COMO PROGRAMAR EN C/C++

SEGUNDA EDICION

H.M. DEITEL / P.J. DEITEL

PRENTICE HALL

COMO PROGRAMAR EN C/C++

SEGUNDA EDICIÓN

H. M. Deitel
Universidad Nova
Deitel y asociados

P. J. Deitel
Deitel y asociados

TRADUCCION:
Gabriel Sánchez García
Ingeniero Mecánico Electricista
UNAM

REVISION TECNICA:
Raymundo Hugo Rangel
Prof. Facultad de Ingeniería
UNAM

PRENTICE HALL

MÉXICO • NUEVA YORK • BOGOTÁ • LONDRES • MADRID
MUNICH • NUEVA DELHI • PARÍS • RÍO DE JANEIRO
SINGAPUR • SYDNEY • TOKIO • TORONTO • ZURICH

EDICION ESPAÑOL:

SUPERVISOR DE TRADUCCION: JOAQUIN RAMOS SANTALLA
SUPERVISION PRODUCCION: JULIAN ESCAMILLA LIQUIDANO

EDICION EN INGLES:

Acquisitions Editor: Marcia Horton
Production Editor: Joe Scordato
Copy Editor: Michael Schiaparelli
Chapter Opener and Cover Designer: Jeannette Jacobs
Buyer: Dave Dickey
Supplements Editor: Alice Dworkin
Editorial Assistant: Dolores Mars

DEITEL: COMO PROGRAMAR EN C/C++ 2/ED

Traducido el inglés de la obra: C How to Programm 2/Ed

Calle 4 Nº 25-2º piso Fracc. Ind. Alce Blanco,
Naucalpan de Juárez, Edo. de México,
C.P. 53370

Miembro de la Cámara Nacional de la Industria Editorial, Reg. Núm. 1524

ISBN 0-13-226119-7

IMPRESO EN MEXICO/PRINTED IN MEXICO

SEP

PROGRAMAS EDUCATIVOS, S. A. DE C.V.
CALZ. CHABACANO No. 65, LOCAL A
COL. ASTURIAS,DELEG. CUAUHTEMOC,
C.P. 06850, MÉXICO, D.F.

EMPRESA CERTIFICADA POR EL
INSTITUTO MEXICANO DE NORMALIZACIÓN
Y CERTIFICACIÓN A.C., BAJO LA NORMA
ISO-9002: 1994/NMX-CC-004: 1995
CON EL No. DE REGISTRO RSC-048

1996

5000

PARA

El Dr. Stephen Feldman, Presidente de la Universidad Nova y el Dr. Edward Lieblein, Rector del Centro para las Ciencias de la Computación y la Información, de la Universidad Nova.

Por su visión en relación a una institución científica para la educación avanzada, las ciencias de la computación y la investigación en el sur de Florida, y por sus incesantes esfuerzos por cristalizar esa visión en esta joven universidad.

H. M. D.

PARA

Mis profesores en Lawrenceville y el M.I.T.

Por infundir en mí el amor hacia el aprendizaje y por escribir.

P.J.D.

Contenido

Ilustraciones

Prefacio

¡Bienvenido a C! Este libro ha sido escrito por un viejo y por un joven. El viejo (HMD; Massachusetts Institute of Technology 1967) ha estado programando y/o enseñando programación por más de 30 años. El joven (PJD; MIT 1991) ha estado programando desde hace una docena de años y se ha infectado del "virus" de la enseñanza y de la escritura. El viejo programa enseña a partir de la experiencia. El joven programa parte de una reserva inagotable de energía. Al viejo le gusta la claridad. Al joven le encanta la espectacularidad. El viejo aprecia la elegancia y la belleza. El joven desea resultados. Nos reunimos los dos para producir un libro que esperamos resulte entre informativo, interesante y entretenido.

En la mayor parte de los entornos educativos, C se enseña a personas que ya saben cómo programar. Muchos educadores piensan que la complejidad de C y cierto número de otras dificultades, incapacitan a C para un primer curso sobre programación —precisamente el curso meta de este libro. por tanto, ¿por qué es que escribimos este texto?

C de hecho se ha convertido en el lenguaje de elección para la implantación de sistemas en la industria, existen buenas razones para creer que su variante orientada a objetos, C++, resultará el lenguaje dominante de los años finales de los 90. Durante trece años Harvey Deitel ha estado enseñando Pascal en entornos universitarios, con énfasis en el desarrollo de programas claramente escritos y bien estructurados. Mucho de lo que se enseña en una secuencia de cursos introductorios de Pascal, forman los principios básicos de la programación estructurada. Hemos presentado este material exactamente de la misma forma en que HMD ha llevado a cabo sus cursos en la universidad. Se presentan algunos escollos, pero cuando éstos ocurren, los hacemos notar y explicamos los procedimientos para manejarlos con eficacia. Nuestra experiencia ha sido en que los alumnos manejan el curso de forma aproximada igual a como manejan Pascal. Existe una diferencia notable: sin embargo, los alumnos resultan muy motivados por el hecho de que están aprendiendo un lenguaje que les será de inmediata utilidad en cuanto dejen el entorno universitario. Esto incrementa su entusiasmo para este material —una gran ayuda cuando usted considera que C es en realidad más difícil de aprender.

Nuestra meta estaba clara: producir un libro de texto de programación en C para cursos a nivel universitario de introducción a la programación de computadoras, para estudiantes con poca o ninguna experiencia en programación,

pero también producir un libro que ofreciera el tratamiento riguroso de la teoría y de la práctica que se requiere en los cursos tradicionales de C. Para alcanzar estas metas, tuvimos que producir un libro más extenso que otros textos de C esto se debe al hecho de que nuestro texto enseña también de forma paciente los principios de programación estructurada. Aproximadamente mil alumnos han estudiado este material en nuestros cursos. Y en todo el mundo, decenas de millares de estudiantes han aprendido a programar en C, partiendo de la primera edición de este libro.

El libro contiene una gran recopilación de ejemplos, ejercicios y proyectos que se tomaron de muchos campos, a fin de proporcionar al estudiante la oportunidad de resolver problemas interesantes del mundo real.

El libro se concentra en principios de buena ingeniería de software y hace hincapié en la claridad de la programación, mediante el uso de la metodología de la programación estructurada. Evitamos el uso de terminología y de especificaciones de sintaxis antiguas, favoreciendo el enseñar mediante el ejemplo.

Entre los dispositivos pedagógicos de este texto, se encuentran programas y resultados ilustrativos completos, para demostrar los conceptos; un conjunto de objetivos y una sinopsis al principio de cada uno de los capítulos; los errores comunes de programación y las prácticas sanas de programación que se enumeran a todo lo largo de cada uno de los capítulos y se resumen al final de los mismos; secciones de resumen y de terminología en cada capítulo; ejercicios de autoevaluación y sus respuestas en cada capítulo; y la recopilación más completa de ejercicios existente en ningún libro de C. Los ejercicios van desde simples preguntas recordatorio a problemas extensos de programación y a proyectos de importancia. Aquellos instructores que requieran sustanciales proyectos de fin de cursos para sus alumnos, encontrarán muchos problemas apropiados enlistados en los ejercicios correspondientes a los capítulos 5 hasta el 21. Hemos puesto un gran esfuerzo en la recopilación de los ejercicios para aumentar el valor de este curso para el alumno. Los programas en el texto se probaron en compiladores que cumplen con normas ANSI C en SPARCstations de Sun, en Macintosh (Think C) de Apple, y en la PC de IBM (Turbo C, Turbo C++, y Borland C++), y en el VAX/VMS de DEC (VAX C).

Este texto se apega al estándar ANSI C. Muchas características de ANSI C no funcionarán con versiones anteriores a ANSI C. Vea los manuales de consulta correspondientes a su sistema particular en relación con mayores detalles con respecto al lenguaje, u obtenga una copia de ANSI/ISO 9899: 1990, "American National Standard for Information Systems Programming Language C", que puede obtener del American National Standards Institute, 11 West 42nd Street, New York, New York 10036.

Acerca del libro

En este libro abundan características que contribuirán al aprendizaje de los alumnos.

Objetivos

Cada capítulo se inicia con un enunciado de objetivos. Esto le indica al alumno qué es lo que puede esperar, y le da una oportunidad, una vez que haya leído el capítulo, de determinar si él o ella han cumplido con estos objetivos. Ello genera confianza y es fuente de refuerzo positivo.

Citas

A los objetivos de enseñanza les siguen una serie de citas. Algunas son humorísticas, otras filosóficas y varias ofrecen pensamientos interesantes. Nuestros alumnos han expresado que disfrutaron relacionando tales citas con el material de cada capítulo.

Sinopsis

La sinopsis del capítulo ayuda al alumno a enfrentarse al material en un orden de descendente. Esto, también ayuda a los estudiantes a anticipar qué es lo que va a seguir y a establecer un ritmo coherente.

Secciones

Cada capítulo está organizado en pequeñas secciones que se ocupan de áreas clave. Las características de C se presentan en el contexto de programas completos que funcionan. Cada programa esta acompañado por una ventana que contiene el resultado que se obtiene al ejecutar el programa. Esto permite al estudiante confirmar que los programas se ejecutan como se espera. La relación de los resultados con los enunciados de los programas que los generan, es una forma excelente de aprender y reforzar los conceptos. Nuestros programas están diseñados para ejercitar las diversas características de C. La lectura cuidadosa del texto es muy similar a la introducción y ejecución de estos programas en computadora.

Ilustraciones

Se incluyen numerosos dibujos y tablas. El análisis de la diagramación estructurada de flujo, ayuda a los estudiantes a apreciar el uso de estructuras de control y de programación estructurada, incluye diagramas de flujo cuidadosamente dibujados. El capítulo sobre estructuras de datos utiliza muchos dibujos de línea para ilustrar la creación y el mantenimiento de estructuras importantes de datos, como son listas enlazadas, colas de espera, pilas y árboles binarios.

Elementos de diseño útiles

Hemos incluido cuatro elementos de diseño para ayudar a los estudiantes a concentrarse en aspectos importantes del desarrollo de programas, su prueba y depuración, su rendimiento y portabilidad. Resaltamos varios de éstos en la forma de sanas prácticas de programación, errores comunes de programación, sugerencias de rendimiento, sugerencias de portabilidad, y observaciones de ingeniería de software.

Prácticas sanas de programación

Las prácticas sanas de programación se resaltan en el texto. Al estudiante le llama la atención las técnicas que ayudan a producir mejores programas. Estas prácticas representan lo mejor que hemos podido recabar de cuatro décadas combinadas de experiencia en programación.

Errores comunes de programación

Los estudiantes que están aprendiendo un lenguaje en especial en su primer curso de programación tienden a cometer ciertos errores que son comunes. Enfocar la atención de los estudiantes a estos errores comunes de programación resulta un gran auxilio. ¡También ayuda a reducir largas filas en la parte exterior de las oficinas de los instructores durante horas hábiles!

Sugerencias de rendimiento

Encontramos que, por mucho, la meta más importante de un primer curso de programación, es escribir programas claros y comprensibles. Pero los estudiantes desean escribir programas que se ejecuten de forma más rápida, utilicen el mínimo de memoria, requieran un mínimo de tecleo, o que brillen en alguna otra forma elegante. A los estudiantes les interesa en verdad el rendimiento.

Quieren saber qué es lo que pueden hacer para "turbocargar" su programas; por tanto, hemos incluido sugerencias de rendimiento para resaltar las oportunidades que mejoran el rendimiento de los programas.

Sugerencias de portabilidad

El desarrollo de software es una actividad compleja y muy costosa. Las organizaciones que desarrollan software a menudo deben producir versiones personalizadas para una variedad de computadoras y de sistemas operativos. Por tanto, hoy día se hace gran énfasis en la portabilidad, es decir, en la producción de software que podrá ser ejecutado sin cambio en muchos sistemas de computación diferentes. Muchas personas presumen que C es el mejor lenguaje para el desarrollo de software portable. Algunas personas suponen que si implantan una aplicación en C, esta última, de forma automática, resultará portátil. Esto no es cierto. Conseguir portabilidad, requiere de un diseño cuidadoso y cauteloso. Existen muchos escollos. En sí mismo, el documento de ANSI Standard C enlista 11 páginas de dificultades potenciales. Nosotros incluimos numerosas sugerencias de portabilidad. Hemos combinado nuestra propia experiencia en la elaboración de software portable con un estudio cuidadoso de la sección estándar ANSI relativa a portabilidad, así como de dos excelentes libros que tratan sobre la portabilidad (véase la referencia Ja89 y Ra90 al final del capítulo 1).

Observaciones de ingeniería de software

Este elemento de diseño es nuevo en esta segunda edición. Hemos resumido muchas observaciones que afectan la arquitectura y la construcción de los sistemas de software, en especial en sistemas a gran escala.

Resumen

Cada uno de nuestros capítulos termina con una cantidad de dispositivos pedagógicos adicionales. Presentamos un resumen detallado del capítulo en forma de lista con viñetas. Esto auxilia a los estudiantes a revisar y reforzar conceptos clave. Entonces recopilamos y enlistamos, en orden, todas las prácticas sanas de programación, los errores comunes de programación, las sugerencias de rendimientos, las sugerencias de portabilidad y las observaciones de ingeniería de software que aparecen en el capítulo.

Terminología

Incluimos una sección de terminología con una lista alfabética de términos importantes que se definen a lo largo del capítulo. Otra vez, se trata de una confirmación redundante. A continuación resumimos las prácticas sanas de programación, los errores comunes de programación, las sugerencias de rendimiento, las sugerencias de portabilidad y las observaciones de ingeniería de software.

Ejercicios de autoevaluación

Se incluyen, para autoestudio, gran cantidad de ejercicios de autoevaluación con sus respuestas completas. Esto le da la oportunidad al estudiante de obtener confianza con el material y prepararse para intentar resolver los ejercicios regulares.

Ejercicios

Cada capítulo concluye con un conjunto sustancial de ejercicios que van desde el simple recordatorio de terminología y conceptos importantes, hasta escribir enunciados individuales en

C, a escribir pequeñas porciones de funciones en C, a escribir funciones y programas completos en C, e inclusive proyectos importantes de fin de curso. El gran número de ejercicios le permite a los instructores ajustar sus cursos a las necesidades únicas de su auditorio y variar las asignaciones del curso cada semestre. Los instructores pueden utilizar estos ejercicios para formar asignaciones para trabajo en casa, exámenes cortos y de importancia.

Un recorrido por el libro

El libro está dividido en tres partes principales. La primera parte, los capítulos 1 hasta el 14, presenta un tratamiento completo del lenguaje de programación C, incluye una introducción formal a la programación estructurada. La segunda parte única en libros de texto de C, los capítulos 15 hasta el 21, presentan un tratamiento sustancial de la programación de C++ orientada a objetos, suficiente para un curso universitario de pregraduación de alto nivel. La tercera parte, los apéndices A hasta E, presentan una variedad de materiales de consulta y referencia en apoyo al texto principal.

El capítulo 1, "Introducción", analiza qué son las computadoras, cómo funcionan, y cómo se programan. Introduce la idea de la programación estructurada y explica por qué ese conjunto de técnicas ha acelerado una revolución en la forma de escribir programas. El capítulo presenta una historia breve del desarrollo de los lenguajes de programación a partir de los de máquina, pasando por los lenguajes ensambladores hasta los de alto nivel. Se presenta el origen del lenguaje de programación C. El capítulo incluye una introducción al entorno de programación de C.

El capítulo 2, "Introducción a la programación de C", da una introducción concisa a la escritura de programas C. Se presenta un tratamiento detallado de la toma de decisiones y de operaciones aritméticas en C. Después de estudiar este capítulo, el estudiante sabrá cómo escribir programas simples, pero completos, de C.

En el capítulo 3, "Programación estructurada", es probable que sea el capítulo más importante del texto, en especial para el estudiante serio de la ciencia de la computación. Introduce la noción de los algoritmos (procedimientos) para la resolución de problemas. Explica la importancia que tiene la programación estructurada para la producción de programas que sean comprensibles, depurables, mantenibles y que es probable funcionen de forma correcta a partir del primer intento. Introduce las estructuras de control fundamental de la programación estructurada, es decir, la secuencia, la selección (if y if/else), y las estructuras de repetición (while). Explica la técnica de refinamiento descendente paso a paso, que es crítica a la producción de programas estructurados en forma correcta. Presenta la ayuda popular del diseño de programas, el seudocódigo estructurado. Los métodos y planteamientos que se utilizan en el capítulo 3 son aplicables a la programación estructurada de cualquier lenguaje de programación, y no solo de C. Este capítulo ayuda a desarrollar en el estudiante buenos hábitos de programación, preparándolo para el manejo de tareas más sustanciales de programación en el resto del texto.

El capítulo 4, "Control de programa", refina las nociones de la programación estructurada y nos presenta estructuras adicionales de control. Examina en detalle la repetición, y compara las alternativas de ciclos controlados por contador con ciclos controlados por centinela. Se presenta la estructura `por` como un medio conveniente para implantar ciclos controlados por contador. La estructura de selección **`switch`** y la estructura de repetición **`do/while`** también se presentan. El capítulo concluye con un análisis de operadores lógicos.

En el capítulo 5, "Funciones", se analiza el diseño y la construcción de módulos de programa. C incluye funciones de biblioteca estándar, funciones definidas por el programador, recursión y capacidades de llamadas por valor. Las técnicas que se presentan en el capítulo 5 son esenciales

a la producción y apreciación de programas correctamente estructurados, en especial aquellos grandes programas de software que los programadores de sistemas y los programadores de aplicaciones quizá tendrán que desarrollar en aplicaciones de la vida real. La estrategia de "divide y vencerás" se introduce como un medio eficaz para la resolución de problemas complejos; las funciones permiten al programador la división de programas complejos en componentes más sencillos interactuantes. Los estudiantes que disfrutan del tratamiento de los números y la simulación aleatoria, apreciarán el análisis del juego de dados, que hace un uso elegante de las estructuras de control. El capítulo ofrece una introducción sólida a la recursión e incluye una tabla resumiendo los 31 ejemplos y ejercicios de recursión que aparecen distribuidos a todo lo largo del libro. Algunos libros dejan la recursión para un postrer capítulo; pero sentimos que este tema se cubre mejor en forma gradual a todo lo largo del texto. La recopilación extensiva de 39 ejercicios al final del capítulo 5, incluye varios problemas clásicos de recursión, como las torres de Hanoi.

El capítulo 6, "Arreglos", analiza la estructuración de datos en arreglos, o grupos, de elementos de datos relacionados del mismo tipo. El capítulo presenta numerosos ejemplos tanto de arreglos de un solo subíndice como de arreglos con doble subíndice. Es muy importante estructurar los datos así como usar estructuras de control en el desarrollo de programas correctamente estructurados. Ejemplos en el capítulo investigan varias manipulaciones comunes de arreglos, la impresión de histogramas, la clasificación y ordenamiento de datos, el pasar arreglos a funciones, y una introducción al campo del análisis de datos de encuestas. Una característica de este capítulo es la presentación cuidadosa de la búsqueda binaria como una mejora dramática en comparación con la búsqueda lineal. Los ejercicios al fin del capítulo, incluyen una selección en especial grande de problemas interesantes y retadores. Estos incluyen técnicas de clasificación mejoradas, el diseño de un sistema de reservaciones de aerolínea, una introducción al concepto de los gráficos tipo tortuga (mismo que se hizo famoso en el lenguaje LOGO), y los problemas de la torre del caballero y de las ocho reinas, que introducen las ideas de la programación heurística, tan ampliamente empleada en el campo de la inteligencia artificial.

En el capítulo 7, "Apuntadores", se presenta una de las características más poderosas del lenguaje C. El capítulo proporciona explicaciones detalladas de los operadores de apuntador, a los que, por referencia, se les da el nombre de expresiones de apuntador, aritméticas de apuntador, de la relación entre apuntadores y arreglos, arreglos de apuntadores y apuntadores a funciones. Los ejercicios del capítulo incluyen una simulación de la carrera clásica entre la tortuga y la liebre, y los algoritmos de barajar y repartir naipes. Una sección especial titulada "Cómo construir su propia computadora" también está incluida. Esta sección explica la noción de la programación del lenguaje máquina y sigue con un proyecto que incluye el diseño y la implantación de un simulador de computadora, que permite al lector escribir y ejecutar programas en lenguaje máquina. Esta característica única del texto será en especial útil al lector que desee comprender cómo en realidad funcionan las computadoras. Nuestros alumnos disfrutan de este proyecto y a menudo implantan mejorías sustanciales; muchas de ellas están sugeridas dentro de los ejercicios. En el capítulo 12, otra sección especial guía al lector a lo largo de la elaboración o construcción de un compilador. El lenguaje máquina producido por el compilador después se ejecuta en el simulador del lenguaje máquina que se produce en el capítulo 7.

El capítulo 8, "Caracteres y cadenas", se ocupa de los fundamentos del procesamiento de datos no numéricos. El capítulo incluye un recorrido completo de las funciones de procesamiento de caracteres y de cadenas, disponibles en las bibliotecas de C. Las técnicas que aquí se analizan son de amplia utilidad en la construcción de procesadores de texto, en software de disposiciones de páginas y tipografía, y en aplicaciones de procesamiento de palabras. El capítulo incluye una recopilación interesante de 33 ejercicios que exploran aplicaciones de procesamiento de texto. El

estudiante disfrutará los ejercicios en la escritura de quintillas humorísticas, poesía aleatoria, conversión del inglés al latín vulgar, generación de palabras de siete letras que sean equivalentes a un número telefónico dado, la justificación texto, la protección de cheques, escribir una cantidad de un cheque en palabras, generación de clave Morse, conversiones métricas y de cartas reclamando deudas. ¡El último ejercicio reta al estudiante a utilizar un diccionario computarizado para crear un generador de crucigramas!

El capítulo 9, "Entrada/salida con formato", presenta todas las capacidades poderosas de formato de printf y scanf. Analizamos las capacidades de formato de salida de **`printf`** como son el redondeo de los valores de punto flotante a un número dado de decimales, la alineación de columnas de números, justificación a la derecha y a la izquierda, inserción de información literal, el forzado del signo más, impresión de ceros a la izquierda, el uso de notación exponencial, el uso de números octales y hexadecimales, y el control de los anchos y precisiones de campo. Se analizan todas las secuencias de escape de **`printf`** en relación con el movimiento del cursor, al imprimir caracteres especiales y para generar una alarma audible. Examinamos todas las capacidades de formato de entrada de **`scanf`**, incluyendo la entrada de tipos específicos de datos y el pasar por alto caracteres específicos del flujo de entrada. Se analizan todos los especificadores de conversión de **`scanf`** para la lectura de valores decimales, octales, hexadecimales, de punto flotante, de carácter y de cadenas. Analizamos las entradas, para que coincidan (o no coincidan) con los caracteres de un conjunto. Los ejercicios del capítulo prueban de forma virtual todas las capacidades de formato de entrada/salida de C.

El capítulo 10, "Estructuras, uniones, manipulaciones de bits y enumeraciones", presenta una variedad de características de importancia. Las estructuras son como los registros de Pascal y de otros lenguajes —agrupan elementos de datos de varios tipos. Las estructuras se utilizan en el capítulo 11 para formar archivos compuestos de registros de información. Las estructuras se utilizan en conjunción con los apuntadores y la asignación dinámica de memoria del capítulo 12, para formar estructuras dinámicas de datos como son listas enlazadas, colas de espera, pilas y árboles. Las uniones permiten que se utilice un área en memoria para diferentes tipos de datos en diferentes momentos; esta capacidad de compartir puede reducir los requerimientos de memoria de un programa o los requerimientos de almacenamiento secundario. Las enumeraciones proporcionan una forma conveniente de definición de constantes simbólicas útiles; esto ayuda a que los programas sean más autodocumentales. La poderosa capacidad de manipulación de bits posibilita a los programadores que escriban propagandas que hagan uso del hardware a bajo del nivel. Esto ayuda a que los programas procesen cadenas de bits, ajusten bits individuales en **`on`** o en **`off`**, y almacenen información de una forma más compacta. Estas que a menudo sólo aparecen en lenguajes ensambladores de bajo nivel, son valiosas para los programadores que escriben software de sistema, como son sistemas operativos y software de redes. Una característica del capítulo es su simulación revisada de alto rendimiento de barajar y repartir naipes. Esta es una excelente oportunidad para que el instructor haga énfasis en la calidad de los algoritmos.

El capítulo 11, "Procesamiento de archivos", analiza las técnicas que se utilizan para procesar archivos de texto con acceso secuencial y aleatorio. El capítulo se inicia con una introducción a la jerarquía de datos desde bits, pasando por bytes, campos, registros y hasta archivos. A continuación, se presenta una vista simple de archivos y flujos. Se analizan los archivos de acceso secuencial utilizando una serie de tres programas que muestran cómo abrir y cerrar archivos, cómo almacenar datos en un archivo en forma secuencial, y cómo leer datos en forma secuencial de un archivo. Los archivos de acceso aleatorio se tratan utilizando una serie de cuatro programas que muestran cómo crear de forma secuencial un archivo para acceso aleatorio, cómo leer y escribir datos a un archivo con acceso aleatorio, y cómo leer datos en forma secuencial de un archivo con

acceso aleatorio. El cuarto programa de acceso aleatorio combina muchas de las técnicas de acceso a archivos tanto secuencial como aleatorio formando un programa completo de proceso de transacciones. Los estudiantes en nuestros seminarios de industria nos han indicado después de estudiar el material relativo a procesamiento de archivos, que fueron capaces de producir programas sustanciales de procesamiento de archivo que resultaron de utilidad inmediata para sus organizaciones.

El capítulo 12, "Estructuras de datos", analiza las técnicas que se utilizan para crear estructuras dinámicas de datos. El capítulo se inicia con un análisis de estructuras autorreferenciadas y asignación dinámica de memoria. El capítulo continúa con un análisis de cómo crear y mantener varias estructuras dinámicas de datos incluyendo listas enlazadas, colas de espera (o líneas de espera), pilas y árboles. Para cada uno de los tipos de estructuras de datos, presentamos programas completos y funcionales mostrando salidas de muestra. El capítulo 12 ayuda al estudiante a dominar de verdad los apuntadores. El capítulo incluye ejemplos abundantes, utilizando indirección y doble indirección un concepto en particular difícil. Un problema al trabajar con apuntadores, es que los estudiantes tienen dificultad para visualizar las estructuras de datos y cómo se enlazan juntos sus nodos. Por tanto, hemos incluido ilustraciones para mostrar los enlaces, y la secuencia en la cual se crean. El ejemplo del árbol binario es una piedra angular soberbia para el estudio de apuntadores y de estructura dinámica de datos. Este ejemplo crea un árbol binario; obliga a eliminación duplicada; y nos inicia en recorridos recursivos de árboles en preorden, en orden y en posorden. Los estudiantes tienen un *verdadero sentido de realización* cuando estudian e implantan este ejemplo. Estiman en forma muy particular el ver que el recorrido en orden imprime los valores de los nodos en orden clasificado. El capítulo incluye una recopilación importante de ejercicios. Una parte a destacar del capítulo es la sección especial "Cómo construir su propio compilador". Los ejercicios guían al estudiante a través del desarrollo de un programa de conversión de infijos a posfijos y un programa de evaluación de expresión de posfijos. Después modificamos el algoritmo de evaluación de posfijos para generar código en lenguaje de máquina. El compilador coloca este código en un archivo (utilizando las técnicas del capítulo 11). ¡Después los estudiantes ejecutan el lenguaje de máquina producido por sus compiladores en los simuladores de software que construyeron en los ejercicios del capítulo 7!

El capítulo 13, "El preprocesador", proporciona análisis detallados de las directrices del preprocesador. El capítulo proporciona información más completa en la directriz **#incluye** que hace que se incluya una copia del archivo especificado en lugar de la directriz antes de la compilación del archivo, y de la directriz **#define** que crea constantes simbólicas y macros. El capítulo explica la compilación condicional, para permitirle al programador el control de la ejecución de las directrices del preprocesador, y la compilación del código del programa. El operador # que convierte su operando en una cadena y el operador ## que concatena dos señales también son analizados. Las cinco constantes simbólicas predefinidas (**__LINE__**, **__FILE__**, **__DATE__**, **__TIME__**, y **__STDC__**) son presentadas. Por último, se estudia el macro **assert** de la cabecera **assert.h** es valioso en la prueba, depuración, verificación y validación de programas.

El capítulo 14, "Temas avanzados", presenta varios temas avanzados que de forma ordinaria no son cubiertos en cursos de introducción. La sección 14.2 muestra cómo redirigir salida a un programa proveniente de un archivo, cómo redirigir salida de un programa para colocarse en un archivo, cómo redirigir la salida de un programa para resultar la entrada de otro programa (entubado) y agregar la salida de un programa a un archivo existente. La sección 14.3 analiza como desarrollar funciones que utilizan listas de argumentos de longitud variable. La sección 14.4 muestra como pueden ser pasados argumentos de la línea de comandos para la función **main** y

utilizados en un programa. La sección 14.5 analiza la compilación de programas cuyos componentes están dispersos en varios archivos. La sección 14.6 estudia el registro de funciones utilizando **atexit** para ser ejecutados en la terminación de un programa, terminar la ejecución de un programa con la función **exit**. La sección 14.7 estudia los calificadores de tipo **const** y **volátiles**. La sección 14.8 muestra cómo especificar el tipo de una constante numérica utilizando los sufijos de entero y de punto flotante. La sección 14.9 explica archivos binarios y el uso de archivos temporales. La sección 14.10 muestra cómo utilizar la biblioteca de manejo de señales para atrapar eventos no esperados. La sección 14.11 analiza la creación y utilización de arreglos dinámicos utilizando **calloc** y **realloc**.

En la primera edición de este texto, se incluyó una introducción de un capítulo a C++ y a la programación orientada a objetos. Durante este tiempo, muchas universidades han decidido incorporar una introducción a C++ y la programación orientada a objetos en sus cursos C. Por lo cual en esta edición, hemos expandido este tratamiento a siete capítulos con suficiente texto, ejercicios y laboratorios para un curso de un semestre.

El capítulo 15, "C++ como un C mejor", introduce las características no orientadas a objetos de C++. Estas características mejoran el proceso de escritura de programas convencionales orientados a procedimientos. El capítulo discute comentarios de una sola línea, flujo de entrada/salida, declaraciones, cómo crear nuevos tipos de datos, prototipos de función y verificación de tipos, funciones en línea (como una sustitución de los macros), parámetros de referencia, el calificador const, asignación dinámica de memoria, argumentos por omisión, el operador de resolución de ámbito unario, la homonimia de funciones, las especificaciones de vinculación y las plantillas de funciones.

El capítulo 16, "Clases y abstracción de datos", representa una maravillosa oportunidad para la enseñanza de la abstracción de datos de la "manera correcta" mediante un lenguaje (C++) dedicado de forma expresa a la implantación de tipos de datos abstractos (ADT). En años recientes, la abstracción de datos se ha convertido en un tema de importancia en los cursos de computación introductorios que se enseñan en Pascal. Conforme estamos escribiendo este libro, tomamos en consideración la presentación de la abstracción de datos en C, pero decidimos que en vez de ello se incluiría esta introducción detallada a C++. Los capítulos 16, 17 y 18, incluyen una presentación sólida de la abstracción de datos. El capítulo 16 analiza la implantación de ADT como *structs*, implantando ADT como clases de estilo C++, el acceso a miembros de clase, la separación de la interfaz de la implantación, el uso de funciones de acceso y de funciones de utilería, la inicialización de objeto mediante constructores, la destrucción de objetos mediante destructores, la asignación por omisión de copia a nivel de miembro, y la reutilización del software.

El capítulo 17, "Clases Parte II", continúa con el estudio de las clases y la abstracción de datos. El capítulo analiza la declaración y el uso de los objetos constantes, de las funciones miembro constantes, de la composición —el proceso de elaboración de clases que tienen otras clases como miembros, funciones amigos y clases amigos que tienen derechos de acceso especiales a los miembros privados de clases, el apuntador **this** permite que un objeto conozca su propia dirección, la asignación dinámica de memoria, los miembros de clase estáticos para contener y manipular datos de todo el ámbito de la clase, ejemplos de tipos de datos abstractos populares (arreglos, cadenas y colas de espera), clases contenedoras, iteradores, y clases plantilla. Las clases plantilla están entre las adiciones más recientes al lenguaje en evolución de C++. Las clases plantilla permiten al programador capturar la esencia de un tipo de datos abstracto (como una pila, un arreglo o una fija de espera) y a continuación crear, con la inclusión mínima de código adicional, versiones de esta ADP para tipos particulares (como una pila de int, una pila de float,

una fija de espera de int, etcétera). Por esta razón, las clases plantilla a menudo se conocen como tipos parametrizados.

El capítulo 18, "Homonimia de operadores", es uno de los temas más populares en los cursos de C++. Los alumnos en realidad disfrutan de este material. Encuentran que encajan de forma perfecta con el análisis de tipos de dato abstractos de los capítulos 16 y 17. La homonimia de operadores permite al programador indicarle al compilador cómo utilizar operadores existentes con objetos de nuevos tipos. C++ ya sabe cómo utilizar estos operadores con objetos de tipos incorporados, como son los enteros, puntos flotantes y caracteres. Pero suponga que creamos una nueva clase de cadena. ¿Qué es lo que significa el signo más? Muchos programadores utilizan el signo más con cadenas para significar concatenación. En este capítulo, el programador aprenderá cómo demostrar el "homónimo" del signo más de tal forma, que cuando este escrito entre dos objetos cadena en una expresión, el compilador generará una llamada de función a una "función operador" que concatenará las dos cadenas. El capítulo estudia los fundamentos de la homonimia de operadores, las restricciones en la homonimia de operadores, la homonimia con funciones miembros de clase en comparación con funciones no miembros, la homonimia de operadores unarios y binarios, y la conversión entre tipos. Una característica del capítulo es la gran cantidad de casos de estudio de importancia, es decir una clase de arreglo, una clase de cadena, una clase de fecha, una clase grande de entero, y una clase de números complejos (los dos últimos aparecen con todo el código fuente en los ejercicios).

El capítulo 19, "Herencia", trata con una de las capacidades fundamentales de los lenguajes de programación orientadas a objetos. La herencia es una forma de reutilización del software en la cual se pueden desarrollar de forma rápida clases nuevas al absorber las capacidades de clases existentes y a continuación añadiendo capacidades nuevas apropiadas. El capítulo discute las nociones de clases base y clases derivadas, miembros protegidos, herencia pública, herencia protegida, herencia privada, clases base directas, clases base indirectas, utilización de constructores y destructores en clases base y derivadas, ingeniería de software con herencia. El capítulo compara la herencia (relaciones "es una") con composición (relaciones "tiene una") e introduce relaciones "utiliza una" y "conoce una". Una característica del capítulo es la inclusión de varios estudios de caso de importancia. En particular, un estudio de caso extenso implanta una jerarquía de clase punto, círculo y cilindro. El capítulo concluye con un estudio de caso de herencia múltiple una característica avanzada de C++ en el cual la clase derivada puede ser formada al heredar atributos y comportamientos de varias clases base.

El capítulo 20, "Funciones virtuales y polimorfismo", trata con otra de las capacidades fundamentales de la programación orientada a objetos, es decir comportamiento polimorfo. Muchas clases están relacionadas mediante la herencia a una clase base común, cada objeto de clase derivada, puede ser tratado como un objeto de clase base. Esto permite a los programas que sean escritos de una forma bastante general independiente de los tipos específicos de objetos de clase derivada. Se pueden manejar nuevos tipos de objetos mediante el mismo programa, haciendo por tanto los sistemas más extensibles. El polimorfismo permite a los programas eliminar lógica compleja de intercambio en favor de una lógica más sencilla "de línea recta". Un administrador de video para un juego de video, por ejemplo, puede simplemente enviar un mensaje de dibujar a todos los objetos de una lista enlazada de objetos a ser dibujados. Cada objeto sabe como dibujarse a sí mismo. Se puede añadir al programa un nuevo objeto sin modificar el programa siempre y cuando dicho objeto también sepa como dibujarse a sí mismo. Este estilo de programación se utiliza de forma típicamente para implantar interfaces gráficas de usuario tan populares hoy en día. El capítulo discute la mecánica de la obtención del comportamiento polimórfico, mediante el uso de funciones virtuales. El capítulo hace la distinción entre clases abstractas (a

partir de las cuales no se puede producir ningún objeto) y clases concretas (a partir de las cuales se pueden producir objetos). Las clases abstractas son útiles para proporcionar una interfaz capaz de heredarse a las clases a todo lo largo de la jerarquía. Una característica del capítulo son sus dos estudios de casos polimórficos de importancia un sistema de nóminas y otra versión de la jerarquía de forma de puntos, círculo y cilindro que fue estudiada en el capítulo 19.

El capítulo 21, "C++ flujo de entrada/salida", contiene un tratamiento en extremo detallado del nuevo estilo orientado a objetos de entrada/salida introducido en C++. Se enseñan o se dan muchos cursos de C utilizando compiladores C++, y los instructores a menudo prefieren enseñar el estilo nuevo de C++ correspondiente a E/S antes de continuar utilizando el estilo anterior o más viejo de **`printf/scanf`**. El capítulo analiza las varias capacidades de E/S de C++ que incluye salida utilizando el operador de inserción de flujo, entrada con el operador de extracción de flujo, E/S de tipo seguro (una agradable mejoría sobre C), E/S con formato, E/S sin formato (para un mayor rendimiento), manipuladores de flujo para controlar la base del flujo (decimal, octal o hexadecimal), números de punto flotante, cómo controlar los anchos de campo, manipuladores definidos por usuario, estados de formato de flujo, estados de error de flujo, E/S de objetos de tipos definido por usuario, y cómo ligar flujos de salida con flujos de entrada (para asegurar que en realidad aparecen indicadores antes que el usuario introduzca respuestas).

Varios apéndices proporcionan material valioso de consulta y referencia. En particular, presentamos en el apéndice A un resumen sintáctico de C; en el apéndice B aparece un resumen de todas las funciones de biblioteca estándares de C, con explicaciones; en el apéndice C una gráfica completa de precedencia y asociatividad de operadores; en el apéndice D el conjunto de códigos de caracteres ASCII; y en el apéndice E un análisis de los sistemas numéricos binarios, octal, decimal y hexadecimal. El apéndice B fue condensado del documento estándar ANSI, con el permiso expreso por escrito del American National Standards Institute; este apéndice resulta una referencia detallada y valiosa para el programador practicante de C. El apéndice E es una guía didáctica completa sobre sistemas numéricos incluyendo muchos ejercicios de autoevaluación y sus respuestas.

Reconocimientos

Uno de los grandes placeres de escribir un libro de texto consiste en reconocer los esfuerzos de muchas personas cuyos nombres pudieran no aparecer en las portadas, pero sin cuyo trabajo, cooperación, amistad y comprensión, la elaboración de este texto hubiera resultado imposible.

HMD desea agradecer a sus colegas de la Universidad Nova Ed Simco, Clovis Tondo, Ed Lieblein, Phil Adams, Raisa Szabo, Raúl Salazar y Bárbara Edge.

Nos gustaría agradecer a nuestros amigos de Digital Equipment Corporation (Stephanie Stosur Schwartz, Sue-Lane Garrett, Janet Hebert, Faye Napert, Betsy Mills, Jennie Connolly, Bárbara Couturier y Paul Sandore), de la Sun Microsystems (Gary Morin), de la Corporation for Open Systems International (Bill Horst, David Litwack, Steve Hudson, y Linc Faurer), Informative Stages (Don Hall), Semaphore Training (Clieve Lee), y Cambridge Technology Partners (Gart Davis, Paul Sherman, y Wilberto Martínez), así como a numerosos clientes corporativos que han hecho de la enseñanza de este material en un entorno industrial una experiencia placentera.

Hemos sido afortunados de haber tenido la posibilidad de trabajar en este proyecto con un equipo talentoso y dedicado de profesionales de la publicación en Prentice Hall. Joe Scordato hizo un magnífico trabajo como editor de producción. Dolores Mars coordinó el esfuerzo complejo de revisión del manuscrito y siempre resultó de increíble ayuda cuando necesitamos asistencia. Su entusiasmo y buena disposición *son* sinceramente apreciados.

Este libro se concibió debido al estímulo, entusiasmo y persistencia de Marcia Horton, Editor en jefe. Resulta un gran crédito para Prentice Hall que sus funcionarios más importantes continúen en sus responsabilidades editoriales. Siempre nos ha impactado con ello y estamos agradecidos de ser capaces de continuar trabajando de cerca con Marcia, inclusive ante su aumento de responsabilidades administrativas.

Apreciamos los esfuerzos de nuestros revisores de la primera y segunda edición (sus afiliaciones en el momento de la revisión se enlistan entre paréntesis).

David Falconer (Universidad Estatal de California en Fullerton)
David Finkel (Worcester Polytechnic)
H. E. Dunsmore (Universidad de Purdue)
Jim Schmolze (Universidad de Tufts)
Gene Spafford (Universidad de Purdue)
Clovis Tondo (Corporación IBM y profesor visitante en la Universidad Nova)
Jeffrey Esakov (Universidad de Pensilvania)
Tom Slezak (Universidad de California, Lawrence Livermore National Laboratory)
Gary A. Wilson (Gary A. Wilson & Associates y Universidad de California, Extensión Berkeley)
Mike Kogan (Corporación IBM; Principal arquitecto de OS/2 2.0 de 32 bits)
Don Kostuch (retirado Corporación IBM; ahora instructor mundial en C, C++ y programación orientada a objetos)
Ed Lieblein (Universidad Nova)
John Carroll (Universidad Estatal de San Diego)
Alan Filipski (Universidad Estatal de Arizona)
Greg Hidley (Universidad Estatal de San Diego)
Daniel Hirschberg (Universidad de California en Irvine)
Jack Tan (Universidad de Houston)
Richard Alpert (Universidad de Boston)
Eric Bloom (Universidad Bentley College)

Estas personas escudriñaron todos los aspectos del texto e hicieron docenas de valiosas sugerencias para mejorar la exactitud y totalidad de la presentación.

Debemos una especial nota de agradecimiento al Dr. Graem Ringwood, Computer Science Dept., QMW Universidad de Londres. El doctor Ringwood envió continuamente sugerencias constructivas mientras impartía sus cursos basándose en nuestro libro. Sus comentarios y críticas jugaron un papel importante en la conformación de la segunda edición.

Tem Nieto contribuyó con largas horas de esfuerzo ayudándonos con la sección especial "Cómo construir su propio compilador" del final del capítulo 12.

También nos vemos en la obligación de agradecer a los muchos profesores, instructores, estudiantes y profesionales que enviaron sus comentarios sobre la primera edición: MacRae, Joe, Sysop del foro Autodesk AutoCad de CompuServe; McCarthy, Michael J., Director de Undergraduate Programs, Universidad de Pittsburgh; Mahmoud Fath El-Den, Departmento de Matemáticas y Ciencias de la Computación, Universidad Estatal Ft. Hays; Rader, Cyndi, Universidad Estatal Wright; Soni, Manish, Universidad de Tufts(estudiante); Bullock, Tom, Departamento de Ingeniería Eléctrica, Universidad de Florida en Gainesville (Professor of EE); Derruks, Jan, Hogeschool van Amsterdam, Technische Maritieme Faculteit, Amsterdam; Duchan, Alan, Chair, MIS Department, Escuela de Negocios Richard J. Wehle, Universidad Canisius; Kenny, Bárbara

T., Departamento de Matemáticas, Universidad Estatal de Boise; Riegelhaupt-Herzig, Scott P., Colegio Metropolitano de la Universidad de Boston, Departamento de Ciencias de la Computación; Yean, Leong Wai, Universidad Tecnológica de Nanyang, División de Tecnología de Ciencias Aplicadas, Singapur (estudiante); Abdullah, Rosni, Universidad Sains Malaysia, Departamento de Ciencias de la Computación; Willis, Bob: Cohoon, Jim, Departamento de Ciencias de la Computación, Universidad de Virginia; Tranchant, Mark, Universidad de Southhampton, Gran Bretaña (estudiante); Martignoni, Stephane, Instituto Real de Tecnología, Suecia (estudiante); Spears, Marlene, Homebrewer (fabrica cerveza)(su esposo que es estudiante utiliza nuestro libro); French, Rev. Michael D.(SJ), Departamento de Ciencias de la Computación, Universidad Loyola, Maryland; Wallace, Ted, Departamento de Ruso y Física, Universidad Dartmouth (estudiante); Wright, Kevin, Universidad de Nebraska (estudiante); Elder, Scott, (estudiante); Schneller, Jeffrey; Byrd, William, Department de Ingeniería Industrial y de Sistemas, Universidad de Florida (estudiante); Naiman, E. J., Compuware; Sedgwick, Arthur E., Dr., Departamento de Matemáticas, Estadística y Ciencias de la Computación, Universidad Dalhouside, Halifax, Nueva Escocia (usuario); Holsberg, Peter J., Profesor, Tecnología de Ingeniería, Computación y Matemáticas, Universidad Comunitaria del Condado de Mecer; Pont, Michel J., DR., Lecturer, Departamento de Ingeniería, Universidad de Leicester, Inglaterra; Linney, John, Profesor Asistente, Departamento of Ciencias de la Computación, Universidad Queen Mary and Westfield, Londres; Zipper, Herbert J., Profesor, Departamento de Ingeniería, SUNY Farmingdale; Humenik, Keith, Universidad de Maryland, Campus, Baltimore; Beeson, Michael, Departamento de Matemáticas y Ciencias de la Computación, Universidad Estatal de San José; Gingo, Peter J., Dr., Departamento de Ciencias Matemáticas, Universidad Buchtel de Artes y Ciencias; y Vaught, Lloyd, Departamento de Ciencias de la Computación, Universidad Modesto Junior.

Los autores desean hacer extensivo su especial agradecimiento a Ed Lieblein, una de las principales autoridades en el mundo sobre ingeniería de software, por su extraordinaria revisión sobre partes del material relativo a C++ y a programación orientada a objetos. El Dr. Lieblein es amigo y colega de HMD en la Universidad Nova en Fort Lauderdale, Florida, donde trabaja como profesor de tiempo completo en ciencias de la computación. El Dr. Lieblein fue Director técnico de Tartan Laboratories, una de las organizaciones líder en el desarrollo de compiladores del mundo. Antes, ocupó el cargo de Director de Computer Software and Systems en la oficina del Secretario de Defensa de Estados Unidos. Como tal, administró la iniciativa de Software DoD, un programa especial para mejorar la capacidad de software de la nación en relación con sistemas futuros de misión crítica. Inició el programa STARS del Pentágono en relación con tecnología de software y reutilización, guió el programa Ada hacia la estandarización internacional, y desempeñó un papel importante en el establecimiento del Software Engineering Institute en la Universidad Carnegie Mellon. Es en verdad un privilegio especial para nosotros el poder trabajar con el Dr. Lieblein en la Universidad Nova.

También deseamos extender una nota especial de agradecimiento al Sr. Dr. Clovis Tondo de IBM Corporation, Profesor visitante en la Universidad Nova. El Dr. Tondo fue el jefe de nuestro equipo de revisión. Sus revisiones meticulosas y completas nos enseñaron mucho en relación con las sutilezas de C y C++. El Dr. Tondo es coautor del *C Answer Book* que contiene respuestas a los ejercicios existentes en —que además se utiliza ampliamente en conjunción con —*The C Programming Language,* libro clásico que escribieron Brian Kernighan y Dennis Ritchie.

Este texto se basa en la versión de C estandarizada a través de American National Standards Institute (ANSI) en los Estados Unidos y a través del International Standards Organization (ISO) a nivel mundial. Hemos utilizado de forma extensiva materiales del documento estándar ANSI con el permiso expreso por escrito de la American National Standards Institute. Sinceramente,

apreciamos la cooperación de Mary Clare Lynch, Directora de publicaciones de ANSI, para ayudarnos a obtener los permisos de publicación necesarios. Las figuras 5.6, 8.1, 8.5, 8.12, 8.17, 8.20, 8.22, 8.30, 8.36, 9.1, 9.3, 9.6, 9.9, 9.16, 10.7 y 11.6, y el apéndice A: Sintaxis de C, y el apéndice B: Biblioteca estándar, se condensaron y adaptaron con el permiso del American National Standard for Information Systems—Programming Language C, ANSI/ISO 9899: 1990. Se pueden adquirir copias de este estándar o norma del American National Standards Institute en 11 West 42nd Street, New York, NY 10036.

Por último, nos gustaría agradecerle a Bárbara y Abbey Deitel, por su cariño, comprensión y sus enormes esfuerzos en ayudarnos a preparar el manuscrito. Aportaron innumerables horas de esfuerzo; probaron todos los programas del texto, auxiliaron en todas las fases de preparación del manuscrito e hicieron revisión de estilo de todos los borradores del texto, hasta su publicación. Su revisión minuciosa impidió que se cometieran innumerables errores. Bárbara también hizo la investigación de las citas, y Abbey sugirió el título para el libro.

Asumimos completa responsabilidad por cualquiera de los errores que hayan quedado en este texto. Agradeceríamos sus comentarios, críticas, correcciones y sugerencias para su mejoría. Por favor envíe su sugerencias para mejorar y añadir a nuestra lista de prácticas sanas de programación, errores comunes de programación, sugerencias de rendimiento, sugerencias de portabilidad y observaciones de ingeniería de software. Reconoceremos a todos los que contribuyan en la siguiente emisión de nuestro libro. Por favor dirija toda su correspondencia a nuestra dirección electrónica:

```
deitel@world.std.com
```

o si no escríbanos a la siguiente dirección:

Harvey M. Deitel (autor)
Paul J. Deitel (autor)
c/o Computer Science Editor
College Book Editorial
Prentice Hall
Englewood Cliffs, New Jersey 07632

Responderemos de inmediato.

Harvey M. Deitel
Paul J. Dietel

COMO PROGRAMAR EN C/C++

1

Conceptos de computación

Objetivos

- Comprender los conceptos básicos de computación.
- Familiarizarse con diferentes tipos de lenguajes de programación.
- Familiarizarse con la historia del lenguaje de programación C.
- Concientizarse de la biblioteca estándar C.
- Comprender el entorno de desarrollo del programa C.
- Apreciar porqué es apropiado aprender C en un primer curso de programación.
- Apreciar porqué C proporciona una base para subsiguientes estudios de programación en general y de C++ en particular.

Las cosas están siempre mejor en su principio.
Blaise Pascal

Pensamientos elevados deben tener un lenguaje elevado.
Aristophanes

Nuestra vida se malgasta a causa de detalles ... simplifique, simplifique.
Henry Thoreau

Sinopsis

1.1 Introducción

¡Bienvenido a C! Hemos trabajado duro para crear lo que esperamos sinceramente resulte para usted una experiencia educativa, informativa y entretenida. C es un lenguaje difícil, que normalmente se enseña sólo a programadores experimentados, por lo que este libro resulta único entre los libros de texto de C:

- Es apropiado para personas técnicamente orientadas, con poca o ninguna experiencia de programación.
- Es apropiado para programadores experimentados, que deseen un tratamiento profundo y riguroso del lenguaje.

¿Cómo puede un solo libro ser atracativo para ambos grupos? La respuesta estriba en que el núcleo común del libro hace énfasis en la obtención de *claridad* en los programas, mediante técnicas probadas de "programación estructurada". Los que no son programadores aprenderán programación de la forma "correcta" desde el principio. Hemos intentado escribir de una forma clara y sencilla. El libro está profusamente ilustrado. Y quizá de forma más importante, es que el libro presenta un número sustancial de programas de C operantes y muestra las salidas producidas al ejecutar estos programas en una computadora.

Los primeros cuatro capítulos presentan los fundamentos de la computación, de la programación de computadoras y del lenguaje C de programación de computadoras. Estos análisis están enmarcados en una introducción a la programación de computadoras utilizando el método estructurado. Los principiantes que han asistido a nuestros cursos nos indican que el material de estos capítulos representa una base sólida para el tratamiento más profundo de C de los capítulos 5 hasta el capítulo 14. Los programadores experimentados, por lo general, leen rápido los primeros cuatro capítulos y entonces descubren que el tratamiento de C de los capítulos 5 al 14 es tan riguroso como estimulante. Aprecian en forma particular el detallado tratamiento de apuntadores, cadenas, archivos y estructuras de datos que aparecen en los capítulos siguientes.

Muchos programadores experimentados nos han dicho que encuentran útil el tratamiento que le damos a la programación estructurada. Han estado a menudo programando en un lenguaje estructurado como Pascal, pero debido a que nunca fueron formalmente iniciados en la programación estructurada, no están escribiendo el mejor código posible. Conforme aprenden C utilizando este libro, se hacen cada vez más capaces de mejorar su estilo de programación. Por tanto , ya sea que sea usted un neófito o un programador experimentado, tenemos aquí mucho para informarle, entretenerlo y estimularlo.

La mayor parte de las personas están familiarizadas con las cosas excitantes que hacen las computadoras. En este curso, aprenderá cómo ordenarle a las computadoras que las hagan. Es el *software* (es decir, las instrucciones que usted escribe para ordenarle a la computadora a que ejecute acciones y a que tome decisiones) quien controla a las computadoras (a menudo conocido como *hardware*), y uno de los lenguajes de desarrollo de software más populares hoy día es C. Este texto proporciona una introducción a la programación en ANSI C, la versión estandarizada en 1989 tanto en los Estados Unidos, a través del American National Standards Institute (ANSI), como en el resto del mundo, a través de la International Standards Organization (ISO).

El uso de computadoras se está incrementando prácticamente en todos los campos de actividad. En una era de costos siempre crecientes, los costos de computación han venido reduciéndose de forma sorprendente, debido a increíbles desarrollos, tanto en la tecnología de hardware como de software. Las computadoras, que hace 25 años podían llenar grandes habitaciones y costaban millones de dólares, pueden ser ahora inscritas en las superficies de chips de silicón de un tamaño menor que una uña, y que quizá cuestan unos cuantos dólares cada uno. De forma irónica, el silicón es uno de los materiales más abundantes sobre la tierra —es un ingrediente de la arena común. La tecnología de los chips de silicón ha convertido a la computación en algo tan económico, que en el mundo se utilizan aproximadamente 150 millones de computadoras de uso general, auxiliando a las personas en los negocios, la industria, el gobierno y en sus vidas personales. Y este número podría con facilidad duplicarse en unos pocos años.

¿Puede C ser enseñado en un primer curso de programación, lo que se supone que es el auditorio para este libro? Así lo pensamos. Hace dos años tomamos este reto, cuando Pascal era el lenguaje que dominaba los primeros cursos de la ciencia de computación. Escribimos *Cómo programar en C*, primera edición de este texto. Cientos de universidades en todo el mundo han utilizado *Cómo programar en C*. Los cursos basados en ese libro han comprobado ser de igual eficacia que sus predecesores, basados en Pascal. No se han observado diferencias significativas, a excepción quizá que sus alumnos están más motivados, porque saben que tienen más probabilidades de utilizar C en vez de Pascal en sus cursos de niveles superiores, así como en sus carreras. Los alumnos aprendiendo C también saben que estarán mejor preparados para aprender de forma rápida C++. C++ es un super conjunto del lenguaje C, orientado a programadores que desean escribir programas orientados a objetos. Diremos más en relación con C++ en la Sección 1.14. •

De hecho, C++ es motivo de tanto interés hoy día, que en el capítulo 15 hemos decidido incluir una introducción detallada a C++ y a la programación orientada a objetos. Un fenómeno interesante, que está ocurriendo en el mercado de los lenguajes de programación, es que muchos de los proveedores clave, ahora sólo ponen en el mercado un producto combinado C/C++, en vez de ofrecer productos por separado. Esto le da a los usuarios la capacidad de continuar programando en C si así lo desean, y cuando lo consideren apropiado emigrar de forma gradual hacia C++.

Entonces, ¡ahí lo tiene! Está a punto de iniciar un camino estimulante y esperamos, lleno de satisfacciones. Conforme vaya avanzando, si desea comunicarse con nosotros, envíenos correo electrónico por Internet a `deitel@world.std.com`. Haremos toda clase de esfuerzos para responder de forma rápida. ¡Buena suerte!

1.2 ¿Qué es una computadora?

Una *computadora* es un dispositivo capaz de ejecutar cálculos y tomar decisiones lógicas a velocidades millones y a veces miles de millones de veces más rápidas de lo que pueden hacerlo los seres humanos. Por ejemplo, muchas de las computadoras personales de hoy día, pueden ejecutar decenas de millones de adiciones por segundo. Una persona utilizando una calculadora de escritorio pudiera requerir décadas para completar el mismo número de cálculos de lo que puede ejecutar una computadora personal poderosa en sólo un segundo. (Puntos a considerar: ¿cómo sabría si la persona sumó correctamente las cifras?, ¿cómo sabría si la computadora sumó correctamente las cifras?) Hoy día las *supercomputadoras* más rápidas pueden ejecutar cientos de miles de millones de sumas por segundo —¡aproximadamente tantos cálculos podrían ejecutar cientos de miles de personas en un año! y en los laboratorios de investigación ya están en funcionamiento computadoras de trillones de instrucciones por segundo.

Las computadoras procesan *datos* bajo el control de un conjunto de instrucciones que se conocen como *programas de computación*. Estos programas de computación guían a la computadora a través de conjuntos ordenados de acciones, especificados por personas a las que se conoce como *programadores de computadora*.

Los varios dispositivos (como el teclado, la pantalla, los discos, la memoria y las unidades procesadoras) que conforman un sistema de computación se conocen como el *hardware*. Los programas de computación que se ejecutan en una computadora se conocen como el *software*. Los costos del hardware han venido reduciéndose de forma drástica en años recientes, hasta el punto que las computadoras personales se han convertido en mercancía. Desafortunadamente, los costos de desarrollo de software han ido creciendo continuamente, conforme los programadores cada día desarrollan aplicaciones más poderosas y complejas, sin la capacidad de hacer mejorías paralelas en la tecnología del desarrollo del software. En este libro aprenderá métodos de desarrollo de software, que pueden reducir de forma sustancial los costos de desarrollo y acelerar el proceso de desarrollo de aplicaciones de software poderosas y de alta calidad. Estos métodos incluyen la *programación estructurada, la refinación por pasos de arriba a abajo, la funcionalización* y en el último capítulo del libro, *la programación orientada a objetos.*

1.3 Organización de la computadora

Si no se toman en cuenta las diferencias en apariencia física, virtualmente todas las computadoras pueden ser concebidas como divididas en seis *unidades lógicas* o secciones. Estas son:

1. *Unidad de entrada.* Esta es la sección "de recepción" de la computadora. Obtiene información (datos y programas de computadora) a partir de varios *dispositivos de entrada* y pone esta información a la disposición de las otras unidades, de tal forma que la información pueda ser procesada. La mayor parte de la información se introduce en las computadoras hoy día a través de teclados de tipo máquina de escribir.
2. *Unidad de salida.* Esta es la sección "de embarques" de la computadora. Toma la información que ha sido procesada por la computadora y la coloca en varios *dispositivos de salida* para dejar la información disponible para su uso fuera de la computadora. La mayor parte de la información sale de las computadoras hoy día mediante despliegue en pantallas o mediante impresión en papel.
3. *Unidad de memoria.* Esta es la sección de "almacén" de rápido acceso y de capacidad relativamente baja de la computadora. Retiene información que ha sido introducida a través de la unidad de entrada, de tal forma que esta información pueda estar de inmediato disponible para su proceso cuando sea necesario. La unidad de memoria también retiene información ya procesada, hasta que dicha información pueda ser colocada por la unidad de salida en dispositivos de salida. La unidad de memoria se conoce a menudo como *memoria o memoria primaria.*
4. *Unidad aritmética y lógica (ALU).* Esta es la sección de "fabricación" de la computadora. Es responsable de la ejecución de cálculos como es suma, resta, multiplicación y división. Contiene los mecanismos de decisión que permiten que la computadora, por ejemplo, compare dos elementos existentes de la unidad de memoria para determinar si son o no iguales.
5. *Unidad de procesamiento central (CPU).* Esta es la sección "administrativa" de la computadora. Es el coordinador de la computadora que es responsable de la supervisión de la operación de las demás secciones. El CPU le indica a la unidad de entrada cuándo debe leerse la información y colocarse en la unidad de memoria, le indica al ALU cuándo deberá utilizar información de la unidad de memoria en cálculos, y le indica a la unidad de salida cuándo enviar información de la unidad de memoria a ciertos dispositivos de salida.
6. *Unidad de almacenamiento secundario.* Esta es la sección de "almacén" a largo plazo de alta capacidad de la computadora. Los programas o los datos que no se estén utilizando de forma activa por otras unidades, están por lo regular colocados en dispositivos de almacenamiento secundario (como discos) en tanto se necesiten otra vez, es posible que sean horas, días, meses o inclusive años después.

1.4 Procesamiento por lotes, multiprogramación y tiempo compartido

Las primeras computadoras sólo eran capaces de ejecutar un *trabajo* o *tarea* a la vez. Esta forma de operación de las computadoras, a menudo se conoce como *procesamiento por lotes* de un solo usuario. La computadora ejecuta un programa a la vez al procesar datos en grupos o en *lotes*. En estos sistemas primarios, los usuarios por lo general entregaban sus trabajos al centro de cómputo en paquetes de tarjetas perforadas. Los usuarios a menudo tenían que esperar horas e inclusive días, antes que se les devolvieran impresiones a sus escritorios.

Conforme las computadoras se hicieron más poderosas, se hizo evidente que el procesamiento por lotes de un solo usuario rara vez utilizaba los recursos de la computadora de manera eficazmente. En vez de ello, se pensó que muchos trabajos o tareas podían hacer que *compartieran* los recursos de la computadora para obtener mejor utilización. Esto se conoce como *multiprogramación.* La multiprogramación implica la operación "simultánea" de muchos trabajos en una computadora —la computadora comparte sus recursos entre los trabajos que compiten por su

atención. En el caso de los primeros sistemas de multiprogramación, los usuarios aún entregaban los trabajos en paquetes de tarjetas perforadas, y tenían que esperar horas o días para los resultados.

En los años 60, varios grupos en la industria y en las universidades se hicieron pioneros en el concepto de *tiempo compartido*. El tiempo compartido es un caso especial de la multiprogramación, en el cual los usuarios tienen acceso a la computadora a través de dispositivos de entrada/salida o *terminales*. En un sistema típico de computadora a tiempo compartido, pudieran existir docenas e inclusive cientos de usuarios, compartiendo a la vez la computadora. La computadora de hecho, no ejecuta las órdenes de todos los usuarios en forma simultánea. En vez de ello, ejecuta una pequeña porción del trabajo de un usuario, y de inmediato pasa a darle servicio al siguiente. La computadora hace esto tan aprisa, que puede darle servicio a cada usuario varias veces por segundo. Por tanto , los usuarios *parece* que estuvieran ejecutando su trabajo de forma simultánea.

1.5 Computación personal, computación distribuida y computación cliente/servidor

En 1977, Apple Computer popularizó el fenómeno de la *computación personal*. Al principio, era el sueño de los aficionados a la computación. Las computadoras se hicieron lo bastante económicas para que las personas las adquirieran para su uso personal o de negocios. En 1981, IBM, el fabricante más grande del mundo de computadoras, introdujo la computadora personal IBM. Literalmente de la noche a la mañana, la computación personal se legitimizó en negocios, industrias y organizaciones gubernamentales.

Pero estas computadoras eran unidades "independientes" —las personas hacían el trabajo en sus propias máquinas y a continuación transportaban discos de ida y vuelta para compartir la información. Aunque las primeras computadoras personales no eran lo bastante poderosas para compartirse entre varios usuarios, estas máquinas podían ser enlazadas juntas en redes de computación, a veces mediante líneas telefónicas y otras en redes de área local dentro de una organización. Esto condujo al fenómeno de la *computación distribuida*, en la cual la carga de trabajo de computación de una organización, en vez de ser ejecutada de manera estrictamente en alguna instalación central de cómputo, se distribuye sobre la red a los lugares donde en realidad se ejecuta el trabajo de la organización. Las computadoras personales eran lo bastante poderosas para manejar las necesidades de cómputo de usuarios individuales, así como para manejar las tareas básicas de comunicación, de pasar la información electrónicamente de ida y vuelta.

Hoy día las computadoras personales más potentes son tan poderosas como las máquinas de un millón de dólares de hace una década. Las máquinas de escritorio más poderosas que se conocen como *estaciones de trabajo* proporcionan a usuarios individuales enormes capacidades. La información puede compartirse, con facilidad a través de redes de cómputo, donde algunas computadoras denominadas *servidores de archivo* ofrecen un almacén común de programas y de datos, que pueden ser utilizados por computadoras *cliente* distribuidas a todo lo largo de la red, y de ahí el término de *computación cliente/servicio*. C y C++ se han convertido en los lenguajes de elección para escribir software para sistemas operativos, para redes de computación y para aplicaciones distribuidas cliente/servidor.

1.6 Lenguajes máquina, lenguajes ensambladores y lenguajes de alto nivel

Los programadores escriben instrucciones en diferentes lenguajes de programación, algunos comprensibles de forma directa por la computadora y otros que requieren pasos intermedios de *traducción*. Existen hoy día cientos de lenguajes de computadora. Estos pueden ser categorizados en tres tipos generales:

1. Lenguajes máquina
2. Lenguajes ensambladores
3. Lenguajes de alto nivel

Cualquier computadora sólo puede entender directamente su propio *lenguaje máquina*. El lenguaje máquina es el "lenguaje natural" de una computadora particular. Está relacionado íntimamente con el diseño del hardware de esa computadora. Los lenguajes máquina, por lo general consisten de cadenas de números (al final reducidos a unos y a ceros) que instruyen a las computadoras para que ejecuten sus operaciones más elementales, una a la vez. Los lenguajes máquina son *dependientes de la máquina*, es decir, un lenguaje máquina particular puede ser utilizado en sólo un tipo de computadora. Los lenguajes máquina son difíciles de manejar por los seres humanos, como puede verse en la siguiente sección de un programa de lenguaje máquina, que añade pago por tiempo extra a la nómina base y almacena el resultado en la nómina bruta.

```
+1300042774
+1400593419
+1200274027
```

Conforme las computadoras se hicieron más populares, se hizo aparente que la programación en lenguaje máquina era demasiado lenta y tediosa para la mayor parte de los programadores. En vez de utilizar las cadenas de números que las computadoras pueden entender de forma directa, los programadores empezaron a usar abreviaturas similares al inglés para representar las operaciones elementales de la computadora. Estas abreviaturas similares al inglés formaron la base de los *lenguajes ensambladores*. Se desarrollaron *programas de traducción* denominados *ensambladores* para convertir los programas de lenguaje ensamblador a lenguaje máquina a la velocidad de las computadoras. La sección siguiente de un programa de lenguaje ensamblador también añade el pago de horas extras a la nómina base y almacena el resultado en una nómina bruta, pero con mayor claridad que su equivalente en lenguaje máquina:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

La utilización de las computadoras aumentó con rapidez con la llegada de los lenguajes ensambladores, pero estos aún necesitaban de muchas instrucciones para llevar a cabo inclusive las tareas más sencillas. Para acelerar el proceso de programación, se desarrollaron *lenguajes de alto nivel*, en los cuales se podían escribir simples enunciados para poder llevar a cabo tareas sustanciales. Los programas de traducción que convierten los programas de lenguaje de alto nivel al lenguaje máquina se llaman *compiladores*. Los lenguajes de alto nivel le permiten a los programadores escribir instrucciones que parecen prácticamente como el inglés de todos los días y contiene notaciones matemáticas por lo común utilizadas. Un programa de nómina escrito en un lenguaje de alto nivel pudiera contener un enunciado como el siguiente:

```
grosspay = basepay + overTimePay
```

Es obvio que, los lenguajes de alto nivel son mucho más deseables desde el punto de vista del programador que los lenguajes máquina o los ensambladores. C y C++ son, de entre los lenguajes de alto nivel, los más poderosos y los más utilizados.

1.7 La historia de C

C evolucionó a partir de dos lenguajes previos, BCPL y B. BCPL fue desarrollado en 1967 por Martin Richards, como un lenguaje para escribir software y compiladores de sistemas operativos.

Ken Thompson modeló muchas características de su lenguaje B siguiendo sus contrapartidas en BCPL, y utilizó B en 1970 para crear versiones iniciales del sistema operativo UNIX en los Laboratorios Bell, sobre una computadora PDP-7 de DEC. Tanto BCPL como B eran lenguajes "sin tipo" cada elemento de datos ocupaba una palabra "en memoria" y quedaba a cargo del programador el tratar un elemento de datos como si se tratara de un número entero o de un número real.

El lenguaje C fue derivado del lenguaje B por Dennis Ritchie, de los Laboratorios Bell, y al inicio se implantó en 1972 en una computadora PDP-11 de DEC. C al inicio se hizo muy conocido como lenguaje de desarrollo del sistema operativo UNIX. Hoy día, virtualmente todos los sistemas principales están escritos en C y/o C++. A lo largo de las últimas dos décadas, C se ha hecho disponible para la mayor parte de las computadoras. C es independiente del hardware. Con un diseño cuidadoso, es posible escribir programas en C que sean *portátiles* hacia la mayor parte de las computadoras. C utiliza muchos de los conceptos importantes de BCPL y de B, además de añadir los tipos de datos y otras características poderosas.

Hacia finales de los 70, C había evolucionado a lo que hoy se conoce como C "tradicional". La publicación en 1978 del libro de Kernighan y de Ritchie, *The C Programming Language*, atrajo gran atención sobre este lenguaje. Esta publicación se convirtió en uno de los libros científicos de computadoras de más éxito de todos los tiempos.

La expansión rápida de C sobre varios tipos de computadoras (denominadas a veces *plataformas de hardware*) trajo consigo muchas variantes. Estas eran similares, pero a menudo no eran compatibles. Esto resultaba en un problema serio para los desarrolladores de programas, que necesitaban escribir códigos que pudieran funcionar en varias plataformas. Se hizo cada vez más evidente que era necesaria una versión estándar de C. En 1983, se creó el comité técnico X3J11, bajo el American National Standards Committee on Computers and Information Processing (X3), para "proporcionar una definición no ambigua e independiente de máquina del lenguaje". En 1989 el estándar o norma quedó aprobado. El documento se conoce como ANSI/ISO 9899: 1990. Se pueden ordenar copias de este documento del American National Standards Institute, cuya dirección se menciona en el prefacio de este texto. La segunda edición de Kernighan y Ritchie, que se publicó en 1988, refleja esta versión que se conoce como ANSI C, la cual ahora se utiliza en todo el mundo (Ke88).

Sugerencia de portabilidad 1.1

Dado que C es un lenguaje independiente del hardware y ampliamente disponible, las aplicaciones que están escritas en C pueden ejecutarse con poca o ninguna modificación en una amplia gama de sistemas distintos de cómputo.

1.8 La biblioteca estándar de C

Como aprenderá en el capítulo 5, los programas C consisten de módulos o piezas que se denominan *funciones*. Usted puede programar todas las funciones que necesita para formar un programa C, pero la mayor parte de los programadores de C aprovechan una gran recopilación de funciones existentes, que se conocen como la *Biblioteca estándar C*. Entonces, para aprender el "universo" C, de hecho existen dos partes. El primero es aprender el lenguaje C mismo, y el segundo es aprender como utilizar las funciones de la Biblioteca estándar C. A lo largo de este libro, analizarémos muchas de estas funciones. El apéndice B (condensado y adaptado a partir del documento estándar de ANSI C mismo) enumera todas las funciones disponibles en la biblioteca estándar C. El libro escrito por Plauger (Pl92) es de lectura obligatoria para aquellos programadores que necesitan un profundo conocimiento de las funciones de biblioteca, cómo implantarlas y utilizarlas para escribir código portátil.

Usted será estimulado en este curso a utilizar un método de *bloques constructivos* para la creación de programas. Evite volver a inventar la rueda. Utilice piezas existentes —esto se conoce como *reutilización del software* y como veremos en el capítulo 15, es piedra angular del campo en desarrollo de la programación orientada a objetos. Cuando esté programando en C, por lo regular utilizará los siguientes bloques constructivos:

- Funciones de la biblioteca estándar de C
- Funciones que debe crear usted mismo
- Funciones a su disposición, que crearan otras personas.

La ventaja de crear su propias funciones, es que sabrá con exactitud como funcionan. Estará en condición de examinar el código C. La desventaja es el esfuerzo y el tiempo que se gasta en el diseño y el desarrollo de nuevas funciones.

El uso de funciones existentes elimina el tener que volver a inventar la rueda. En el caso de las funciones estándar ANSI, usted sabe que están escritas con cuidado, y lo sabe porque está utilizando funciones que están disponibles en todas las implantaciones de ANSI C y, por lo mismo sus programas tendrán una mayor oportunidad de ser portátiles.

Sugerencia de rendimiento 1.1

El usar funciones de la biblioteca estándar ANSI, en vez de escribir sus propias versiones comparables, puede mejorar el rendimiento de los programas, porque estas funciones están escritas de forma cuidadosa para que se ejecuten con eficacia.

Sugerencia de portabilidad 1.2

El usar funciones de biblioteca estándar ANSI en vez de escribir sus propias versiones comparables, puede mejorar la portabilidad del programa porque estas funciones están incluidas en casi todas las implantaciones de ANSI C.

1.9 Otros lenguajes de alto nivel

Se han desarrollado cientos de lenguajes de alto nivel, pero sólo unos pocos han alcanzado una amplia aceptación. *FORTRAN* (FORmula TRANslator) fue desarrollado por IBM entre 1954 y 1957, para uso en aplicaciones científicas y de ingeniería, que requieran de complejos cálculos matemáticos. FORTRAN es aún muy utilizado.

COBOL (COmmon Business Oriented Language) fue desarrollado en 1959 por un grupo de fabricantes de computadoras y de usuarios industriales y de gobierno. COBOL se utiliza sobre todo en aplicaciones comerciales, que requieren manipulación precisa y eficiente de grandes cantidades de datos. Hoy día, más de la mitad del software de negocios se programa aún en COBOL. Mas de un millón de personas están empleadas como programadores de COBOL.

Pascal fue diseñado casi al mismo tiempo que C. Se concibió para uso académico. En relación con Pascal diremos más en la sección siguiente.

1.10 Programación estructurada

Durante los años 60, el desarrollo de software se encontró con severas dificultades. Por lo regular los programas de entrega del software se retrasaban, sus costos excedían en gran medida los presupuestos, y los productos terminados no eran confiables. Las personas empezaron a darse

cuenta que el desarrollo de software era una actividad mucho más compleja de lo que se habían imaginado. La actividad de investigación de los años 60 dió como resultado la evolución de la *programación estructurada* —un método disciplinado de escribir programas que sean claros, que se demuestre que son correctos y fáciles de modificar. En el capítulo 3 y capítulo 4 se da una visión general de los principios de la programación estructurada. El resto del texto analiza el desarrollo de los programas estructurados de C.

Uno de los resultados más tangibles de esta investigación, fue el desarrollo en 1971 hecho por el profesor Nicklaus Wirth del lenguaje de programación Pascal. Pascal, al que se le da ese nombre en honor a Blaise Pascal, matemático y filósofo del siglo XVII, fue diseñado para la enseñanza de la programación estructurada en entornos académicos, y se convirtió con rapidez en el lenguaje introductorio de programación de la mayor parte de las universidades. Por desgracia, el lenguaje carece de muchas características necesarias para hacerlo útil en aplicaciones comerciales, industriales y de gobierno, por lo que no ha sido muy aceptado en esos últimos ámbitos. Quizá la historia registre que la verdadera significación del Pascal fue su elección como base del lenguaje de programación *Ada*.

Ada fue desarrollado bajo el patrocinio del Departamento de Defensa de los Estados Unidos (DOD) durante los años 70 y principio de los 80. Se estaban utilizando cientos de lenguajes distintos para producir los sistemas masivos de software de comando y de control de DOD. DOD deseaba un solo lenguaje que pudiera llenar sus objetivos. Pascal fue seleccionado como base, pero el lenguaje final Ada, es muy distinto de Pascal. Este lenguaje se llamó así en honor a Lady Ada Lovelace, hija del poeta Lord Byron. A Lady Lovelace se le da por lo general el crédito de haber escrito el primer programa de computación del mundo a principios de 1800. Una capacidad importante de Ada se conoce como *multitareas*; esto permite a los programadores especificar qué actividades deben ocurrir en paralelo. Otros lenguajes muy utilizados de alto nivel que hemos analizado incluyendo C y C++ permiten al programador escribir programas que sólo ejecuten una actividad a la vez. Está pendiente ver si Ada cumple sus objetivos de producir un software confiable y reducir de forma sustancial los costos de desarrollo y mantenimiento del software.

1.11 Los fundamentos del entorno de C

Todos los sistemas C consisten, en general, de tres partes: el entorno, el lenguaje y la biblioteca estándar C. En el siguiente análisis se explica el entorno típico de desarrollo de C, que se muestra en la figura 1.1.

Los programas C casi siempre pasan a través de seis fases para su ejecución (figura 1.1). Estas fases son: *editar, preprocesar, compilar, enlazar, cargar* y *ejecutar*. Nos estamos concentrando en este momento en el sistema típico UNIX, basado en C. Si usted no está utilizando un sistema UNIX, refiérase a los manuales de su sistema, o pregunte a su instructor cómo llevar a cabo estas tareas en su entorno.

La primera fase consiste en editar un archivo. Esto se ejecuta con un *programa de edición*. El programador escribe un programa C utilizando el editor, y si es necesario hace correcciones. El programa a continuación se almacena en un dispositivo de almacenamiento secundario, como sería un disco. Los nombres de archivo de los programas C deben terminar con la extensión `.c` Dos editores muy utilizados en sistemas UNIX son `vi` y `emacs`. Los paquetes de software C/C++, como son Borland C++ para las PC de IBM y compatibles, y Symantec C++ para el Macintosh de Apple, tienen editores incorporados, que están integrados en el entorno de programación. Suponemos que el lector sabe cómo editar un programa.

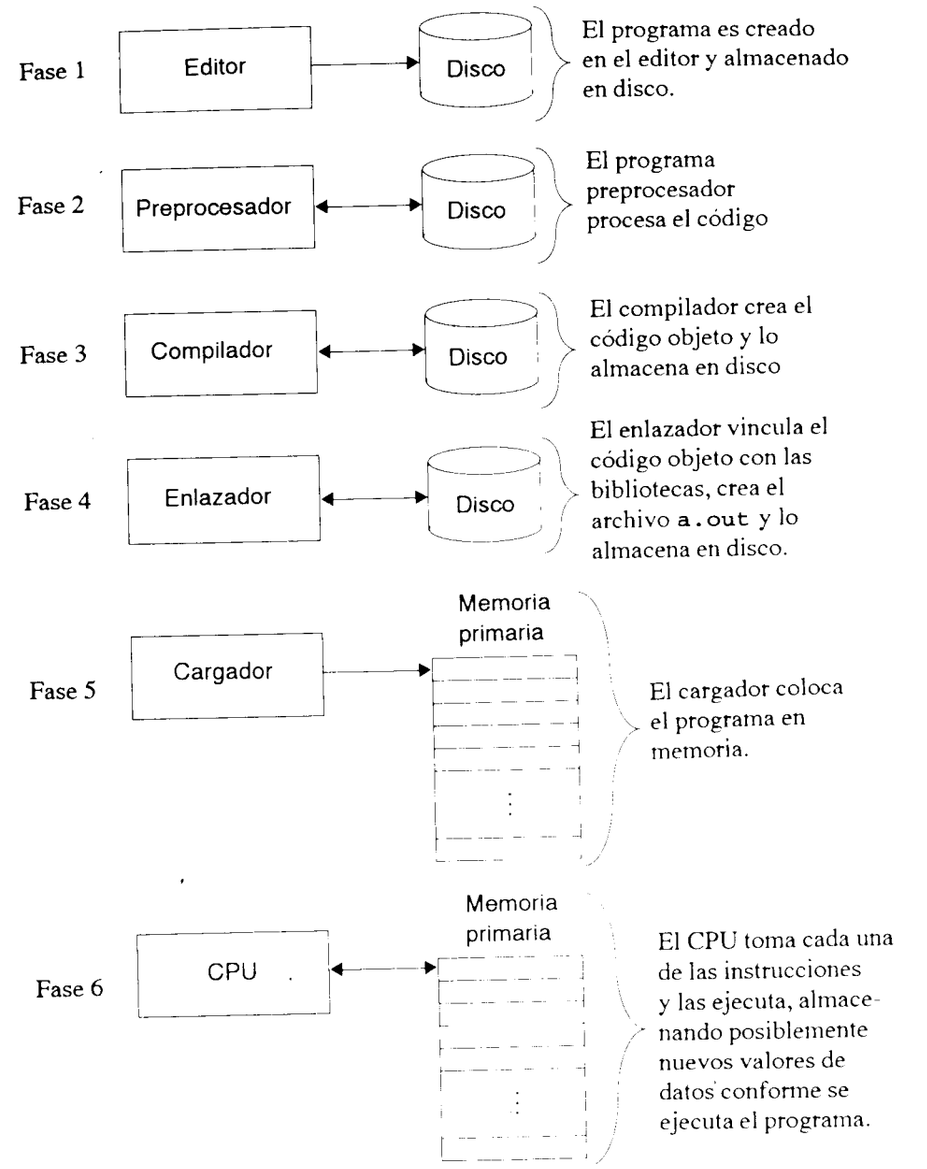

Fig. 1.1 Un entorno típico de C.

A continuación el programador da el comando de *compilar* el programa. El compilador traduce el programa C a código de lenguaje máquina (que también se conoce como *código objeto*). En un sistema C, un programa *preprocesador* ejecuta de forma automática antes de la fase de

traducción. El preprocesador C obedece comandos especiales que se llaman *directrices de preprocesador,* que indican que antes de su compilación se deben de ejecutar ciertas manipulaciones sobre el programa. Estas manipulaciones por lo regular consisten en la inclusión de otros archivos en el archivo a compilar y en el remplazo de símbolos especiales con texto de programa. Las directrices de preprocesador más comunes se analizan en los primeros capítulos; una presentación detallada de todas las características del preprocesador aparece en el capítulo 13. El preprocesador es invocado de manera automática por el compilador, antes que el programa sea convertido a lenguaje máquina.

La cuarta fase se conoce como *enlace*. Los programas C por lo general contienen referencias a funciones definidas en algún otro lugar como en bibliotecas estándar o en bibliotecas de un grupo de programadores que trabajen en un proyecto en particular. Entonces, el código objeto producido por el compilador C típicamente contendrá "huecos" debido a estas partes faltantes. Un *enlazador* vinculará el código objeto con el código de las funciones faltantes para producir una *imagen ejecutable* (sin ninguna parte faltante). En un sistema típico basado en UNIX, el comando para compilar y enlazar un programa es **cc**. Por ejemplo, para compilar y enlazar un programa de nombre **welcome.c**, escriba:

```
cc welcome.c
```

en la indicación de UNIX y presione la tecla de entrar. Si el programa compila y enlaza en forma correcta, se producirá un archivo con el nombre de **a.out**. Esta es la imagen ejecutable de nuestro programa **welcome.c**.

La quinta fase se llama *cargar*. Antes de que un programa pueda ser ejecutado, el mismo debe de ser colocado en memoria. Esto se lleva a cabo mediante el *cargador*, que toma la imagen ejecutable del disco y la transfiere a la memoria.

Por último, la computadora, bajo el control de su CPU, ejecuta el programa, una instrucción a la vez. Para cargar y ejecutar el programa en un sistema UNIX, escribimos **a.out** en la indicación de UNIX, y oprimimos la tecla de entrar.

La mayor parte de los programas en C reciben o proporcionan datos. Ciertas funciones de C toman sus entradas de **stdin** (*el dispositivo de entrada estándar*) que por lo regular queda asignado al teclado, pero **stdin** puede ser conectado a otro dispositivo. Los datos salen a **stdout** (el *dispositivo estándar de salida*) que por lo regular es la pantalla de la computadora, pero que puede ser conectado a otro dispositivo. Cuando decimos que un programa imprime un resultado, queremos decir que el resultado aparece desplegado en una pantalla. Los datos pueden salir a otros dispositivos, como son discos o impresoras. Existe también un *dispositivo estándar* de error, que se conoce como **stderr**. El dispositivo **stderr** (conectado a la pantalla), se utiliza para el despliegue de los mensajes de error. Es común encaminar los datos regulares de salida, es decir **stdout**, a un dispositivo distinto de la pantalla, mientras se conserva **stderr** asignado a la pantalla, de tal forma que el usuario de inmediato quede informado de errores.

1.12 Notas generales en relación con C y este libro

C es un lenguaje difícil. En ocasiones los programadores experimentados de C se enorgullecen de ser capaces de crear usos extraños, retorcidos y complicados del lenguaje. Esto es una mala práctica de programación. Hace que los programas sean difíciles de leer, es probable que se comporten en forma extraña y sean más difíciles de probar y de depurar. Este libro está organizado para programadores neófitos, por lo que hacemos hincapié en escribir programas claros y bien

estructurados. Una de las metas clave de este libro es conseguir *claridad* en el programa a través de las técnicas probadas de la programación estructurada y a través de las muchas buenas prácticas de programación relacionadas.

Práctica sana de programación 1.1

Escriba sus programas C de una forma simple y sencilla. Esto a veces se conoce como KIS (del inglés "keep it simple"). No "le saque punta al lenguaje" intentando "rarezas".

Habrá escuchado que C es un lenguaje portátil, y que los programas escritos en C pueden ser ejecutados en muy diversas computadoras. *La portabilidad es una meta elusiva*. El documento estándar ANSI (An90) enlista 11 páginas de temas sutiles de portabilidad. Se han escrito libros completos tratando el tema de la portabilidad en C (Ja89)(Ra90).

Sugerencia de portabilidad 1.3

Aunque es posible escribir programas portátiles, existen muchos problemas entre diferentes implantaciones de C y diferentes computadoras, que dificultan la portabilidad a alcanzar. La simple escritura de programas en C no garantiza la portabilidad.

Hemos hecho un cuidadoso recorrido del documento estándar ANSI C y auditado nuestra presentación contra éste, en relación con su integridad y exactitud. Sin embargo, C es un lenguaje muy rico, y existen algunas sutilezas del lenguaje y algunos temas avanzados que no hemos cubierto. Si necesita detalles técnicos adicionales sobre ANSI C, sugerimos que lea el documento estándar ANSI C mismo o el manual de referencia en Kernighan y Ritchie (Ke88).

Hemos limitado nuestras exposiciones a ANSI C. Muchas características de ANSI C no son compatibles con implantaciones anteriores de C, por lo que usted podrá encontrar que algunos de los programas de este texto no funcionan en compiladores C antiguos.

Práctica sana de programación 1.2

Lea los manuales correspondientes a la versión de C que esté utilizando. Consulte con frecuencia estos manuales para asegurar que está consciente de la gran cantidad de características de C y que está utilizando estas características de forma correcta.

Práctica sana de programación 1.3

Su computadora y su compilador son buenos maestros. Si no está seguro de cómo funciona una característica de C, escriba un programa de muestra con dicha característica, compile y ejecute el programa y vea qué es lo que ocurre.

1.13 C Concurrente

En los Laboratorios Bell mediante continuos esfuerzos de investigación se han desarrollado otras versiones de C. Gehani (Ge89) ha desarrollado *C Concurrente* un superconjunto de C que incluye capacidades de especificación de actividades múltiples en paralelo. Lenguajes como C Concurrente y funciones de sistemas operativos que dan soporte a paralelismo a aplicaciones de usuarios, se harán cada vez más populares en la siguiente década, conforme vaya aumentando el uso de *multiprocesadores* (es decir, computadoras con más de un CPU). A la fecha de esta escritura, C Concurrente es primordialmente aún un lenguaje de investigación. Los cursos de sistemas operativos y los libros de texto (De90) por lo regular incluyen análisis sustancial de la programación concurrente.

1.14 Programación orientada a objetos y C++

Otro superconjunto de C, es decir C++, fue desarrollado por Stroustrup (St86) en los Laboratorios Bell. C++ proporciona un cierto número de características que "engalanan" el lenguaje C. Pero lo que es aún más importante, permite llevar a cabo *programación orientada a objetos*.

Los *objetos* son en esencia *componentes* de software reutilizables que modelan elementos del mundo real. Está en marcha una revolución en la comunidad del software. Sigue siendo una meta no alcanzable la elaboración de software rápida, correcta y económica, y esto en un momento en que las demandas del software nuevo y más poderoso están encumbrándose.

Los desarrolladores de software están descubriendo que, utilizando un diseño e implantación modular orientada a objetos, puede hacer que los grupos de desarrollo de software sean de 10 a 100 veces más productivos que lo que era posible mediante técnicas convencionales de programación.

Se han desarrollado muchos lenguajes orientados a objetos. Se cree en general que C++ se convertirá en el lenguaje dominante de implantación de sistemas en la segunda parte de la década de los 90.

Muchas personas sienten que la mejor estrategia educacional hoy día es dominar C, y a continuación estudiar C++. Por ello, hemos dispuesto los capítulos 15 al 21, introduciendo programación orientada a objetos y C++. Esperamos que el lector los encuentre valiosos, y que los capítulos lo estimularán a seguir estudios posteriores de C++ una vez que haya terminado el estudio de este libro.

Resumen

- El software (es decir las instrucciones que usted escribe para darle órdenes a la computadora para que ejecute acciones y tome decisiones) es el que controla las computadoras (que a menudo se conocen como hardware).
- ANSI C es la versión del lenguaje de programación C estandarizada en 1989, tanto en los Estados Unidos, a través del American National Standards Institute (ANSI), como en todo el mundo a través de la International Standards Organization (ISO).
- Computadoras que hace 25 años pudieran haber llenado grandes habitaciones y costado millones de dólares, pueden ser ahora inscritas en la superficie de chips de silicón más pequeños que una uña, y que quizá cuestan cada una de ellas unos cuantos dólares.
- Aproximadamente se utilizan 150 millones de computadoras de uso general en todo el mundo, auxiliando a la personas en: negocios, industria, gobierno y en sus vidas personales. Esta cifra podría duplicarse con facilidad en pocos años.
- Una computadora es un dispositivo capaz de llevar a cabo cálculos y tomar decisiones lógicas a velocidades millones y a veces miles de millones más rápidas de lo que lo pueden hacer los seres humanos.
- Las computadoras procesan datos bajo control de programas de computación.
- Los diversos dispositivos (como: teclado, pantalla, discos, memoria y las unidades de procesamiento), que forman un sistema de cómputo, se conocen como hardware.
- Los programas de computación que se ejecutan en una computadora se conocen como software.
- La unidad de entrada es la sección de "recepción" de la computadora. La mayor parte de la información hoy día entra en las computadoras a través de teclados de tipo máquina de escribir.

- La unidad de salida es la sección de "embarques" de la computadora. La mayor parte de la información sale de las computadoras hoy día desplegándose en pantallas o imprimiéndose en papel.
- La unidad de memoria es la sección de "almacén" de la computadora, y a menudo se llama memoria o memoria primaria.
- La unidad aritmética y lógica (ALU) ejecuta cálculos y toma decisiones.
- La unidad de procesamiento central (CPU) es el coordinador de la computadora y es responsable de supervisar la operación de las otras secciones.
- Los programas y los datos que no están en uso activo por otras unidades, por lo regular están colocados en dispositivos de almacenamiento secundario (como discos) hasta que se necesitan de nuevo.
- En el procesamiento por lotes de un solo usuario, la computadora ejecuta un programa a la vez, mientras procesa datos en grupos o en lotes.
- La multiprogramación implica la operación "simultánea" de muchas tareas en la computadora —la computadora comparte sus recursos entre las tareas.
- El tiempo compartido es un caso especial de la multiprogramación, en el cual los usuarios tienen acceso a la computadora desde terminales. Al parecer los usuarios están operando en forma simultánea.
- Con la computación distribuida, la computación de una organización está distribuida, mediante redes a los lugares en donde es ejecutado el trabajo real de la organización.
- Los servidores de archivos almacenan programas y datos que pueden ser compartidos por las computadoras cliente, distribuidas a todo lo largo de la red, de ahí el término computación cliente/servidor.
- Cualquier computadora sólo puede entender de forma directa su propio lenguaje máquina.
- Los lenguajes máquina por lo general consisten de cadenas de números (reducidos en forma última a unos y a ceros) que instruyen a las computadoras para que ejecuten sus operaciones más elementales, una a la vez. Los lenguajes máquina son dependientes de la máquina.
- Las abreviaturas similares al inglés forman la base de los lenguajes ensambladores. Los ensambladores traducen los programas en lenguaje ensamblador a lenguaje máquina.
- Los compiladores traducen programas de lenguajes de alto nivel a lenguaje máquina. Los lenguajes de alto nivel contienen palabras inglesas y notaciones matemáticas convencionales.
- C se conoce como el lenguaje de desarrollo del sistema operativo UNIX.
- Es posible escribir programas en C que resulten portátiles para la mayor parte de las computadoras.
- El estándar ANSI C fue aprobado en 1989.
- FORTRAN (FORmula TRANslator) se utiliza para aplicaciones matemáticas.
- COBOL (COmmon Business Oriented Language) se usa sobre todo para aplicaciones comerciales, que requieren de una manipulación precisa y eficiente en datos de grandes cantidades.
- La programación estructurada es un método disciplinado para escribir programas que resulten claros, demostrablemente correctos y fáciles de modificar.
- Pascal fue diseñado para enseñar programación estructurada en entornos académicos.
- Ada fue desarrollada bajo el patrocinio del Departamento de Defensa de los Estados Unidos (DOD) utilizando a Pascal como base.

- La capacidad de *multitareas* de Ada le permite a los programadores especificar actividades en paralelo.
- Todos los sistemas C consisten de tres partes: entorno, lenguaje y las bibliotecas estándar. Las funciones de biblioteca no forman parte del lenguaje C mismo; estas funciones ejecutan operaciones como entrada/salida y cálculos matemáticos.
- Los programas C por lo regular pasan a través de seis fases para ser ejecutados: edición, preproceso, compilación, enlace, carga, y ejecución.
- El programador escribe un programa utilizando un editor, y si es necesario hace correcciones.
- Un compilador traduce un programa C en código de lenguaje máquina (o código objeto).
- El preprocesador de C obedece directrices de preprocesador, que por lo regular indican otros archivos que se deben incluir en el archivo a compilar, y que ciertos símbolos deben ser remplazados con texto de programa.
- Un enlazador vincula el código objeto con el código de funciones faltantes, para producir una imagen ejecutable (sin ninguna pieza faltante).
- Un cargador toma del disco una imagen ejecutable, y la transfiere a la memoria.
- Una computadora bajo el control de su CPU ejecutará el programa una instrucción a la vez.
- Ciertas funciones de C (como **`scanf`**), toman su entrada de **`stdin`** (el dispositivo de entrada estándar), que por lo regular está asignado al teclado.
- Los datos salen a **`stdout`** (el dispositivo de salida estándar), que por lo regular es la pantalla de la computadora.
- También existe un dispositivo estándar de error, que se conoce como **`stderr`**. El dispositivo **`stderr`** (la pantalla), se utiliza para el despliegue de mensajes de error.
- Aunque es posible escribir programas portátiles, existen muchos problemas entre diferentes implantaciones de C y diferentes computadoras, que pueden hacer la portabilidad difícil de alcanzar.
- El C concurrente es un superconjunto de C, que incluye capacidades para especificar la ejecución de varias actividades en paralelo.
- C++ proporciona capacidades para hacer programación orientada a objetos.
- Los objetos son en esencial componentes de software reutilizables, que modelan elementos del mundo real.
- Se cree que C++ se convertirá en el lenguaje dominante de implantación de sistemas en los años finales de 1990.

Terminología

`.c` extensión
Ada
ALU
ANSI C
método de bloques constructivos
C
preprocesador C
biblioteca estándar C
unidad aritmética y lógica (ALU)
ensamblador
lenguaje ensamblador
procesamiento por lotes
independiente de la máquina
lenguaje máquina
memoria
unidad de memoria

C++
unidad de procesamiento central (CPU)
claridad
cliente
computación cliente/servidor
COBOL
compilador
computadora
programa de computación
programador de computación
C Concurrente
CPU
datos
computación distribuida
editor
entorno
imagen ejecutable
ejecutar un programa
servidor de archivo
FORTRAN
función
funcionalización
hardware
plataforma de hardware
lenguaje de alto nivel
dispositivo de entrada
unidad de entrada
entrada/salida (E/S)
enlazador
cargador
unidades lógicas
dependiente de la máquina
multiprocesador
multiprogramación
multitareas
lenguaje natural de una computadora
objeto
código objeto
programación orientada a objetos
dispositivo de salida
unidad de salida
Pascal
computadora personal
portabilidad
memoria primaria
lenguaje de programación
ejecutar un programa
pantalla
software
reutilización del software
error estándar (**`stderr`**)
entrada estándar (**`stdin`**)
salida estándar (**`stdout`**)
programa almacenado
programación estructurada
supercomputadora
tarea
terminal
tiempo compartido
refinamiento con paso de arriba a abajo
programa traductor
UNIX
estación de trabajo

Prácticas sanas de programación

1.1 Escriba sus programas C de una forma simple y sencilla. Esto a veces se conoce como KIS (del inglés "keep it simple"). No "le saque punta al lenguaje" intentando "rarezas".

1.2 Lea los manuales correspondientes a la versión de C que esté utilizando. Consulte a menudo estos manuales para asegurarse que está consciente de la gran cantidad de características de C y que está utilizando estas características de forma correcta.

1.3 Su computadora y su compilador son buenos maestros. Si no está seguro de cómo funciona una característica de C, escriba un programa de muestra con dicha característica, compile y ejecute el programa y vea qué es lo que ocurre.

Sugerencias de portabilidad

1.1 Dado que C es un lenguaje independiente de hardware y muy disponible, las aplicaciones que están escritas en C pueden ejecutarse con poca o ninguna modificación en una amplia gama de distintos sistemas de cómputo.

1.2 El usar funciones de biblioteca estándar ANSI en vez de escribir sus propias versiones comparables, puede mejorar la portabilidad del programa porque estas funciones están incluidas en casi todas las implantaciones de ANSI C.

1.3 Aunque es posible escribir programas portátiles, existen muchos problemas entre diferentes implantaciones de C y diferentes computadoras, que dificultan la portabilidad a alcanzar. La simple escritura de programas en C no garantiza la portabilidad.

Sugerencia de rendimiento

1.1 El usar funciones de la biblioteca estándar ANSI, en vez de escribir sus propias versiones comparables, puede mejorar el rendimiento de los programas, porque estas funciones están escritas con cuidado para que se ejecuten con eficacia.

Ejercicios de autoevaluación

1.1 Llene los espacios en blanco en cada uno de los siguientes:

a) La compañía que en el mundo inició el fenómeno de la computadora personal fue_____.
b) La computadora que legitimizó la computadora personal en los negocios y en la industria fue la _____.
c) Las computadoras procesan datos bajo el control de conjuntos de instrucciones que se conocen como _____ de computación.
d) Las seis unidades lógicas claves de la computadora son los _____, _____, _____, _____, _____, y los _____.
e) _____ es un caso especial de multiprogramación en el cual los usuarios tienen acceso a la computadora mediante dispositivos que se conocen como terminales.
f) Las tres clases de lenguajes que se analizaron en el capítulo son_____, _____, y _____.
g) Los programas que traducen los programas de lenguaje de alto nivel al lenguaje máquina se llaman_____.
h) C se conoce ampliamente como el lenguaje de desarrollo del sistema operativo _____.
i) Este libro presenta la versión de C que se conoce como _____C la cual fue recientemente estandarizada a través de la American National Standards Institute.
j) El lenguaje _____ fue desarrollado por Wirth para la enseñanza de la programación estructurada en las universidades.
k) El departamento de la defensa desarrolló el lenguaje Ada con una capacidad que se conoce como _____ la cual permite a los programadores especificar que varias actividades pueden proceder en paralelo.

1.2 Llene los espacios en blanco en cada una de las siguientes oraciones en relación con el entorno de C.

a) Los programas de C se escriben por lo regular en una computadora utilizando un programa _____.
b) En un sistema C se ejecuta automáticamente un programa _____ antes que empiece la fase de traducción.
c) Los dos tipos más comunes de directrices de preprocesador son _____, y _____.
d) El programa _____ combina la salida del compilador con varias funciones de biblioteca a fin de producir una imagen ejecutable.
e) El programa _____ transfiere la imagen ejecutable del disco a la memoria.
f) Para cargar y ejecutar el programa recién compilado en un sistema UNIX, escriba _____.

Respuestas a los ejercicios de autoevaluación

1.1 a) Apple. b) Computadora personal de IBM. c) programas. d) unidad de entrada, unidad de salida, unidad de memoria, unidad de aritmética y lógica (ALU), unidad de procesamiento central (CPU), unidad de almacenamiento secundario. e) tiempo compartido. f) lenguajes de máquina, lenguajes ensambladores, y lenguajes de alto nivel. g) compiladores. h) UNIX. i) ANSI. j) Pascal. k) multitareas.

1.2 a) editor, b) preprocesador. c) incluyendo otros archivos en el archivo a compilarse, y remplazando símbolos especiales por texto de programa. d) enlazador. e) cargador. f) `a.out`.

Ejercicios

1.3. Clasifique cada uno de los elementos siguientes como hardware o software.

a) CPU
b) compilador C
c) ALU
d) preprocesador C
e) unidad de entrada
f) programa de procesamiento de texto

1.4 ¿Por qué escribiría usted un programa en un lenguaje independiente de máquina en vez de un lenguaje dependiente de máquina?. ¿Por qué sería más apropiado un lenguaje dependiente de máquina para escribir ciertos tipos de programas?

1.5 Los programas de traducción como son los ensambladores y los compiladores, convierten programas de un lenguaje (que se conoce como lenguaje *fuente*) a otro lenguaje (el cual se conoce como lenguaje *objeto*). Determine cuál de los siguientes enunciados son ciertos y cuáles falsos.

a) Un compilador traduce programas de alto nivel en lenguaje objeto.
b) Un ensamblador traduce programas de lenguaje fuente en programas de lenguaje máquina.
c) Un compilador convierte programas de lenguaje fuente en programas de lenguaje objeto.
d) Los lenguajes de alto nivel son normalmente dependientes de máquina.
e) Un programa de lenguaje máquina requiere de traducción antes de que el programa pueda ser ejecutado en una computadora.

1.6 Llene los espacios en blanco en cada uno de los enunciados siguientes.

a) Los dispositivos a partir de los cuales los usuarios tienen acceso a sistemas de computación de tiempo compartido por lo común se conocen como _____.
b) Un programa de computación que convierte programas de lenguaje ensamblador al lenguaje de máquina se llama _____.
c) La unidad lógica de la computadora que recibe información desde fuera de la misma para su uso se conoce como _____.
d) El proceso de instruir a la computadora para la solución de problemas específicos se llama _____.
e) ¿Qué tipo de lenguaje de computadora utiliza abreviaturas similares al inglés para instrucciones de lenguaje máquina? _____.
f) ¿Cuáles son las seis unidades lógicas de la computadora? _____.
g) ¿Cuál unidad lógica de la computadora envía información ya procesada por la computadora hacia varios dispositivos, de tal forma que esta información pueda ser utilizada fuera de la computadora? _____.
h) El nombre genérico de un programa que convierte programas escritos en un lenguaje específico de computadora al lenguaje máquina es_____.
i) ¿Qué unidad lógica de la computadora guarda la información? _____.
j) ¿Qué unidad lógica de la computadora ejecuta los cálculos? _____.
k) ¿Qué unidad lógica de la computadora toma decisiones lógicas? _____.

l) La abreviatura que por lo general se utiliza para la unidad de control de la computadora es _____.

m) El nivel de lenguaje de la computadora más conveniente para el programador para rápida y fácilmente escribir programas es _____.

n) El lenguaje orientado a los negocios más común y de amplio uso hoy en día es _____.

o) El único lenguaje que una computadora puede entender directamente se llama el lenguaje _____ de la computadora.

p) ¿Qué unidad lógica de la computadora coordina las actividades de todas las demás unidades lógicas? _____.

1.7. Diga si cada uno de los siguientes es verdadero o falso. Explique sus respuestas.

a) Los lenguajes máquina son en general dependientes de la máquina.

b) El tiempo compartido en realidad hace operar a varios usuarios a la vez en una computadora.

c) Al igual que otros lenguajes de alto nivel, C se considera en general como independiente de la máquina.

1.8. Analice el significado de cada uno de los nombres siguientes en el entorno UNIX:

a) `stdin`

b) `stdout`

c) `stderr`

1.9. ¿Qué capacidad clave se proporciona en C concurrente que no está disponible en ANSI C?.

1.10. ¿Por qué se le da tanta atención a la programación orientada a objetos hoy día en general y a C++ en particular?

Lectura recomendada

(An 90) ANSI, *American National Standard for Information Systems Programming Language C (ANSI Document ANSI/ISO 9899: 1990)*, New York, NY: American National Standards Institute, 1990.

Este es el documento de definición para ANSI C. El documento está disponible para su venta de la American National Standards Institute, 1430 Broadway, New York, New York 10018.

(De90) Deitel, H. M., *Operating Systems* (segunda edición), Reading. MA: Addison-Wesley Publishing Company, 1990.

Un libro de texto para el curso tradicional de ciencia de la computación en sistemas operativos. Los capítulos 4 y 5 presentan un análisis extenso de la programación concurrente.

(Ge89) Gehani, N., y W. D. Roome, *The Concurrent C Programming Language*, Summit, NJ:Silicon Press, 1989.

Es el libro definitorio correspondiente a C concurrente —un superconjunto del lenguaje C que permite a los programadores ejecutar ejecución en paralelo de muchas actividades. También incluye un resumen de C++ concurrente.

(Ja89) Jaeschke, R., *Portability and the C Language*, Indianapolis, IN: Hayden Books, 1989.

Este libro analiza la escritura de programas portátiles en C. Jaeschke sirvió tanto en los comités de estándares ANSI como en el ISO C.

(Ke88) Kernighan, B. W., y D. M Ritchie, *The C Programming Language* (segunda edición), Englewood Cliffs, NJ: Prentice Hall, 1988.

Este libro es el clásico en su campo. El libro se utiliza de forma extensa en cursos de C y en seminarios para programadores establecidos, e incluye un manual de consulta excelente. Ritchie es el autor del lenguaje C y uno de los codiseñadores del sistema operativo UNIX.

(Pl92) Plauger, P. J., *The Standard C Library*, Englewood Cliffs, NJ: Prentice Hall, 1992.

Define y demuestra la utilización de las funciones de la biblioteca estándar C. Plauger sirvió como director del subcomité de bibliotecas del comité que desarrolló el estándar ANSI C, y ahora sirve como Convenor del comité ISO del cual resultó C.

(Ra90) Rabinowitz, H., y C. Schaap, *Portable C*, Englewood Cliffs, NJ: Prentice Hall, 1990.

Este libro se desarrolló para un curso sobre portabilidad que se dio en los Laboratorios Bell de AT&T. Rabinowitz está con el laboratorio de inteligencia artificial de la corporación NYNEX, y Schaap es un funcionario importante en la corporación Delft Consulting.

(Ri78) Ritchie, D. M.; S. C. Johnson; M. E. Lesk; y B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language", *The Bell System Technical Journal*, Vol. 57, No. 6 Part 2, julio-agosto 1978, pp. 1991-2019.

Este es uno de los artículos clásicos de introducción al lenguaje C. Apareció en una emisión especial del *Bell System Technical Journal* dedicado al "sistema de tiempo compartido UNIX"

(Ri84) Ritchie, D. M., "The UNIX System: The Evolution of the UNIX Time-Sharing System", *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Part 2, octubre de 1984, pp. 1577-1593.

Un artículo clásico sobre el sistema operativo UNIX. Este artículo apareció en una edición especial del *Bell System Technical Journal* totalmente dedicado "al sistema UNIX"

(Ro84) Rosler, L., "The UNIX System: The Evolution of C —Past and Future", *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, Parte 2, octubre de 1984, pp. 1685-1699.

Un excelente artículo para seguir al (Ri78) para aquel lector interesado en rastrear la historia de C y las raíces del esfuerzo de normalización de ANSI C. Apareció en una edición especial del *Bell System Technical Journal* "dedicado al sistema UNIX"

(Si84) Stroustrup, B., "The UNIX System: Data Abstraction in C", *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8 Parte 2, octubre 1984, pp. 1701-1732.

Este artículo clásico de introducción a C++. Apareció en una edición especial del *Bell System Technical Journal* dedicado al "sistema UNIX".

(St91) Stroustrup, B. *The C++ Programming Language* (segunda edición), Reading, MA: Addison-Wesley Series in Computer Science, 1991.

Este libro es la referencia definitoria de C++, un superconjunto de C que incluye varias mejoras a C, especialmente características para la programación orientada a objetos. Stroustrup desarrolló C++ en los Laboratorios Bell de AT&T.

(To89) Tondo, C. L., y S. E. Gimpel, *The C Answer Book*, Englewood Cliffs. NJ: Prentice Hall, 1989.

Este libro único proporciona las respuestas a los ejercicios de Kernighan y Ritchie (Ke88). Los autores demuestran un estilo de programación ejemplar, y proporcionan pensamientos en sus métodos de resolución de problemas y en su decisiones de diseño. Tondo trabaja en IBM y en la Universidad de Nova en Ft. Lauderdale, Florida. Gimpel es un asesor.