dentro. El diagrama de flujo para el módulo llamado debe estar en alguna otra parte. En otras palabras, cada vez que usted vea una instrucción de llamada de módulo, busque otro diagrama de flujo con el nombre del módulo (figura C.6).

Instrucción compuesta

Aun cuando no hay un símbolo real que muestre una instrucción compuesta, encapsulamos todas las instrucciones que hacen una instrucción compuesta en un rectángulo de línea discontinua.

INSTRUCCIONES DE SELECCIÓN

A diferencia de la instrucción de secuencia, las instrucciones de selección pueden provocar que el flujo del programa cambie. Permiten la ejecución de instrucciones seleccionadas y el salto de otras instrucciones. En la programación estructurada hay dos instrucciones de selección: bidireccional y multidireccional.

Selección bidireccional

El símbolo bidireccional es el diamante. Cuando se utiliza para representar una instrucción `if-else`, la lógica verdadera se muestra en la parte derecha del flujo lógico, y la condición falsa, si la hay, se coloca en la parte izquierda del mismo. Con `if-else`, siempre debe haber dos flujos lógicos, aunque a menudo uno de ellos es nulo. (Recuerde que la instrucción nula se representa mediante una línea de flujo; no hay un símbolo para nulo.) Finalmente, la instrucción termina con un conector donde los flujos verdadero y falso se unen. En este caso, el conector no tiene nada dentro de él.

Aunque con frecuencia verá decisiones esbozadas con el inicio del flujo en la parte inferior del diamante, este estilo no es bueno. Aunque uno de los flujos sea nulo, éste debe fluir de la izquierda o de la derecha del diamante de todos modos.

La figura C.7 muestra el uso del símbolo de decisión en la instrucción `if-else`. Como señalamos, siempre hay dos ramas. En una de ellas se permite tener una y sólo una instrucción. Desde luego, la instrucción en cada rama puede ser una instrucción nula o compuesta. Pero sólo se permite una instrucción en cada rama; ni más, ni menos. También recuerde que toda la figura es una sola instrucción, no dos o tres; es una sola instrucción `if-else`.

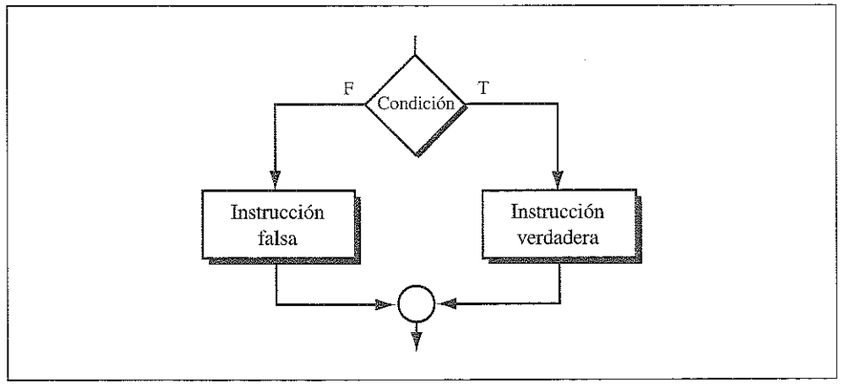


Figura C.7 Selección bidireccional

Selección multidireccional

La segunda aplicación del símbolo de selección utilizado en la programación estructurada es la selección multibidireccional (figura C.8). Como puede ver, se pueden tener tantas ramas como sea necesario. En cada rama, se permite tener una y sólo una instrucción. Desde luego, la instrucción en cada rama puede ser una instrucción nula o compuesta. Pero recuerde que sólo se permite una instrucción en cada rama; ni más ni menos. También recuerde que toda la figura es una sola instrucción, no dos o tres.

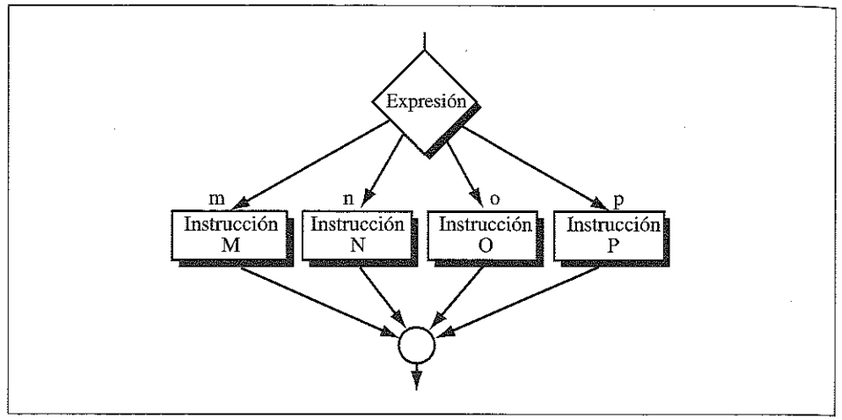


Figura C.8 Selección multidireccional

INSTRUCCIONES DE CICLO

Hay tres instrucciones de ciclo: `for`, `while` y `do-while`.

Instrucción `for`

La instrucción `for` es un ciclo controlado por contador. En realidad es una instrucción compleja que tiene tres partes, cualquiera de las cuales puede ser nula: (1) la inicialización del ciclo, la cual normalmente establece el contador del ciclo; (2) la prueba de límite y (3) las instrucciones de acción de fin del ciclo, las cuales por lo general incrementan un contador. Dado que la instrucción `for` es un ciclo de preprueba, es posible que el ciclo no se ejecute. Si la condición que termina es verdadera al inicio, el cuerpo de la instrucción `for` se salta.

Como sucede en todos los constructores de programación estructurada, el cuerpo del ciclo puede contener una y sólo una instrucción. Como sucede con los otros constructores, esta instrucción puede ser nula o compuesta. La figura C.9 muestra el ciclo `for`.

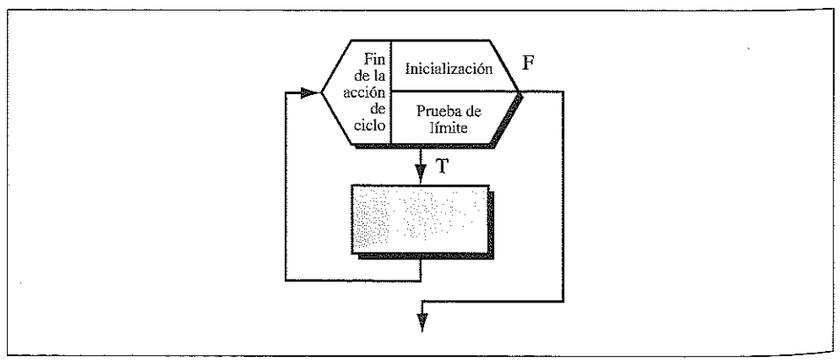


Figura C.9 Ciclo for

Instrucción while

El segundo constructor de ciclo es la instrucción `while`. La principal diferencia entre los ciclos `for` y `while` es que el ciclo `while` no es un ciclo de conteo. Ambos son ciclos de preprueba; esto significa que, como sucede con `for`, el cuerpo del ciclo `while` puede no ejecutarse nunca.

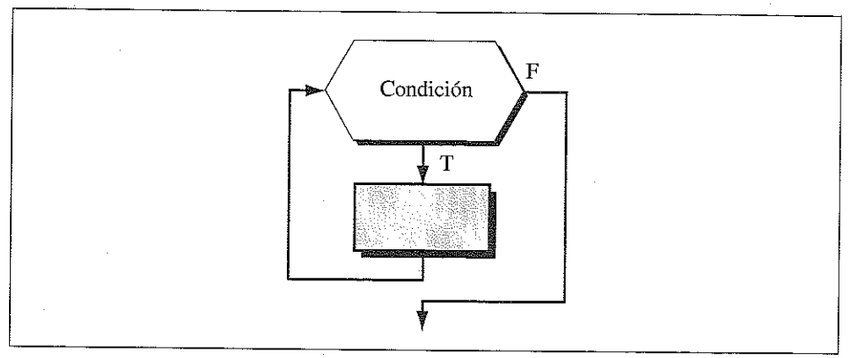


Figura C.10 Ciclo `while`

Usted utiliza el mismo símbolo básico para el ciclo `while`, pero debido a que sólo hay una prueba de límite, las divisiones internas no son necesarias. La figura C.10 muestra el formato básico de la instrucción `while`.

Instrucción `do-while`

La tercera aplicación del símbolo de ciclo es la instrucción `do-while` (figura C.11). Debido a las diferencias inherentes entre los ciclos `for` y `while` y el ciclo `do-while`, éste se presenta de manera diferente en un diagrama de flujo. Hay dos diferencias importantes entre `while` y `do-while`:

1. Un ciclo `while` es un ciclo de preprueba. El ciclo `do-while` es un ciclo de posprueba.
2. El cuerpo de un ciclo `while` puede no ejecutarse nunca. El cuerpo de un ciclo `do-while` se ejecuta al menos una vez.

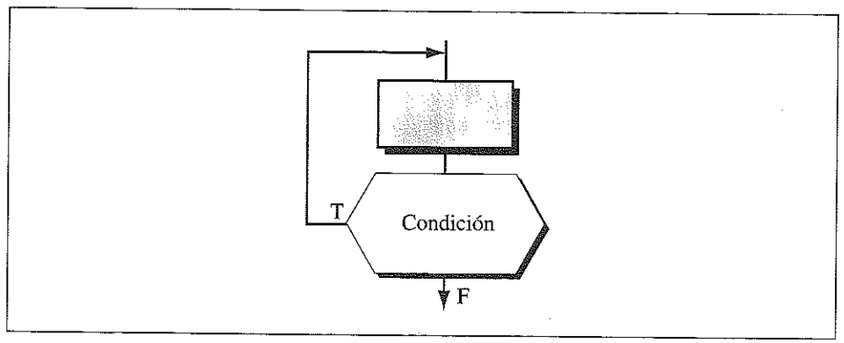


Figura C.11 Ciclo `do-while`

Pseudocódigo

Una de las herramientas más comunes para definir algoritmos es el pseudocódigo. El pseudocódigo es una representación tipo idioma inglés del código requerido para un algoritmo. Es parte inglés y parte código estructurado. La parte de inglés proporciona una sintaxis relajada fácil de leer. La parte del código consiste de una versión ampliada de los constructores de algoritmos básicos: secuencia, selección e iteración. El algoritmo D.1 muestra un ejemplo de pseudocódigo.

Algoritmo D.1

```
Algoritmo: Encontrar Menor
Propósito: Este algoritmo encuentra el número más pequeño entre una
           lista de números.
Pre: Lista de números
Post: Ninguno
Devuelve: El menor
1. Establece el menor como el primer número
2. loop (no al final de la lista)
   2.1 if (siguiente número < menor)
         2.1.1 establece el menor como el número siguiente
   2.2 end if
3. end loop
4. return menor
End Encontrar Menor
```

D.1 COMPONENTES

Un algoritmo escrito en pseudocódigo puede decomponerse en varios elementos y constructores.

ENCABEZADO DE ALGORITMO

Cada algoritmo comienza con un encabezado que lo nombra. Por ejemplo, en el algoritmo D.1, el encabezado comienza con la palabra *Algoritmo*, la cual nombra al algoritmo como "Encontrar Menor".

PROPÓSITO, CONDICIONES Y DEVOLUCIÓN

Después del encabezado, usted por lo general menciona el propósito y las pre y postcondiciones, y devueltos por el algoritmo.

Propósito

El propósito es una breve frase sobre lo que hace el algoritmo. Necesita describir sólo el procesamiento del algoritmo general. No debe intentar describir todo el procesamiento. En el algoritmo D.1, el propósito comienza con la palabra *Propósito* y continúa con el objetivo del algoritmo.

Precondición

La precondición lista cualquier requisito precursor. Por ejemplo, en el algoritmo D.1, se requiere que la lista esté disponible para el algoritmo.

Postcondición

La postcondición identifica cualquier efecto creado por el algoritmo. Por ejemplo, tal vez el algoritmo especifica la impresión de los datos.

Devolución

Creemos que todo algoritmo debe mostrar lo que devuelve el mismo. Si no hay nada que devolver, aconsejamos que se especifique una devolución nula. En el algoritmo D.1, se devuelve el valor más pequeño encontrado.

NÚMEROS DE INSTRUCCIÓN

Las instrucciones se numeran como se muestra en el algoritmo D.1 (1, 2, 3 . . .). Las instrucciones dependientes se numeran de manera que muestren sus dependencias (1.1, 2.4 . . .).

CONSTRUCTORES DE INSTRUCCIÓN

Cuando Niklaus Wirth propuso por primera vez el modelo de programación estructurada, declaró que cualquier algoritmo puede escribirse con sólo tres constructores de programación: secuencia, selección y ciclo. Nuestro pseudocódigo contiene sólo estos tres constructores bá-

sicos. La implementación de estos tres constructores se basa en la riqueza del lenguaje de implementación. Por ejemplo, el ciclo puede implementarse como una instrucción `while`, `do-while` o `for` en el lenguaje C.

SECUENCIA

Una secuencia es una serie de instrucciones que no alteran la ruta de ejecución dentro de un algoritmo. Aunque es obvio que instrucciones como assign (asignar) y add (sumar) son instrucciones de secuencia, no es tan obvio que una llamada a otros algoritmos también se considera una instrucción de secuencia. La razón radica en el concepto de programación estructurada de que cada algoritmo tiene sólo una entrada y una salida. Además, cuando un algoritmo se completa, regresa a la instrucción inmediatamente después de la llamada que lo invocó. Por consiguiente, usted puede considerar de manera apropiada la llamada del algoritmo como una instrucción de secuencia. El algoritmo D.2 muestra una secuencia.

Algoritmo D.2

```
. . .
7. establece x como el primer número
8. establece y como el segundo número
9. multiplica x por y y almacena el resultado en z
. . .
```

SELECCIÓN

Las instrucciones de selección evalúan una o más alternativas. Si son verdaderas, se toma una ruta. Si son falsas se toma una ruta distinta. La instrucción de selección típica es la selección bidireccional (`if-else`). Aun cuando la mayor parte de los lenguajes proporciona selecciones multidireccionales, no proporcionamos ninguna en el pseudocódigo. Las alternativas de la selección se identifican mediante sangrías (sangrando), como se muestra en el algoritmo D.3.

Algoritmo D.3

```
. . .
5. if (x < y)
       5.1 incrementa x
       5.2 imprime x
6. else
       6.1 disminuye y
       6.2 imprime y
7. end if
. . .
```

CICLO

Un ciclo itera un bloque de código. El ciclo en nuestro pseudocódigo se parece más al ciclo `while`. Es un ciclo de preprueba; es decir, la condición se evalúa antes de ejecutar el cuerpo del ciclo. Si la condición es verdadera, el cuerpo se ejecuta. Si la condición es falsa, el ciclo termina. El algoritmo D.4 muestra un ejemplo de un ciclo.

Algoritmo D.4

```
. . .
3. loop (no al final del Archivo 1)
   3.1 lee la línea siguiente
   3.2 elimina los espacios principales
   3.3 copia la línea al Archivo 2
4. end loop
. . .
```

Diagramas de estructura

El diagrama de estructura es la principal herramienta de diseño para un programa. Como herramienta de diseño, ésta se crea antes de comenzar a escribir un programa.

E.1 SÍMBOLOS DEL DIAGRAMA DE ESTRUCTURA

La figura E.1 muestra los diversos símbolos utilizados en un diagrama de estructura.

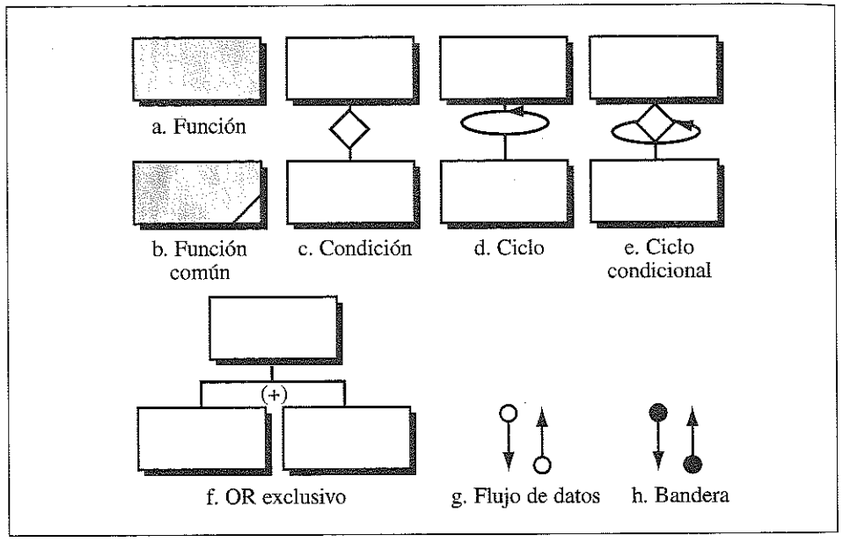


Figura E.1 Símbolos del diagrama de estructura

SÍMBOLO DE FUNCIÓN

Cada rectángulo en un diagrama de estructura representa una función que usted escribe. El nombre en el rectángulo es el nombre que usted da a la función (figura E.2).

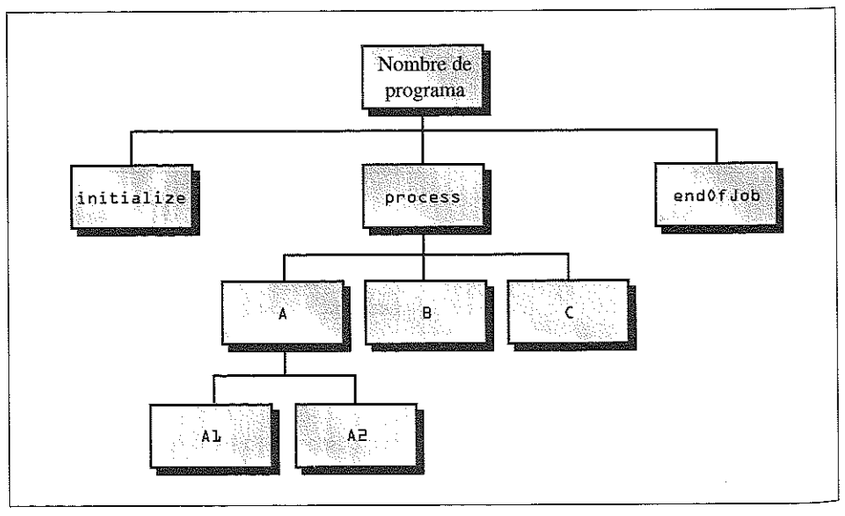


Figura E.2 Ejemplo de un diagrama de estructura

SELECCIÓN EN EL DIAGRAMA DE ESTRUCTURA

La figura E.3 muestra dos símbolos para una función que es llamada por una instrucción de selección: la condición y el OR exclusivo.

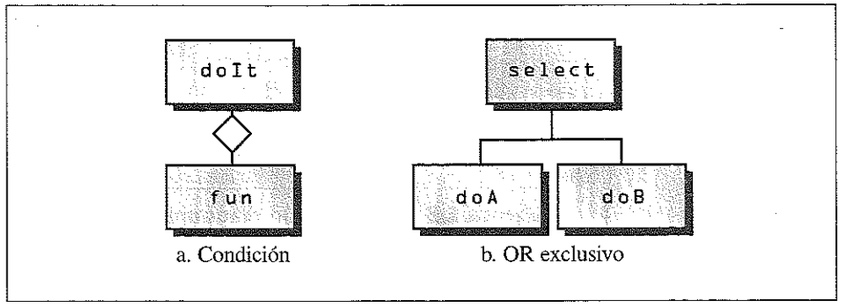


Figura E.3 Selección en una tabla de estructura

En la figura E.3a, la función `doIt` contiene una llamada condicional a una subfunción, `fun`. Si la condición es verdadera, se llama a `fun`. Si no es verdadera, se omite `fun`. Esta situación se representa en un diagrama de estructura como un diamante en la línea vertical entre los dos bloques de función.

La figura E.3b representa la selección entre dos funciones diferentes. En este ejemplo, la función `select` elige entre `doA` y `doB`. Una y sólo una de ellas se llamará cada vez que se ejecute la instrucción de selección. Esto se conoce como OR exclusivo; una de las dos alternativas se ejecuta excluyendo la otra. El OR exclusivo se representa mediante un signo más entre los procesos.

Ahora considere el diseño encontrado para una serie de funciones que puede llamarse exclusivamente. Esto ocurre cuando una selección multidireccional contiene llamadas a varias funciones diferentes. La figura E.4 contiene un ejemplo de una instrucción de selección que llama a diferentes funciones con base en el color.

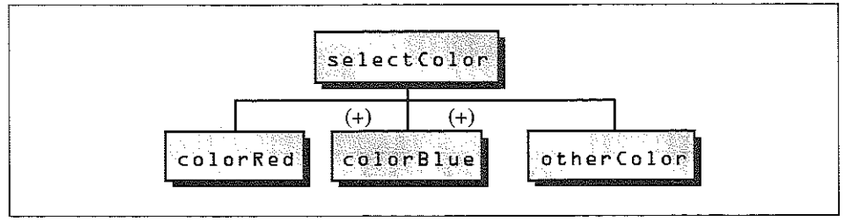


Figura E.4 Ejemplo de una selección

CICLOS EN EL DIAGRAMA DE ESTRUCTURA

Veamos cómo se representan los ciclos en un diagrama de estructura. Los símbolos son muy simples. Los ciclos pueden ir en círculos, así que el símbolo usado es un círculo. Los programadores usan dos símbolos de ciclo básicos. El primero es un ciclo simple, como se muestra en la figura E.5a. El otro es el ciclo condicional, exhibido en la figura E.5b.

Cuando la función se llama de manera incondicional, como en un ciclo `while`, el círculo fluye alrededor de la línea que está sobre la función llamada. Por otra parte, si la llamada es condicional, como en una función llamada en una instrucción `if-else` dentro de un ciclo, entonces el círculo incluye un diamante de decisión en la línea.

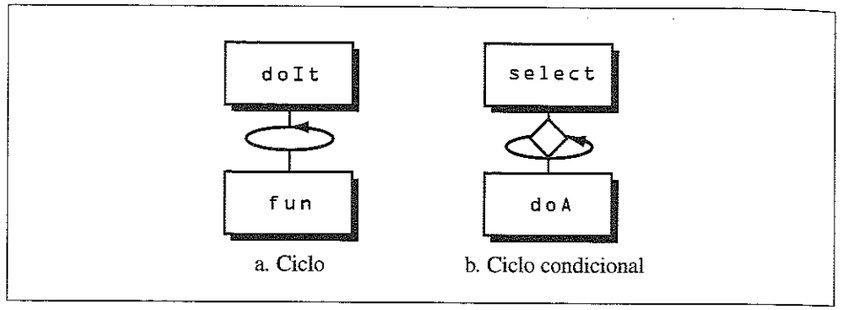


Figura E.5 Ciclos en un diagrama de estructura

La figura E.6 muestra la estructura básica para una función llamada `process`. El círculo está *debajo* de la función que controla el ciclo. En este ejemplo la instrucción de ciclo está contenida en `process` y llama a tres funciones: `A`, `B` y `C`. La naturaleza exacta del ciclo no puede determinarse a partir del diagrama de estructura. Puede ser cualquiera de los tres constructores de ciclo básicos.

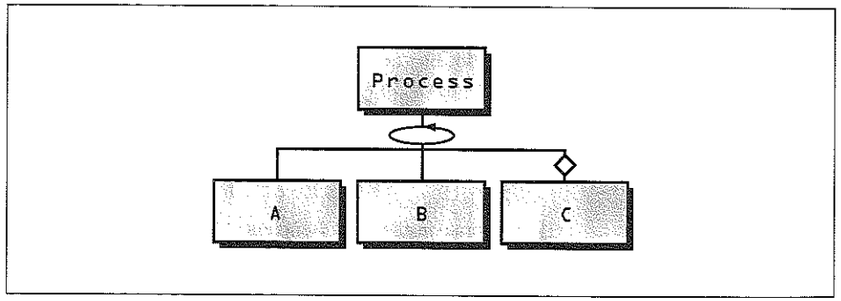


Figura E.6 Ejemplo de un ciclo

FLUJO DE DATOS

No es necesario mostrar los flujos de datos y las banderas, aunque puede ser útil en ciertas circunstancias. Si éstos se muestran, las entradas se colocan a la izquierda de la línea vertical y las salidas, a la derecha. Cuando se incluyen, el nombre de los datos o la bandera también deben indicarse.

E.2 LECTURA DE DIAGRAMAS DE ESTRUCTURA

Los diagramas de estructura se leen de *arriba hacia abajo* y de *izquierda a derecha.* Respecto a la figura E.2, esta regla dice que el Nombre de programa (`main`) consiste de tres subfunciones: `initialize`, `process` y `endOfJob`. De acuerdo con la regla de izquierda a derecha, la primera llamada en el programa es `initialize`. Después de que se completa `initialize`, el programa llama a `process`. Cuando `process` está completo, el programa llama a `endOfJob`. En otras palabras, las funciones en el mismo nivel de un diagrama de estructura se llaman en orden de izquierda a derecha.

El concepto de arriba hacia abajo se demuestra mediante `process`. Cuando se llama a `process`, ésta llama a su vez a `A`, `B` y `C`. Sin embargo, la función `B` no comienza su ejecución hasta que `A` haya terminado. Mientras `A` está en ejecución, ésta llama a su vez a `A1` y `A2`. En otras palabras, todas las funciones en una línea de `process` a `A2` deben llamarse antes de que la función `B` pueda comenzar.

A menudo, un programa contendrá varias llamadas a una función común. Estas llamadas por lo general se esparcen en todo el programa. El diagrama de estructura mostrará la llamada siempre que ésta ocurra lógicamente en el programa. Para identificar estructuras comunes, la esquina inferior derecha del rectángulo estará sombreada. Si la función común es compleja y contiene subfunciones, estas subfunciones necesitan mostrarse sólo una vez. Debe mostrarse una indicación de que las referencias incompletas contienen estructura adicional. Por lo general esto se hace con una línea debajo de la función rectángulo y un símbolo de corte (~). Este concepto se muestra en la figura E.7, la cual utiliza una función común, `average`, en dos lugares diferentes del programa. No obstante, observe que nunca muestra gráficamente una función conectada a dos funciones que hacen una llamada.

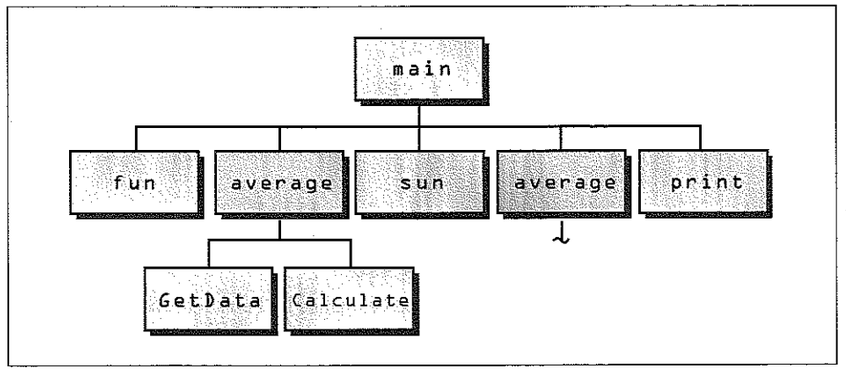


Figura E.7 Varias llamadas a la misma función

E.3 REGLAS DE LOS DIAGRAMAS DE ESTRUCTURA

A continuación resumimos las reglas analizadas en esta sección:

- Cada rectángulo en un diagrama de estructura representa una función escrita por el programador.
- El nombre en el rectángulo es el nombre que se usará en la codificación de la función.
- El diagrama de estructura contiene sólo un flujo de función. No se indica ningún código.
- Las funciones comunes se indican mediante sombreado en la esquina inferior derecha del rectángulo de la función.
- Los flujos de datos y las banderas son opcionales. Cuando se utilizan, debe asignárseles un nombre.
- Los flujos de entrada y las banderas se muestran a la izquierda de la línea vertical; los flujos de salida y las banderas se muestran a la derecha.

Transformada coseno discreta

En este apéndice, damos las bases matemáticas para las transformaciones de coseno discreta y de coseno discreta inversa.

F.1 TRANSFORMADA COSENO DISCRETA

En esta transformación, cada bloque de 64 pixeles pasa por una transformación llamada transformada coseno discreta (DCT: *discrete cosine transform*). La transformación cambia los 64 valores de modo que la relación relativa entre los pixeles se mantiene pero las redundancias se revelan. La fórmula es la siguiente: $P(x,y)$ define un valor particular en el bloque de imagen; $T(m,n)$ define un valor en el bloque transformado.

$$T(m,n) = 0.25c(m)c(n)\sum_{x=0}^{7}\sum_{y=0}^{7}P(x,y) \quad \cos\left(\frac{(2x+1)m\pi}{16}\right) \quad \cos\left(\frac{(2y+1)n\pi}{16}\right)$$

donde

$$c(i) = \begin{cases} \frac{1}{\sqrt{2}} \text{ si } i = 0 \\ 1 \text{ de lo contrario} \end{cases}$$

F.2 TRANSFORMACIÓN INVERSA

La transformación inversa se utiliza para crear la tabla $P(x,y)$ a partir de la tabla $T(m,n)$.

$$P(x,y) = 0.25c(x)c(y)\sum_{m=0}^{7}\sum_{n=0}^{7}T(m,n) \quad \cos\left(\frac{(2m+1)x\pi}{16}\right) \quad \cos\left(\frac{(2x+1)y\pi}{16}\right)$$

donde

$$c(i) = \begin{cases} \frac{1}{\sqrt{2}} \text{ si } i = 0 \\ 1 \text{ de lo contrario} \end{cases}$$

Acrónimos

ALU: (*arithmetic logic unit*) unidad lógica aritmética.
ANSI: (*American National Standards Institute*) Instituto Nacional Norteamericano de Estándares.
ASCII: (*American Standard Code for Information Interchange*) Código norteamericano de estándares para intercambio de información.
bit: (*binary digit*) dígito binario.
CD-R: (*compact disc recordable*) disco compacto grabable.
CD-ROM: (*compact disc read-only memory*) disco compacto de memoria de sólo lectura.
CISC: (*complex instruction set computer*) computadora de serie de instrucciones complejas.
COBOL: (*COmmon Business-Oriented Language*) lenguaje común orientado a negocios.
CPU: (*central process unit*) unidad central de procesamiento .
DBMS: (*database management system*) sistema de administración de bases de datos.
DES: (*data encryption standard*) estándar de cifrado de datos.
DMA: (*direct memory access*) acceso directo a memoria.
DRAM: (*dynamic RAM*) RAM dinámica.
DVD: (*digital versatile disc*) disco versátil digital.
EBDIC: (*Extended Binary Coded Decimal Interchange Code*) Código extendido de intercambio decimal codificado en binario.
EEPROM: (*electronically erasable programmable read-only memory*) memoria de sólo lectura programable y borrable electrónicamente.
EPROM: (*erasable programmable read-only memory*) memoria de sólo lectura programable y borrable.
FIFO: (*first en, first out*) primero en llegar, primero en salir.
FORTRAN: (*FORmula TRANslation*: traducción de fórmulas.
FTP: (*file transfer protocol*) protocolo de transferencia de archivos.
HTML: (*hypertext markup language*) lenguaje para marcado de hipertexto.
HTTP: (*hypertext transfer protocol*) protocolo de transferencia de hipertexto.
IP: (*Internet protocol*) protocolo Internet.
ISO: (*International Standard Organization*) Organización para la Estandarización Internacional.
JPEG: (*joint photographic experts group*) grupo unido de expertos en fotografía.
LAN: (*local area network*) red de área local.
LIFO: (*last in, first out*) último en entrar, primero en salir.
LISP: (*LISt programming*) procesamiento de lista.
LZ: codificación de Lempel Ziv.
LZW: codificación de Lempel Ziv Welch.
MAN: (*metropolitan area network*) red de área metropolitana.
MPEG: (*motion picture experts group*) grupo de expertos en imágenes en movimiento.
OSI: (*Open Systems Interconnection*) Interconexión de sistemas abiertos.
PERL: (*Practical Extraction and Report Language*) lenguaje práctico de extracción e informes.
PC: (*personal computer*) computadora personal.
pixel: (*picture elements*) elementos de imagen.
PROM: (*programmable read-only memory*) memoria de sólo lectura programable.
RAM: (*random access memory*) memoria de acceso aleatorio.
RDBMS: (*relational database management system*) sistema de bases de datos relacionales.
RISC: (*reduced instruction set computer*) computadora de serie de instrucciones reducidas.
ROM: (*read-only memory*) memoria de sólo lectura.
RSA: cifrado de Rivest-Shamir-Adleman.
SCSI: (*small computer system interface*) interfaz pequeña de sistemas de computadoras.
SMTP: (*simple mail transfer protocol*) protocolo simple de transferencia de correo.
SQL: (*Structured Query Language*) lenguaje de consultas estructurado.
SRAM: (*static RAM*) RAM estática.
TDA: tipo de datos abstracto.
TCP: protocolo de control de transmisión.
TCP/IP: protocolo de control de transmisión/protocolo Internet.
TELNET: (*TErminal NETwork*) red terminal.
UDP: (*user datagram protocol*) protocolo de datagrama de usuario.
URL: (*uniform resource locator*) localizador uniforme de recursos.
USB: (*universal serial bus*) bus serial universal.
WAN: (*wide area network*) red de área amplia.
WORM: (*write once, read many*) escribir una vez, leer muchas.
WWW: World Wide Web.

Glosario

A

abstracción: la generalización de las operaciones de un algoritmo sin una implementación especificada.
acceso directo a memoria (DMA): una forma de E/S en la cual un dispositivo especial controla el intercambio de datos entre la memoria y los dispositivos de E/S.
acceso secuencial: un método de acceso en el cual se tiene acceso a los registros en un archivo en forma serial comenzando por el primer elemento.
acoplamiento: una medida de la interdependencia entre dos funciones separadas. Véase también acoplamiento de contenido, acoplamiento de control, acoplamiento global y acoplamiento de sello.
acoplamiento de contenido: la referencia directa a los datos en un módulo mediante instrucciones en otro módulo; la forma inferior de acoplamiento misma que debe evitarse.
acoplamiento de control: comunicación entre funciones en las cuales un módulo establece las banderas para controlar las acciones de otro.
acoplamiento de datos: comunicación entre módulos en la cual sólo se pasan los datos requeridos; considerada la mejor forma de acoplamiento.
acoplamiento de sello: la técnica de comunicación entre módulos en la cual los datos se pasan como una estructura; con frecuencia resulta en que se pasan datos no requeridos.
acoplamiento global: una técnica de comunicación en la cual los datos están accesibles para todos los módulos de un programa; se le considera un método muy pobre para la comunicación entre programas.
acoplamiento holgado: un tipo de acoplamiento entre módulos que los vuelve más independientes.
actualización en línea: un proceso de actualización en el cual un usuario que tiene acceso directo al sistema introduce y procesa las transacciones.
Ada: un lenguaje de programación de alto nivel concurrente desarrollado por el Departamento de Defensa de Estados Unidos.
administrador de archivos: el componente del sistema operativo que controla el acceso a los archivos.
administrador de dispositivos: un componente de un sistema operativo que controla el acceso a los dispositivos de entrada/salida.
agujero: en un disco óptico, una área que hace el láser en la traducción de un patrón de bits; por lo general representa un bit 0.
algoritmo: los pasos lógicos necesarios para resolver un problema con una computadora; una función o parte de una función.
almacenamiento de filas mayores: un método de ordenación de elementos de un arreglo en la memoria en el cual los elementos se almacenan fila a fila.
altura: un atributo de árbol que indica la longitud de la ruta desde la raíz al último nivel; el nivel de la hoja en la ruta más larga desde la raíz más 1.
análoga: una entidad que varía continuamente.
análisis sintáctico: un proceso que divide datos en piezas o señales.
apuntador: una constante o variable que contiene una dirección que puede utilizarse para tener acceso a los datos almacenados en alguna otra parte.
árbol: un conjunto de nodos conectados estructurado de tal manera que cada nodo sólo tiene un predecesor.
árbol casi completo: un árbol con grado de salida limitado que tiene la altura mínima para sus nodos y en la cual el nivel de hoja se llena desde la izquierda.
árbol de expansión: un árbol extraído de un grafo conectado que contiene todos los vértices en el grafo.
árbol de expansión mínima: un árbol extraído de una red conectada tal que la suma de los pesos es el mínimo de todos los árboles posibles contenidos en el grafo.
árbol de expresión: un árbol en el cual las hojas son elementos de datos y el nodo interno y la raíz son operadores.
árbol nulo: un árbol sin nodos.
arco: una línea dirigida a un grado. Contraste con arista.
archivo binario: una colección de datos almacenados en el formato interno de la computadora. Contraste con *archivo de texto*.

archivo de reporte de errores: en un proceso de actualización de archivos, un informe de errores detectado durante la actualización.

archivo de texto: un archivo en el cual todos los datos se almacenan como caracteres. Contrasta con archivo binario.

archivo de transacción: un archivo que contiene datos relativamente transitorios que se utilizan para cambiar el contenido de un archivo maestro.

archivo ejecutable: un archivo que puede ejecutarse (correr); un programa.

archivo fuente: el archivo que contiene instrucciones de programa escritas por un programador antes de que se conviertan en lenguaje de máquina; el archivo de entrada a un ensamblador o compilador.

archivo hashed: un archivo en el cual se busca utilizando uno de los métodos de hashing.

archivo maestro: un archivo permanente que contiene los datos más actuales referentes a una aplicación.

archivo maestro nuevo: el archivo maestro que se crea desde un archivo maestro viejo cuando se actualiza el archivo.

archivo maestro viejo: el archivo maestro que se procesa junto con el archivo de transacción para crear el nuevo archivo maestro.

archivo secuencial: una estructura de archivos en la cual los datos deben procesarse en forma serial desde el primer elemento en el archivo.

área principal: en una lista hashed, la memoria que contiene la dirección base.

arista: una línea de grafo que no tiene dirección.

arreglo bidimensional: un arreglo con elementos que tiene dos niveles de indexación. Véase también arreglo multidimensional.

arreglo multidimensional: un arreglo con elementos que tienen más de un nivel de indexación.

ASCII: Véase Código norteamericano de estándares para intercambio de información.

ASCII extendido: un conjunto de caracteres que amplía el ASCII básico. Los caracteres extras representan caracteres para lenguajes extranjeros así como para otros símbolos.

B

bandera: un indicador utilizado en un programa para designar la presencia o ausencia de una condición; switch.

base de datos: una colección de información organizada.

base de datos distribuida: una base de datos en la cual los datos se almacenan en varias computadoras.

base de datos distribuida fragmentada: una base de datos distribuida en la cual se localizan los datos.

base de datos distribuida replicada: una base de datos en la cual cada sitio aloja una réplica de otro sitio.

base de datos orientada a objetos: una base de datos en la cual los datos se tratan como estructuras (objetos).

base de datos relacional: un modelo de base de datos en el cual los datos se organizan en tablas relacionadas llamadas relaciones.

bit: acrónimo de binary digit (dígito binario). En una computadora, la unidad de almacenamiento básico con un valor ya sea de 0 o 1.

bloque: un grupo de instrucciones tratadas como un todo.

bus: el canal físico que enlaza los componentes de hardware en una computadora; el medio físico compartido utilizado en una red de topología de bus.

bus de control: el bus que transporta información entre componentes de computadoras.

bus de datos: el bus dentro de una computadora utilizado para transportar datos entre componentes.

bus de dirección: la parte del bus del sistema utilizada para transferir direcciones.

bus serial universal (USB: universal serial bus): un controlador de dispositivos de E/S que conecta dispositivos más lentos, tales como el teclado y el ratón, a una computadora.

búsqueda: el proceso que examina una lista para localizar uno o más elementos que contienen un valor designado conocido como el argumento de búsqueda.

búsqueda binaria: un algoritmo de búsqueda en el cual el valor de búsqueda se localiza al dividir repetidamente la lista a la mitad.

búsqueda en una lista: Véase búsqueda.

búsqueda secuencial: una técnica de búsqueda utilizada con una lista lineal en la cual la búsqueda comienza con el primer elemento y continúa hasta que el valor de un elemento igual al valor que se busca se localiza o hasta que el final de la lista se alcanza.

byte: una unidad de almacenamiento, por lo general 8 bits.

caja negra: un dispositivo con mecanismos internos desconocidos para el operador.

C

calidad del software: software que satisface los requisitos explícitos e implícitos del usuario, está bien documentado, cumple con los estándares operativos de la organización y se ejecuta de manera eficiente en el

camino: una secuencia de nodos en los cuales cada vértice es adyacente al siguiente.

campo: la unidad nombrada de datos más pequeña que tiene significado en la descripción de información. Un campo puede ser una variable o una constante.

capa física: la primera capa en el modelo OSI; responsable por formar y transmitir bits a lo largo de la red.

capa de presentación: la sexta capa en el modelo OSI; responsable de dar formato a los datos, el cifrado/descifrado y la compresión.

capa de sesión: el quinto nivel en el modelo OSI; responsable de establecer y terminar sesiones y controlar diálogos.

capa de transporte: la cuarta capa en el modelo OSI; responsable por la entrega extremo a extremo de todo el mensaje.

capacidad de corrección: el factor de calidad que aborda la facilidad con la cual pueden corregirse los errores en un módulo.

capacidad de pruebas: un atributo del software que mide la facilidad con la cual el software puede probarse como un sistema operacional.

cargador: la función del sistema operativo que busca y trae un programa ejecutable a la memoria para su ejecución.

char: el tipo de dato en lenguaje C para carácter.

ciclo controlado por contador: una técnica de ciclo en la cual el número de iteraciones es controlado por un contador.

ciclo controlado por eventos: un ciclo cuya terminación se basa en la ocurrencia de un evento especificado. Contraste con ciclo controlado por contador.

ciclo de máquina: la secuencia repetitiva de eventos en la ejecución de instrucciones de programa (fetch, decode y execute).

ciclo de postprueba: un ciclo en el cual la condición de finalización se prueba sólo después de la ejecución de las instrucciones de ciclo. Contraste con ciclo de preprueba.

ciclo de preprueba: un ciclo en el cual la condición de finalización se prueba antes de la ejecución de las instrucciones de ciclo. Contraste con ciclo de postprueba.

ciclo de vida del desarrollo del sistema: una secuencia de pasos requeridos para desarrollar software; comienza con la necesidad de software y concluye con su implementación.

ciclo de vida del software: la vida de un paquete de software.

ciclo do-while: en los lenguajes C y C++, un ciclo postprueba controlado por eventos.

ciclo for: un ciclo controlado por contador en C y C++.

ciclo while: un ciclo controlado por eventos en C y C++.

ciclo: en un programa, un constructor de la programación estructurado que provoca que una o más instrucciones se repitan; en un grafo es una línea que comienza y finaliza con el mismo vértice.

ciencias de la computación: estudio de los temas relacionados con una computadora.

cifrado: conversión de un mensaje a una forma ininteligible que es ilegible a menos que se descifre.

cifrado a nivel de bits: un método de cifrado en el cual los datos primero se dividen en bloques de bits antes de cifrarse.

cifrado a nivel de carácter: un método de cifrado en el cual el carácter es la unidad de cifrado.

cifrado de llave pública: un método de cifrado que utiliza dos llaves: privada y pública. La llave privada se mantiene en secreto; la llave pública se revela al público.

cifrado de Rivest-Shamir-Adleman (RSA): un método de cifrado de llave pública popular.

cinta magnética: un medio de almacenamiento con capacidad secuencial.

circuito integrado: transistores, cableado y otros componentes en un solo chip.

círculo de calidad: un diagrama en forma circular de los pasos para la calidad del software.

clase: la combinación de datos y funciones unidos para formar un tipo.

clear: en las máscaras, una técnica para hacer un bit 0; también conocida como forzar a 0.

cliente: en un programa cliente-servidor, la aplicación que solicita servicios de un servidor.

COBOL: un lenguaje de programación de negocios (COmmon Business-Oriented Languaje) desarrollado por Grace Hopper.

codificación basada en diccionario: un método de compresión en el cual un diccionario se crea durante la sesión.

codificación de Huffman: un método de compresión estadístico que utiliza código de longitud variable.

codificación de Lempel Ziv (LZ): un algoritmo de compresión que utiliza un diccionario.

codificación de Lempel Ziv Welch (LZW): una versión mejorada de la codificación LZ.

codificación de longitud de ejecución: un método de compresión sin pérdida en el cual una serie de símbolos se reemplaza mediante el símbolo y el número de símbolos repetidos.

código: una serie de patrones de bits diseñados para representar símbolos de texto.

código de intercambio decimal codificado en binario extendido (EBCDIC: Extended Binary Coded Decimal Interchange Code): el conjunto de caracteres diseñado por IBM para sus sistemas de cómputo más grandes.

Código norteamericano de estándares para intercambio de información (ASCII: American Standard Code for Information Interchange): un esquema de codificación que define los caracteres de control e imprimibles para 128 valores.

cohesión: el atributo de un módulo que describe qué tan estrechamente se relacionan los procesos escritos en un módulo entre sí.

cohesión casual: la combinación de procedimientos que no están relacionados.

cohesión de comunicación: un atributo de diseño en el cual los procesos de módulo se relacionan debido a que comparten los mismos datos.

cohesión de procedimiento: un atributo de diseño en el cual el procesamiento dentro del módulo se relaciona con los flujos de control. Se considera un modelo aceptable sólo en los niveles más altos de un programa.

cohesión funcional: un atributo de diseño en el cual todo el procesamiento se relaciona con una sola tarea. El nivel más alto de cohesión.

cohesión lógica: un atributo de diseño que describe un módulo en el cual el procesamiento dentro del módulo se relaciona sólo con el tipo general de procesamiento que se está realizando. Se considera inaceptable en la programación estructurada.

cohesión secuencial: un atributo de diseño en el cual el proce-

samiento dentro del módulo es tal que los datos de un proceso se utilizan en el siguiente proceso.

cohesión temporal: un diseño de módulo en el cual los procesos se combinan debido a que todos necesitan procesarse en la misma secuencia de tiempo.

cola de espera: una lista lineal en la cual los datos sólo pueden insertarse en un extremo, llamado la parte de atrás, y eliminarse desde el otro extremo llamado la parte de adelante.

colisión: en hashing, un evento que ocurre cuando un algoritmo de hashing produce una dirección para una inserción y esa dirección ya está ocupada.

columna vertebral backbone: los medios y dispositivos que crean conectividad para redes pequeñas.

comentario: en un programa C, una nota para el lector del programa que es ignorada por el compilador.

compilador: software del sistema que convierte un programa fuente en código objeto ejecutable; tradicionalmente asociado con lenguajes de alto nivel. Véase también ensamblador.

complemento a uno: la operación relacionada con bits que invierte el valor de los bits en una variable.

complemento a dos: una representación de números binarios en la cual el complemento de un número se encuentra al complementar todos los bits y sumar un 1 después de eso.

compresión: la reducción de un mensaje sin pérdida significativa de información.

compresión de datos con pérdida: compresión de datos en la cual se permite que algunos datos se pierdan; utilizada para compresión de imagen, audio o video.

compresión de datos sin pérdida: compresión de datos en la cual no se pierde ningún dato; utilizada para comprimir texto o programas.

compresión de datos: la reducción de la cantidad de datos sin una pérdida significativa.

compresión espacial: compresión realizada por JPEG en un cuadro.

compresión estadística: un método de compresión en el cual la codificación se basa en la frecuencia de los símbolos.

compresión temporal: compresión realizada por MPEG en los cuadros.

computadora de conjunto de instrucciones complejas (CISC: complex instruction set computer): una computadora que define un amplio conjunto de instrucciones, incluso aquellas que se utilizan con menor frecuencia.

computadora de conjunto de instrucciones reducidas (RISC: reduced instruction set computer): una computadora que utiliza sólo las instrucciones utilizadas con frecuencia.

computadora personal (PC): una computadora diseñada para uso individual.

concentrador activo: un concentrador (hub) que regenera las señales recibidas (un repetidor).

concentrador pasivo: un tipo de dispositivo de conexión que no regenera los datos.

conmutador: Véase bandera.

constante: un valor de datos que no puede cambiar durante la ejecución del programa. Contraste con variable.

constante literal: una constante sin nombrar codificada en una expresión.

constante nombrada: una constante a la que el programador da un nombre.

constante simbólica: una constante que está representada por un indentificador.

contador de programa: un registro en el CPU que mantiene la dirección de la siguiente instrucción a ejecutarse en la memoria.

contenedor: un algoritmo de hashing, una localidad que puede acomodar varias unidades de datos.

controlador de entrada/salida (E/S): un dispositivo que controla el acceso a los dispositivos de entrada/salida.

controlador: un componente de una máquina Turing que es equivalente al CPU de una computadora.

conversión binaria a decimal: el cambio de un número binario a número decimal.

conversión decimal a binaria: el cambio de un número decimal a número binario.

correo electrónico (e-mail): un método para enviar mensajes electrónicamente basados en la dirección del buzón de correo electrónico en vez del intercambio de anfitrión a anfitrión.

CPU: *Véase unidad central de procesamiento.*

cuadro bidireccional (cuadro B): en MPEG un cuadro que se relaciona tanto con los cuadros precedentes como con los posteriores.

cuadro intracodificado (cuadro I): en MPEG, un cuadro independiente.

cuadro pronosticado (cuadro P): en MPEG, un cuadro que se relaciona con el cuadro I o el cuadro B precedentes.

cuantización: asignar un valor de una serie finita de valores.

cuerpo: la parte de una función que contiene las definiciones e instrucciones; el todo de una función con excepción de la declaración del encabezado. Contraste con encabezado de función.

cuerpo de la función: el código dentro de una función contenido dentro de la definición de la función y las secciones de instrucciones.

D

datagrama: el paquete enviado mediante el protocolo IP.

datagrama de usuario: el nombre de la unidad de datos utilizados por el protocolo VDP.

datagrama IP: la unidad de datos en el nivel de red.

datos de entrada: información del usuario que se presenta a una computadora para ejecutar un programa.

datos de salida: los resultados de ejecutar un programa de computadora.

datos lógicos: datos con un valor ya sea de verdadero o falso.

declaración: en C, la asociación de un nombre con un objeto, tal como un tipo, una variable, una estructura o una función. *Véase también definición.*

declaración de función: en C, una instrucción de prototipo que describe el tipo de regreso de la función, el nombre y los parámetros formales.

decodificación: proceso de restaurar un mensaje codificado a su forma anterior a la codificación.

definición de función: en C, la implementación de una declaración de función.

definición: en C, el proceso que reserva memoria para un objeto nombrado, tal como una variable o constante.

desbordamiento: la condición que resulta cuando no hay suficientes bits para representar un número en binario. Véase también declaración.

descendiente: cualquier nodo en el camino desde el nodo actual a una hoja.

descifrado: la recuperación del mensaje original desde los datos cifrados. Véase cifrado.

descompresión: la acción realizada en los datos comprimidos para obtener los datos originales.

diagrama de clase: un diagrama en la programación orientada a objetos que muestra la relación entre objetos.

diagrama de estado: un diagrama que muestra los diferentes estados de un proceso.

diagrama de flujo: una herramienta de diseño de programa en la cual los símbolos gráficos estándar se utilizan para representar el flujo lógico de datos a través de una función.

diagrama estructurado: una herramienta de diseño y documentación que representa un programa como un flujo jerárquico de funciones.

digital: una entidad discreta (no continua).

dígito binario (bit): la unidad más pequeña de información (0 o 1).

dígito hexadecimal: un símbolo en el sistema hexadecimal.

digrafo: un grado dirigido.

dirección base: en una lista hashed, la primera dirección producida por el algoritmo de hashing.

dirección de puerto: la dirección utilizada en TCP y UDP para distinguir un proceso de otro.

dirección física: la dirección de un dispositivo en el nivel de enlace de datos.

dirección Internet: una dirección de 32 bits utilizada para definir únicamente una computadora en Internet.

dirección IP: Véase dirección Internet.

dirección lógica: una dirección definida en el nivel de red.

directivas de preprocesador: comandos para el precompilador C.

disco: un medio de almacenamiento directo auxiliar para datos de computadora y programas.

disco compacto: un medio de almacenamiento óptico de acceso directo con una capacidad de 650 megabytes.

disco compacto de memoria de sólo lectura (CD-ROM): un disco compacto en el cual el fabricante escribe los datos en el disco y sólo pueden ser leídos por el usuario.

disco compacto escribible (CD-RW): un disco compacto en el que puede escribirse muchas veces y leerse muchas veces.

disco compacto grabable (CD-R): un disco compacto en el cual el usuario sólo puede escribir una vez y leerlo muchas veces.

disco maestro: el disco que aloja el archivo maestro en la actualización de archivos.

disco magnético: un medio de almacenamiento con capacidad de acceso aleatorio.

disco versátil digital (DVD): un medio de almacenamiento óptico de acceso directo que puede almacenar hasta 17 gigabytes (una película de 2 horas)

diseño de arriba hacia abajo: un concepto de diseño de programa en el cual un diseño progresa a través de una descomposición de las funciones, comenzando por la parte superior del diagrama estructurado y trabajando hacia los módulos interiores.

dispositivo de almacenamiento auxiliar o secundario: cualquier dispositivo de almacenamiento fuera de la memoria principal; almacenamiento de datos permanente; almacenamiento externo; almacenamiento secundario.

dispositivo de almacenamiento óptico: un dispositivo de E/S que utiliza luz (láser) para almacenar y recuperar datos.

dispositivo de almacenamiento secundario: *Véase dispositivo de almacenamiento auxiliar.*

dispositivo de almacenamiento: un dispositivo E/S que puede almacenar grandes cantidades de información para recuperar en un momento posterior.

dispositivo de interconexión de redes: dispositivos como enrutadores o gateways que conectan redes para formar una interred.

dispositivo de no almacenamiento: un dispositivo E/S que puede comunicarse con la memoria del CPU, pero no puede almacenar información.

dispositivo de salida: un dispositivo en el que puede escribirse pero no leerse.

dispositivos de conexión: dispositivos como enrutadores, puentes y repetidores que conectan LAN o WAN.

documentación del sistema: un registro estructurado formal de un paquete de software.

documento activo: en el World Wide Web, un documento ejecutado en el sitio local utilizando Java.

documento dinámico: un documento web creado mediante la ejecución de un programa en el sitio del servidor.

documento estático: una página web que se crea en el sitio remoto y es recuperada por el sitio local. Contraste con documento dinámico.

E

E/S aislada: un método de direccionamiento de un módulo E/S en el cual las instrucciones utilizadas para leer/escribir memoria, difieren totalmente de las instrucciones utilizadas para leer/escribir en dispositivos de entrada/salida.

E/S mapeada a memoria: un método de direccionamiento de un módulo E/S en un solo espacio de dirección; utilizada tanto para la memoria como para los dispositivos.

E/S manejada por interrupciones: una forma de E/S en la cual el CPU, después de emitir un comando de E/S, continúa sirviendo otros procesos hasta que recibe una señal de interrupción de que la operación de E/S se ha completado.

E/S programada: una forma de E/S programada en la cual el CPU debe esperar a que la operación de E/S se complete.

EBCDIC: Véase Código extendido de intercambio decimal codificado en binario.

editor de texto: software que crea y mantiene archivos de texto tales como un procesador de palabras o un editor de programa fuente.

efecto secundario: un cambio en una variable que resulta de la evaluación de una expresión; cualquier entrada/salida realizada por una función llamada.

eficiencia: el factor de calidad que trata el uso óptimo del hardware de computadora o la velocidad de respuesta para un usuario.

ejecutar: poner en funcionamiento un programa.

elemento de imagen (píxel): la unidad más pequeña de una imagen.

eliminación (dequeue): eliminación de un elemento de una cola.

encabezado: la información añadida al principio de un paquete para enrutamiento y otros propósitos.

encabezado de función: en una definición de función, la parte de la función que proporciona el tipo de regreso, el identificador de función y los parámetros formales. Contraste con cuerpo.

encapsulamiento: el concepto de diseño de ingeniería de software en el cual los datos y sus operaciones se incluyen juntos y se mantienen en forma separada de la aplicación que los utiliza.

enrutador: un dispositivo que opera en tres primeras capas del modelo OSI que conecta redes independientes. Un enrutador dirige un paquete con base en su dirección de destino.

enrutamiento: el proceso realizado por un enrutador.

ensamblador: software del sistema que convierte un programa fuente en código objeto ejecutable; asociado tradicionalmente con un programa de lenguaje ensamblador. Véase también compilador.

entero: un número entero, un número sin una parte fraccionaria.

entero negativo: un entero que varía de infinito negativo a 0.

entero positivo: un entero que varía de 0 a infinito positivo.

entero sin signo: un entero sin un signo; su valor varía entre 0 y el infinito positivo.

entrega de nodo a nodo: la entrega de datos desde un nodo al siguiente.

entrega de origen a destino: la entrega de un paquete de datos de la fuente al destino.

equilibrio: un atributo de nodo de árbol que representa la diferencia en altura entre los subárboles del nodo.

escribir una vez, leer muchas (WORM): otro nombre para un CD-R.

espacio de direcciones: un intervalo de direcciones.

espacio entre pistas: el vacío entre las pistas de un disco.

espacio entre sectores: el vacío entre los sectores de un disco.

espacios en blanco: en C, el espacio, los tabuladores horizontales y verticales, la línea nueva y los caracteres de cambio de página.

esquema: el estándar de facto del lenguaje LISP.

estado de ejecución: en la administración de procesos, un estado en el cual un proceso está utilizando el CPU.

estado de espera: el estado de una tarea que espera a ser cargada en la memoria.

estado de espera: un estado en el cual un proceso espera a recibir la atención del CPU.

estado de listo: en la administración de procesos, el estado de procesamiento en el cual el proceso está esperando obtener atención del CPU.

estado de terminación: en la administración de procesos, un estado en el cual un proceso ha terminado de ejecutarse.

estándar de cifrado de datos (DES): el método de cifrado del gobierno estadounidense para uso no militarizado y no clasificado.

estructura autorreferencial: una estructura que contiene un apuntador hacia sí misma.

estructura de datos: la representación sintáctica de los datos organizados para mostrar la relación entre los elementos individuales.

explorador: un programa de aplicación que despliega un documento del WWW.

expresión: una secuencia de operadores y operandos que se reduce a un solo valor.

expresión primaria: una expresión que consiste de un solo operador; la expresión de prioridad más alta.

extracción de dígitos: selección de dígitos de una llave para usar como una dirección.

extraer (pop): la operación de eliminación en una pila.

F

facilidad de cambio: el factor de calidad que aborda la facilidad con la cual los cambios pueden hacerse con precisión para un programa.

factor de equilibrio: en un árbol, la diferencia entre la altura del subárbol derecho y el subárbol izquierdo.

fase de análisis: una fase en el ciclo de vida del sistema de software que define los requisitos que especifican lo que el sistema propuesto va a lograr.

fase de diseño: una fase en el ciclo de vida del sistema de software que define cómo logrará el sistema lo que se definió en la fase de análisis.

fase de implementación: una fase en el ciclo de vida del software del sistema en la cual se crean los programas reales.

fase de pruebas: un fase en el ciclo de vida del software en la cual se realizan experimentos para probar que un paquete de software funciona.

fetch: la parte del ciclo de instrucciones en el cual la instrucción a ser ejecutada se trae desde la memoria.

fiabilidad: el factor de calidad que trata la confianza en la operación total de un sistema.

FIFO: *Véase primero en entrar, primero en salir.*

FireWire: un controlador de dispositivo de entrada/salida con una interfaz serial de alta velocidad que transfiere los datos en paquetes.

firma digital: un método utilizado para autenticar el emisor del mensaje y preservar la integridad de los datos.

flexibilidad: el factor de calidad que trata la facilidad con la cual un programa puede cambiarse para cumplir con los requisitos de los usuarios.

flip: cambiar un bit de 0 a 1 o de 1 a 0.

float: un tipo de punto flotante.

formato de doble precisión: un estándar para almacenar números de punto flotante en la memoria con mayor precisión que el formato de precisión simple.

FORTRAN: un lenguaje de procedimiento de alto nivel utilizado para aplicaciones científicas y de ingeniería.

forzar a 0: el concepto de manipulación de bits utilizado para cambiar los bits seleccionados a 0.

forzar a 1: el concepto de manipulación de bits utilizado para cambiar los bits seleccionados de a 1.

forzar al cambio: el concepto de manipulación de bits utilizado para cambiar los bits seleccionados de 0 a 1 y de 1 a 0.

fracción: una parte de un número entero.

función: un bloque nombrado de código que realiza un proceso dentro de un programa; una unidad ejecutable de código consistente en un encabezado, el nombre de la función y un cuerpo, diseñada para realizar una tarea dentro del programa.

G

gateway: un dispositivo que conecta dos redes separadas que utilizan diferentes protocolos de comunicación.

grado: el número de líneas incidentes a un nodo en un grafo.

grado de entrada: el número de líneas que entran a un nodo en un árbol o grafo.

grado de salida: el número de líneas que dejan un nodo en un árbol o grafo.

gráfica de barras: una gráfica con valores representados mediante barras.

gráfico de mapa de bits: una representación gráfica en la cual una combinación de pixeles define la imagen.

gráfico de vectores: el tipo de gráficos en el cual las líneas y las curvas se definen utilizando fórmulas matemáticas.

grafo: una colección de nodos, llamados vértices, y segmentos de línea, llamados aristas o arcos, que conectan pares de nodos.

grafo con peso: un grafo con líneas con peso. Cada línea tiene un entero que define el peso.

grafo débilmente conectado: un grafo en el cual hay al menos un nodo sin un camino para al menos uno de los otros nodos. Contraste con grafo fuertemente conectado.

grafo dirigido: un grafo en el cual la dirección se indica en las líneas (arcos).

grafo fuertemente conectado: un grafo en el cual hay un camino desde cada nodo a cada otro nodo. Contraste con grafo débilmente conectado.

grafo inconexo: un grafo que no está conectado.

grafo no dirigido: un grafo que consiste sólo de dos aristas; es decir, un grafo en el cual no hay indicación de la dirección de las líneas.

grupo de expertos en imágenes en movimiento (MPEG: motion picture experts group): un método de compresión con pérdida para comprimir video (y audio).

grupo unido de expertos en fotografía (JPEG: joint photographic experts group): un estándar para comprimir imágenes.

H

hardware: cualquiera de los componentes físicos de un sistema de cómputo, tales como un teclado o una impresora.

hashing de contendor: un método de hashing que utiliza contenedores para reducir la colisión.

hashing de extracción de dígitos: un método de hashing que utiliza la extracción de dígitos.

hashing de residuo de división: un tipo de hashing en el cual la llave se divide por un número y el residuo se utiliza como la dirección.

hashing directo: un método de hashing en el cual la llave se obtiene sin modificación algorítmica.

hashing doble: un método de resolución de colisiones de hashing en el cual la dirección de la colisión se produce para determinar la siguiente dirección.

herencia: capacidad para ampliar una clase para crear una clase nueva mientras se mantienen los objetos de datos y los métodos de la clase base y se añaden nuevos objetos de datos y métodos.

hermanos: nodos en un árbol con el mismo padre.

hipertexto: un documento con referencias a otros documentos.

histograma: una representación gráfica de una distribución de frecuencia. Véase también arreglo de frecuencia.

hoja: un grafo o nodo de árbol con una grado de salida de 0.

HTML: Véase lenguaje de marcación de hipertexto.

Hub: Un dispositivo que interconecta otros dispositivos en una red.

I

identificador: el nombre dado a un objeto en un lenguaje de programación.

imagen: datos en la forma de gráficos o imágenes.

inanición: un problema en la operación de un sistema operativo en el cual los procesos no pueden tener acceso a los recursos que necesitan.

incremento de prefijo: en C, el operador (por ejemplo, ++a) que suma 1 a una variable antes de que su valor se utilice en una expresión. También conocido como incremento unario.

índice: la dirección de un elemento en un arreglo.

infijo: una notación aritmética en la cual el operador se coloca entre dos operandos.

ingeniería de software: el diseño y la escritura de programas estructurados.

inicializador: una instrucción que inicializa el valor de una variable.

inicio de sesión local: un inicio de sesión en una computadora conectada directamente a la terminal.

inicio de sesión remoto: inicio de sesión en una computadora conectada a la computadora local.

insertar (enqueue): insertar un elemento en una cola de espera.

insertar (push): la operación de inserción de pila.

Instituto Nacional Norteamericano de Estándares (ANSI: American National Standards Institute): una organización que crea estándares en los lenguajes de programación, especificaciones eléctricas, protocolos de comunicación y así por el estilo.

instrucción compuesta: en algunos lenguajes de programación, una colección de instrucciones tratadas como una sola por el lenguaje.

instrucción de ciclo: una instrucción que provoca que el programa itere una serie de distintas instrucciones.

instrucción de decremento: la instrucción que resta 1 del valor de una variable.

instrucción de expresión: en C, una expresión terminada con un signo de punto y coma.

instrucción de incremento: en C o C++, la instrucción que suma un 1 a un valor entero.

instrucción de selección: una instrucción que elige entre dos o más alternativas. En C, las instrucciones if-else y switch.

instrucción if-else: un constructor que implementa una selección de dos sentidos.

instrucción switch: la implementación C de la selección de múltiples rutas.

instrucción: un comando que indica qué hacer a una computadora. Un constructor sintáctico en C que representa una operación en una función.

Interconexión de sistemas abiertos (OSI: Open Systems Interconnection): un modelo de siete capas diseñado por la ISO como una guía para comunicación de datos.

interfaz de usuario: un programa que acepta solicitudes de los usuarios (procesos) y las interpreta para el resto del sistema operativo

interfaz pequeña de sistemas de computadoras (SCSI: small computer system interface): un controlador de dispositivo de E/S con una interfaz paralela.

Internet: la interred global que utiliza el conjunto de protocolos TCP/IP.

interoperabilidad: el factor de calidad que trata la capacidad de un sistema para intercambiar datos con otro sistema.

interred: abreviatura de interconexión de redes.

iteración: una sola ejecución de instrucciones en un ciclo.

J

Java: un lenguaje de programación orientada a objetos para crear programas independientes o documentos dinámicos en Internet.

L

lenguaje C++: un lenguaje orientado a objetos desarrollado por Bjarne Stroustrup.

lenguaje C: un lenguaje de procedimiento desarrollado por Dennis Ritchie.

lenguaje de alto nivel: un lenguaje de programación (portátil) diseñado para permitir al programador concentrarse en la aplicación y no en la estructura de una computadora o sistema operativo en particular.

lenguaje de computadora: cualquiera de los lenguajes sintácticos utilizados para escribir programas para computadoras, tales como el lenguaje de máquina, el lenguaje ensamblador, C, COBOL y FORTRAN.

lenguaje de consultas estructurado (SQL: Structured Query Language): un lenguaje de base de datos que incluye instrucciones para definición, manipulación y control de la base de datos.

lenguaje de máquina: las instrucciones nativas para el procesador central de una computadora que son ejecutables sin ensamblaje o compilación.

lenguaje de marcación de hipertexto (HTML: hypertext markup language): el lenguaje de computadora para especificar el contenido y formato de un documento web; permite que el texto incluya fuentes, pantallas, gráficos incrustados y vínculos a otros documentos.

lenguaje de procedimiento (procedual): un lenguaje de computadora en el cual una serie de instrucciones por lo general se ejecuta una a una de principio a fin.

lenguaje de programación: un lenguaje con palabras limitadas y reglas limitadas diseñadas para resolver problemas en una computadora.

lenguaje declarativo: un lenguaje de computadora que utiliza el principio del razonamiento lógico para contestar consultas.

lenguaje ensamblador: un lenguaje de programación en el cual hay una correspondencia uno a uno entre el lenguaje de máquina de la computadora y el conjunto de instrucciones simbólicas del lenguaje.

lenguaje funcional: un lenguaje de programación en el cual un programa se considera una función matemática.

lenguaje imperativo: otro nombre para un lenguaje de procedimiento.

lenguaje natural: cualquier lenguaje hablado.

lenguaje orientado a objetos: un lenguaje de programación en el cual los objetos y las operaciones a aplicar en ellos se incluyen juntas.

lenguaje simbólico: un lenguaje de computadora, un nivel eliminado del lenguaje de máquina, que tiene un identificador nemónico para cada instrucción de máquina y tiene la capacidad de nombres de datos simbólicos.

LIFO: Véase último en entrar, primero en salir.

liga: en una estructura de lista, el campo que identifica el siguiente elemento en la lista.

ligador: la función en el proceso de creación del programa en la cual un módulo de objeto se une a funciones precompiladas para formar un programa ejecutable.

línea de tiempo: un atributo de softwarte que mide la rapidez de un sistema para los requisitos de tiempo de un usuario.

línea: un elemento de grafo que conecta dos vértices en el grafo. Véase también arco y arista.

Linux: un sistema operativo desarrollado por Linux Torvalds para volver a UNIX más eficiente cuando se ejecuta en un microprocesador Intel.

LISP: un lenguaje de programación de procesamiento de listas en el cual todo se considera una lista.

lista: un conjunto ordenado de datos contenidos en la memoria principal. Contraste con archivo.

lista aleatoria: una lista sin orden de los datos.

lista de adyacencia: un método de representación de un grafo que utiliza una lista ligada para almacenar los vértices y un arreglo de lista ligada bidimensional para almacenar las aristas.

lista de parámetros: una lista de valores pasados a una función.

lista doblemente ligada: una colección ordenada de datos en la cual cada elemento contiene dos apuntadores, uno que apunta hacia el elemento previo y uno que apunta hacia el elemento siguiente.

lista general: una lista en la cual los datos pueden insertarse o eliminarse en cualquier parte de la lista.

lista ligada: una estructura de lista lineal en la cual el orden de los elementos se determina mediante campos de liga.

lista ligada individualmente: una colección ordenada de datos en la cual cada elemento contiene sólo la localización del siguiente elemento. Contraste con lista doblemente ligada.

lista lineal: una estructura de lista en la cual cada elemento, excepto el último, tiene un sucesor único.

lista ordenada: una lista en la cual los elementos se acomodan de manera que los valores clave se coloquen en secuencia ascendente o descendente.

lista restringida: una lista en la cual los datos sólo pueden añadirse o eliminarse en los extremos de la lista y el procesamiento está restringido a operaciones en los datos de los extremos.

llamada de función: una instrucción que invoca a otra función.

llave: uno o más campos utilizado para identificar un registro (estructura).

llave privada: una de las dos llaves utilizadas en el cifrado de llave pública.

llave pública: una de las llaves en un cifrado de llave pública; se revela al público.

llave secreta: una llave que se comparte por dos participantes en el cifrado de llave secreta.

M

macro: un procedimiento diseñado personalmente que puede utilizarse una y otra vez.

mantisa: la parte de un número de punto flotante que muestra la precisión.

máquina analítica: la computadora inventada por Charles Babbage.

máquina de Turing: un modelo de computadora con tres componentes (cinta, controlador y cabeza de lectura/escritura) que puede implementar instrucciones en un lenguaje de computadora.

máscara: una variable o constante que contiene una configuración de bits utilizada para controlar el arreglo de bits en una operación relativa a los bits.

matriz de adyacencia: un método de representación de un grafo que utiliza un arreglo para los vértices y una matriz (arreglo bidimensional cuadrado) para almacenar las aristas.

memoria: la memoria principal de una computadora consistente en memoria de acceso aleatorio (RAM) y memoria de sólo lectura (ROM); utilizada para almacenar datos e instrucciones de programa.

memoria caché: una memoria pequeña y rápida utilizada para almacenar elementos de datos que se están procesando.

memoria de acceso aleatorio (RAM: random access memory): la memoria principal de la computadora que almacena datos y programas.

memoria de sólo lectura (ROM: read-only memory): memoria permanente con contenido que no puede cambiarse.

memoria de sólo lectura programable (PROM): memoria con contenidos eléctricamente establecidos por el fabricante; el usuario puede restablecerla.

memoria de sólo lectura programable borrable electrónicamente (EPROM): memoria de sólo lectura programable que puede programarse y borrarse usando impulsos electrónicos sin ser eliminados de la computadora.

memoria de sólo lectura programable y borrable electrónicamente (EEPROM: electronically erasable programmable read-only memory): memoria de sólo lectura programable que puede programarse; el borrado de la EPROM requiere que ésta sea removida de la computadora.

memoria primaria: la memoria de alta velocidad de una computadora, donde los programas y los datos se almacenan cuando el programa se está ejecutando. La memoria primaria es volátil, lo cual significa que el contenido se borra cuando la computadora se apaga; memoria principal.

memoria principal: *Véase memoria primaria.*

memoria virtual: la memoria que resulta de intercambiar programas hacia el interior y el exterior de la memoria durante la ejecución para dar la impresión de un memoria principal más grande de la que realmente existe.

método de acceso: una técnica para leer datos desde un dispositivo de almacenamiento secundario (auxiliar).
microcomputadora: una computadora lo suficientemente pequeña para que quepa en un escritorio.
modelo: la especificación establecida por una organización de estándares como un lineamiento para diseñar redes.
modelo cliente-servidor: el modelo de interacción entre dos programas de aplicación en los cuales un programa en un extremo (cliente) solicita un servicio de un programa en el otro extremo (servidor).
modelo de cascada: un modelo de desarrollo de software en el cual cada módulo se termina completamente antes de que comience el siguiente módulo.
modelo de red: un modelo de base de datos en el cual un registro puede tener más de un registro padre.
modelo de von Neumann: un modelo de computadora (consistente en memoria, la unidad lógica aritmética, la unidad de control y los subsistemas de entrada/salida), sobre el cual la computadora moderna se fundamenta.
modelo jerárquico: un modelo de base de datos que organiza datos en una estructura tipo árbol en el que se pueden realizar búsquedas desde la parte superior a la parte inferior.
modelo incremental: un modelo en ingeniería de software en el cual todo el paquete se construye con cada módulo conformado por sólo una consola de comandos; los módulos ganan complejidad con cada interación del paquete.
modelo relacional: *Véase base de datos relacional.*
módulo: *Véase subalgoritmo.*
módulo objeto: la salida de una compilación que consiste de instrucciones en lenguaje de máquina.
módulo de la división: dividir dos números y mantener el residuo.
monitor: la unidad de despliegue visual de un sistema de cómputo; por lo general un dispositivo de despliegue de video.
monoprogramación: la técnica que permite que sólo un programa esté en la memoria a la vez.
muestreo: tomar medidas a intervalos iguales.
multiprogramación: una técnica que permite a más de un programa residir en la memoria mientras se está procesando.

N

nivel: un atributo de un nodo que indica su distancia desde la raíz.
nivel conceptual: relativo a la estructura lógica de la base de datos. Trata con el significado de la base de datos, no con su implementación física.
nivel de aplicación: el séptimo nivel en el modelo OSI; proporciona acceso a los servicios de red.
nivel de enlace de datos: el segundo nivel en el modelo OSI; responsable de la entrega de datos nodo a nodo.
nivel de red: el tercer nivel del modelo OSI, responsable de la entrega de paquetes desde el anfitrión original hasta el destino final.
nivel externo: la parte de la base de datos que interactúa con el láser.
nivel interno: la parte de la base de datos que define dónde se almacenan en realidad los datos.
nodo: en una estructura de datos, un elemento que contiene tanto datos como elementos estructurales utilizados para procesar la estructura de datos.
nodo interno: cualquier nodo de árbol excepto la raíz y las hojas; un nodo en medio de un árbol.
normalización: en una base de datos relacional, el proceso de eliminar redundancias.
notación O: una medida de la eficiencia de un algoritmo con sólo el factor dominante considerado.
notación hexadecimal: un sistema de numeración con base 16. Sus dígitos son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.
notación octal: un sistema de numeración con una base de 8; los dígitos octales son de 0 a 7.
número de Gödel: un número asignado a cada programa que puede escribirse en un lenguaje específico.
número de punto flotante: un número que contiene tanto un entero como una fracción.
número entero: *Véase entero.*

O

operabilidad: el factor de calidad que trata la facilidad con la cual puede utilizarse un sistema.
operación AND: una de las operaciones a nivel de bits; el resultado de la operación es 1 sólo si ambos bits son 1; de lo contrario es 0.
operación aritmética: una operación que toma dos números y crea otro número.
operación binaria: una operación que necesita dos operandos de entrada.
operación de actualización: una operación en una base de datos relacional en la cual se cambia la información sobre una tupla.
operación de diferencia: una operación en dos conjuntos. El resultado es el primer conjunto menos los elementos comunes en las dos conjuntos.
operación de intersección: una operación en dos conjuntos en la cual el resultado es un conjunto con los elementos comunes a los dos conjuntos.
operación de proyección: una operación en una base de datos relacional en la cual se selecciona una serie de columnas con base en un criterio.
operación de selección: una operación en una base de datos relacional que selecciona una serie de tuplas.
operación de unión: una operación en dos conjuntos en la cual el resultado contiene todos los elementos de ambos conjuntos sin duplicados.
operación de XOR: una operación relacionada con bits cuyo resultado es 1 sólo si uno de los operandos es 1.
operación lógica: una operación en la cual el resultado es un valor lógico (verdadero o falso).
operación NOT: la operación que cambia un bit 0 a 1 o un bit 1 a 0.

operación OR: una operación binaria que da como resultado una salida de 0 sólo si las dos entradas son 0; de lo contrario es 1.
operación unaria: una operación que necesita sólo un operando de entrada.
operador: la señal sintáctica que representa una acción sobre los datos (el operando). Contraste con operando.
operador AND: el operador utilizado en la operación AND.
operador aritmético: el operador utilizado en una operación aritmética.
operador binario: un operador que se utiliza en una operación binaria.
operador de actualización: en una base de datos relacional, un operador que cambia algunos valores en una tupla.
operador de asignación: en C y C++, el operador que asigna un valor a una variable.
operador de diferencia: un operador en una base de datos relacional que se aplica a dos relaciones con los mismos atributos. La tuplas en la relación resultante son aquellas que están en la primera relación pero no en la segunda.
operador de eliminación (delete): en una base de datos relacional, el operador que elimina una tupla de la relación.
operador de inserción: un operador en una base de datos relacional que inserta una tupla en una relación.
operador de intersección: un operador en el álgebra relacional que encuentra tuplas comunes entre dos relaciones.
operador de juntura: un operador en una base de datos relacional que toma dos relaciones y las combina con base en sus atributos comunes.
operador de proyección: un operador de álgebra relacional (utilizado en bases de datos relacionales) en el cual una columna o columnas de datos se extraen con base en un criterio dado.
operador de selección: un operador de álgebra relacional que extrae tuplas con base en el criterio seleccionado.
operador de unión: en el álgebra relacional, un operador que combina filas de dos relaciones.
operador lógico: un operador que realiza una operación lógica.
operador NOT: el operador utilizado en una operación NOT.
operador OR: el operador usado en una operación OR.
operador relacional: un operador que compara dos valores.
operador unario: un operador que realiza una operación unaria.
operador XOR: el operador utilizado en una operación XOR.
operando: un objeto en una instrucción en la cual se realiza una operación. Contraste con operación.
OR exclusivo (XOR): una operación lógica binaria en la cual el resultado es verdadero sólo si uno de los operandos es verdadero y el otro falso.
orden por burbuja: un algoritmo de ordenación en el cual cada paso a través de los datos mueve (burbujea) el elemento inferior al principio de la porción sin ordenar de la lista.
ordenamiento por inserción: un algoritmo de ordenación en el cual el primer elemento de la porción de la lista sin ordenar se inserta en su posición adecuada en la lista ordenada.
orden interno: un orden en el cual todos los datos se alojan en el almacenamiento principal durante el proceso de ordenación.
ordenación: el proceso que ordena una lista o un archivo.
Organización para la Estandarización Internacional (ISO: International Standard Organization): una organización mundial que define y desarrolla estándares para una variedad de tópicos.

P

padre: un árbol o nodo de grafo con un grado de salida mayor que 0; es decir, con sucesores.
página: una de una serie de secciones de igual tamaño de un programa.
página web: una unidad de hipertexto o hipermedia disponible en el Web.
página principal: la página principal de un documento de hipertexto disponible en el Web.
paginación: una técnica de multiprogramación en la cual la memoria se divide en secciones de igual tamaño llamadas bloques.
paginación bajo demanda y segmentación: un método de asignación de memoria en el cual una página o un segmento de un programa se carga en la memoria sólo cuando se requiere.
paginación bajo demanda: un método de asignación de memoria en el cual una página de un programa se carga en la memoria sólo cuando ésta se requiere.
palabras clave: Véase palabras reservadas.
palabras reservadas: la serie de palabras en un lenguaje que tiene una interpretación predeterminada y no puede estar definida por el usuario.
parámetro: un valor pasado a una función.
parámetros formales: la declaración de parámetros en una función para describir los tipos de datos que se pasarán a la función.
parámetros reales: los parámetros en la instrucción de llamada de función que contienen los valores que se van a pasar a la función. Contraste con parámetros formales.
particionamiento: una técnica utilizada en la multiprogramación que divide la memoria en secciones de longitud variable.
Pascal: un lenguaje de programación diseñado con un objetivo específico en mente: enseñar programación a los novatos al enfatizar el método de programación estructurada.
paso de ordenación: un ciclo durante el cual todos los elementos se prueban mediante un programa de ordenación.
paso por referencia: una técnica de paso de parámetros en la cual la función llamada se refiere a un parámetro pasado utilizando un alias.
paso por valor: una técnica de paso de parámetros en la cual el valor de una variable se pasa a una función.
patrón de bits: una secuencia de bits (0 y 1).
PERL: un lenguaje de alto nivel (con una sintaxis similar a C) que utiliza expresiones regulares que permiten el análisis sintáctico de una cadena de caracteres en componentes.
permutación: revoltura.

pila: una estructura de datos restringida en la cual los datos pueden insertarse y eliminarse sólo en un extremo, llamado cima.
pista: parte de un disco.
pixel: *Véase elemento de imagen.*
planificación: asignar los recursos de un sistema operativo a diferentes programas, y decidir cuáles programas deben utilizar cuáles recursos y cuándo.
planificador de procesos: un mecanismo del sistema operativo que despacha los procesos que esperan obtener acceso al CPU.
planificador de tareas: un planificador que selecciona una tarea para su procesamiento de una cola de espera de tareas que aguardan a ser movidas a la memoria.
polimorfismo: en C++, definir varias operaciones con el mismo nombre que pueden hacer diferentes cosas en las clases relacionadas.
portabilidad: el factor de calidad relacionado con la facilidad con la cual un sistema puede moverse a otros entornos de hardware.
postergación: cambiar el orden de acceso de los datos.
postfijo: una notación aritmética en la cual el operador se coloca después de sus operandos.
precisión: el factor de calidad que trata la repetibilidad de un sistema.
prefijo: una notación aritmética en la cual el operador se coloca antes de los operandos.
preprocesador: la primera fase de una compilación C en la cual las instrucciones fuente se preparan para la compilación y se cargan cualesquier bibliotecas necesarias.
primero en entrar, primero en salir (FIFO: first en, first out): un algoritmo en el cual el primer elemento de datos que se añade a una lista es el primero que se elimina de la lista.
problema sin solución: un problema que no puede resolverse mediante una computadora.
problema no polinomial: un problema que no puede resolverse con complejidad polinomial.
problema polinomial: un problema que puede resolverse con la complejidad polinomial.
problema con solución: un problema que puede resolverse mediante una computadora.
procedimiento: otro término para un subalgoritmo.
procesador de datos programable: una máquina que toma datos de entrada y un programa para producir datos de salida.
procesador de datos: una entidad que introduce datos, los procesa y genera el resultado.
proceso: un programa en ejecución.
programa: una serie de instrucciones.
Prolog: un lenguaje de programación de alto nivel basado en la lógica formal.
protocolo: una serie de reglas para intercambio de datos entre computadoras.
protocolo de control de transmisión (TCP): uno de los protocolos del nivel de transporte en el conjunto de protocolos TCP/IP.
protocolo de control de transmisión/protocolo Internet (TCP/IP): el protocolo oficial de Internet, formado por cinco niveles.
protocolo de datagrama de usuario (UDP): uno de los protocolos de capa de transporte en el conjunto de protocolos TCP/IP.
protocolo de transferencia de archivos (FTP: file transfer protocol): un servicio del nivel de aplicación en TCP/IP para transferir archivos desde y hacia un sitio remoto.
protocolo de transferencia de hipertexto (HTTP: hypertext transfer protocol): el protocolo que se utiliza para recuperar páginas web en Internet.
protocolo Internet (IP): el protocolo de nivel de red en el protocolo TCP/IP responsable de la transmisión de paquetes de una computadora a otra a través de Internet.
protocolo simple de transferencia de correo (SMTP): el protocolo TCP/IP para servicio de correo electrónico.
pruebas de caja blanca: pruebas de programa en las cuales se considera el diseño interno del programa; también se le conoce como pruebas de caja clara.
pruebas de caja negra: pruebas basadas en los requisitos del sistema en lugar de un conocimiento del programa.
pseudocódigo: instrucciones del tipo del idioma inglés que siguen una sintaxis definida libremente y se utilizan para expresar el diseño de un algoritmo o función.
puente: un dispositivo de conexión que opera en los primeros dos niveles del modelo OSI con capacidades de filtrado y envío.
punto de sincronización: un punto introducido en los datos por el nivel de sesión para propósito de control de flujo y de errores.
punto muerto: una situación en la cual los recursos requeridos por una tarea para terminar su trabajo están alojados en otros programas.

N

notación de punto decimal: la notación ideada para facilitar la lectura de las direcciones IP; cada byte se convierte a número decimal; los números están separados mediante un punto.

R

raíz: el primer nodo de un árbol.
RAM dinámica (DRAM): RAM en la cual las celdas usan capacitores. La DRAM debe refrescarse periódicamente para retener sus datos.
RAM estática (SRAM): una tecnología que utiliza las compuestas flip-flop tradicionales (una compuerta con dos estados: 0 y 1) para almacenar datos.
rama: un línea en un árbol que conecta dos nodos adyacentes.
recorrido de árbol binario: el proceso de visitar cada nodo en un árbol binario.
recorrido de lista ligada: un método de recorrido en el cual cada elemento de una lista ligada se procesa en orden.

recorrido en past orden: un método de recorrido de árbol binario en el cual el subárbol izquierdo se procesa primero, luego el subárbol derecho y después la raíz.
recorrido en pre orden: un recorrido de árbol binario en el cual el subárbol izquierdo se recorre primero, la raíz se recorre enseguida y el subárbol derecho se recorre al último.
recorrido en orden: un método de recorrido de árbol binario en el cual la raíz se recorre después del subárbol izquierdo y antes del subárbol derecho.
recorrido primero en amplitud: un método de recorrido gráfico en el cual los nodos adyacentes al nodo actual, se procesan antes que sus descendientes.
recorrido primero en profundidad: un método de recorrido en el cual todos los descendientes de los nodos se procesan antes de cualesquier nodos adyacentes (hermanos).
recorrido: un proceso algorítmico en el cual cada elemento en una estructura se procesa una vez y sólo una vez.
recuperación: la localización y devolución de un elemento en una lista.
recursividad: un diseño de función en el cual la función se llama a sí misma.
red: un sistema de nodos conectados que pueden compartir recursos.
red de área amplia (WAN): una red que abarca una distancia geográfica grande.
red de área local (LAN: local area network): una red que conecta dispositivos dentro de un área limitada.
red de área metropolitana (WAN): una red que puede abarcar una ciudad o un pueblo.
red de computadoras: *Véase red.*
redes interconectadas: una red de redes.
registro autorreferencial: un registro en el cual parte del registro se utiliza para apuntar a otro registro del mismo tipo.
registro de datos: un área que aloja los datos a ser procesados dentro del CPU.
registro de instrucción: un registro en el CPU que aloja la instrucción antes de que sea interpretada por la unidad de control.
registro: la información relacionada con una entidad. Una localidad de almacenamiento rápido independiente que aloja los datos en forma temporal.
relación: una tabla en una base de datos relacional.
representación de signo y magnitud: un método de representación de enteros en el cual 1 bit representa el signo del número y los bits restantes representan la magnitud.
representación del complemento a dos: un método de representación de enteros en el cual un número negativo se representa al dejar todos los 0 en el extremo derecho y el primer 1 sin cambios y complementar los bits resta
representación en complemento a uno: un método de representación de enteros en el cual un número negativo se representa al complementar el número positivo.
resina de policarbonato: en la producción de CD-ROM, un material inyectado en un molde.
resolución de colisiones: un proceso algorítmico que determina una dirección opcional después de una colisión.
resolución de direccionamiento abierto: un método de resolución de colisiones en el cual la nueva dirección está en el área base.
resolución de lista ligada: un método de resolución de colisiones en hashing que utiliza un área separada para los sinónimos, los cuales se mantienen en una lista ligada.
retroceso: un proceso algorítmico, por lo general implementado con una pila o a través de la recursión, que recuerda la ruta a través de una estructura de datos y puede regresar por la ruta en orden inverso.
reutilización: el factor de calidad que trata la facilidad con la cual el software puede utilizarse en otros programas.

S

sector: una parte de una pista en un disco.
segmentación bajo demanda: un método de asignación de memoria en el cual un segmento de un programa se carga en la memoria sólo cuando éste se necesita.
seguridad: el factor de calidad que trata la facilidad o dificultad con la cual un usuario no autorizado puede acceder a los datos.
selección bidireccional: una instrucción de selección que es capaz de evaluar sólo dos alternativas. En C, la instrucción if-else. Contraste con la selección de múltiples caminos.
selección multidireccional: una instrucción de selección que es capaz de evaluar más de dos alternativas. En C, la instrucción switch. Contraste con selección de dos caminos.
señal (elemento sintáctico): un constructo sintáctico que representa una operación, una bandera o una pieza de datos.
servidor: en un sistema cliente-servidor, la computadora centralizada que proporciona servicios auxiliares (programas servidor).
set: en máscaras, una técnica para hacer un bit 1.
simulación de cola de espera: una actividad de modelado utilizada para generar estadísticas sobre el rendimiento de una cola de espera.
sin rechazo: una calidad de un mensaje recibido que no permite al emisor negar lo enviado.
sincronización de procesos: un mecanismo del sistema operativo que controla el acceso a un recurso por más de un proceso.
sinónimo: una lista hashed, dos o más llaves que se dirigen a la misma dirección base.
sintaxis: las reglas "gramaticales" para un lenguaje. En C, el conjunto de palabras clave y reglas de formato que deben seguirse cuando se escribe un programa.
sistema binario: un sistema de numeración que utiliza dos símbolos (0 y 1).
sistema de administración de base de datos (DBMS: database management system): un programa o una serie de programas que manipula una base de datos.
sistema de bases de datos relacionales (RDBMS: relational database management system): una serie de programas que maneja las relaciones en un modelo de bases de datos relacional.

sistema decimal: un método de representación numérica que utiliza 10 símbolos (de 0 a 9).
sistema distribuido: un sistema operativo que controla los recursos localizados en las computadoras en sitios diferentes.
sistema Excess: un método de representación numérica utilizado para almacenar el valor exponencial de una fracción.
sistema operativo monousuario: un sistema operativo en el cual sólo un programa puede estar en la memoria a la vez.
sistema operativo por lotes: el sistema operativo utilizado en las primeras computadoras, en el cual las tareas se agrupaban antes de atenderlas.
sistema operativo: el software que controla el entorno de computación y proporciona una interfaz para el usuario.
sistema paralelo: un sistema operativo con varios CPU en la misma máquina.
sobredesbordamiento: un evento que ocurre cuando se hace un intento para eliminar datos de una estructura de datos vacía.
software: los programas de aplicación y del sistema necesarios para que el hardware de computadora logre realizar una tarea.
sonda: en un algoritmo de hashing, el cálculo de una dirección y su prueba satisfactoria; en un algoritmo de búsqueda, una iteración del ciclo que incluye la prueba para el argumento de búsqueda.
SQL: Véase lenguaje de consultas estructurado.
subalgoritmo: una parte de un algortimo que está escrita independientemente. Se ejecuta cuando se llama dentro del algortimo.
subárbol: cualquier estructura conectada debajo de la raíz de un árbol
subcadena: una parte de una cadena.
subprograma: Véase subalgoritmo.
subrutina: *Véase subalgoritmo.*
subscript: un número ordinal que indica la posición de un elemento dentro de un arreglo. *Véase también índice.*
subsistema de entrada/salida: la parte de la organización de la computadora que recibe datos desde el exterior y los envía al exterior.
suma: suma de una serie de números.

T

tabla de enrutamiento: la tabla utilizada por un enrutador para enrutar un paquete.
tabla de verdad: una tabla que lista todas las combinaciones lógicas posibles con la salida lógica correspondiente.
TDA: *Véase tipo de datos abstracto.*
teclado: un dispositivo de entrada que consiste de teclas alfanuméricas y teclas de función utilizado para texto o datos de control.
TELNET (Terminal Network): un programa cliente-servidor de propósito general que permite inicio de sesión remota.
texto: datos almacenados como caracteres.
texto cifrado: los datos cifrados.
texto simple (plano): el texto antes de ser cifrado.
tiempo compartido: un concepto del sistema operativo en el cual más de un usuario tiene acceso a una computadora al mismo tiempo.
tiempo de búsqueda: en el acceso a disco, el tiempo requerido para mover la cabeza de lectura/escritura sobre la pista donde están los datos.
tiempo de transferencia: el tiempo para mover datos desde el disco al CPU/memoria.
tipo: un conjunto de valores y una serie de operaciones que pueden aplicarse a esos valores.
tipo de datos abstracto (TDA): una declaración de datos empacada junto con operaciones que se aplican a los tipos de datos.
tipo de datos: un conjunto nombrado de valores y operaciones definidas para manipularlos, tales como carácter y entero.
tipo derivado: un tipo de datos compuesto construido a partir de otros tipos (arreglo, estructura, unión, apuntador y tipo enumerado).
topología: la estructura de una red, incluyendo el arreglo físico de dispositivos.
topología de anillo: una topología en la cual los dispositivos se conectan en un anillo; cada dispositivo recibe una unidad de datos de un vecino y la envía al otro vecino.
topología de bus: una topología de red en la cual todas las computadoras se conectan a un medio compartido.
topología de estrella: una topología en la cual todas las computadoras se conectan a un concentrador común.
topología de malla: una topología en la cual cada dispositivo se conecta a todos los otros dispositivos.
topología híbrida: una topología formada por más de una topología básica.
traductor: un término genérico para cualquiera de los programas de conversión de lenguajes. Vea también ensamblador y compilador.
trailer: información de control agregada a una unidad de datos.
trama: una unidad de datos en el nivel de enlace de datos.
transferencia de datos: mover datos de una computadora a otra.
transformación discreta de coseno (DCT): una transformación matemática utilizada en JPEG.
tupla: en una base de datos relacional, un registro (línea) en una relación.

U

último en entrar, primero en salir (LIFO: last in, first out): un algoritmo en el cual el último elemento de dato que se añadió a una lista se elimina de la lista en primer lugar.
UML (lenguaje de modelado unificado): una herramienta utilizada para diseñar en las ciencias de la computadión y los negocios.
Unicode: un código de 65 536 caracteres que incluye los símbolos de la mayoría de los idiomas del mundo.
unidad central de procesamiento (CPU): la parte de una computadora que contiene los componentes de control para interpretar instrucciones. En una computadora personal, un microchip que contiene una unidad de control y unidad lógica aritmética.
unidad de control: el componente de un CPU que interpreta las instrucciones y controla el flujo de datos.
unidad de traslación: en C, un archivo de compilación temporal utilizado para almacenar el código fuente modificado.
unidad lógica aritmética (ALU: arithmetic logic unit): la parte de un sistema de computadora que realiza las operaciones aritméticas y lógicas con los datos.
UNIX: un popular sistema operativo entre los programadores de computadoras y los científicos de la computación.
unset: *Véase forzar a 0.*
URL (localizador uniforme de recursos): una cadena de caracteres que define una página en Internet.
usuario final: la entidad que utiliza el producto final.

V

valor AC: el valor que cambia con el tiempo.
valor DC: el valor que no cambia con el tiempo.
variable: un objeto de almacenamiento de memoria cuyo valor puede cambiarse durante la ejecución de un programa. Contraste con constante.
velocidad de transmisión: un número de bits enviado por segundo.
velocidad rotacional: la velocidad de giro de un disco magnético.
vértice: un nodo en un gráfico.
vértices adyacentes: dos vértices en un grafo que están conectados por una línea.
video: una representación de imágenes (llamadas cuadro) en el tiempo.

W

WAN: Véase red de área amplia.
Web: Véase World Wode Web.
World Wide Web (WWW): un servicio Internet multimedia que permite a los usuarios recorrer Internet al moverse de un documento a otro por medio de vínculos.

Z

zona: en un disco óptico, un área no tocado por el láser en la traducción de un patrón de bits; por lo general representa un bit.

Índice