CENGAGE Learning®

Programación C++:

del Análisis de Problemas al Diseño de Programas

SEXTA EDICIÓN

D.S. MALIK

Programación C++:

del Análisis de Problemas al Diseño de Programas

SEXTA EDICIÓN

D. S. MALIK

Traducción

Enrique C. Mercado González

Traductor profesional

Revisión técnica

Mtro. Roberto de Luna Caballero

Dra. Fabiola Ocampo Botello

Mtro. José Sánchez Juárez

Escuela Superior de Cómputo, Instituto Politécnico Nacional

CENGAGE Learning®

Australia • Brasil • Japón • Corea • México • Singapur • España • Reino Unido • Estados Unidos

Programación C++: del Análisis de Problemas al Diseño de Programas, Sexta edición.
D.S. Malik

Presidente de Cengage Learning Latinoamérica:
Fernando Valenzuela Migoya

Director Editorial, de Producción y de Plataformas Digitales para Latinoamérica:
Ricardo H. Rodríguez

Gerente de Procesos para Latinoamérica:
Claudia Islas Licona

Gerente de Manufactura para Latinoamérica:
Raúl D. Zendejas Espejel

Gerente Editorial de Contenidos en Español:
Pilar Hernández Santamarina

Gerente de Proyectos Especiales:
Luciana Rabuffetti

Coordinador de Manufactura:
Rafael Pérez González

Editora:
Abril Vega Orozco

Diseño de portada:
Roycroft Design

Imagen de portada:
©Masterfile Royalty Free

Composición tipográfica:
Foto Grafic & Diseño

Traducido del libro *C++ Programming: From Problem Analysis to Program Design*, Sixth Edition.
D.S. Malik.
Publicado en inglés por Cengage Learning ©2013
ISBN: 978-1-133-62638-1

Datos para catalogación bibliográfica:
Malik, D.S. *Programación C++: del Análisis de Problemas al Diseño de Programas,* sexta edición.
ISBN: 978-607-481-921-2

Visite nuestro sitio en:
http://latinoamerica.cengage.com

Impreso en México
1 2 3 4 5 6 7 16 15 14 13

Tabla de contenido

CAPÍTULO 1

VISIÓN PANORÁMICA DE LAS COMPUTADORAS Y LOS LENGUAJES DE PROGRAMACIÓN

EN ESTE CAPÍTULO:

- Aprenderás acerca de los diferentes tipos de computadoras
- Explorarás los componentes de hardware y software de un sistema de computación
- Aprenderás acerca del lenguaje de una computadora
- Aprenderás acerca de la evolución de los lenguajes de programación
- Examinarás los lenguajes de programación de alto nivel
- Descubrirás qué es un compilador y qué hace
- Examinarás un programa en C++
- Explorarás cómo se procesa un programa en C++
- Aprenderás qué es un algoritmo y explorarás técnicas de resolución de problemas
- Conocerás las metodologías de programación estructurada y de programación orientada a objetos
- Conocerás el C++ estándar y el C++ estándar de ANSI/ISO

Introducción

Términos como "internet", desconocidos hace apenas veinte años, son comunes en la actualidad. Hoy, estudiantes de educación primaria suelen "navegar" en internet y usar computadoras para hacer sus tareas escolares. Muchas personas usan la internet para buscar información y comunicarse entre sí. Todo esto es posible gracias a la disponibilidad de diferentes tipos de *software*, llamados también programas de computación. Sin software, una computadora no sirve para nada. El software se desarrolla usando lenguajes de programación. El lenguaje de programación C++ está especialmente indicado para el desarrollo de software con el cual realizar tareas específicas. El principal objetivo de este libro es que aprendas a escribir programas en el lenguaje de programación C++. Pero antes de empezar a programar, es útil que conozcas algo de la terminología básica y los diferentes componentes de una computadora. Comencemos con un panorama de la historia de las computadoras.

Breve recuento de la historia de las computadoras

El primer artefacto para realizar cálculos de que se tiene noticia fue el ábaco, inventado en Asia pero usado en la antigua Babilonia, China y Europa hacia fines de la Edad Media. Este mecanismo se sirve de un sistema de cuentas que se deslizan sobre un bastidor para sumar y restar. En 1642 el filósofo y matemático francés Blaise Pascal inventó el dispositivo de cálculo conocido como pascalina. Tenía ocho esferas móviles montadas en ruedas y podía calcular cantidades de hasta ocho dígitos. Tanto el ábaco como la pascalina sólo podían hacer operaciones de suma y resta. Ya avanzado el siglo XVII, Gottfried von Leibniz inventó un aparato capaz de sumar, restar, multiplicar y dividir. En 1819, el tejedor francés Joseph Jacquard descubrió que las instrucciones de tejido para sus telares podían almacenarse en tarjetas perforadas. Mientras las tarjetas se movían en secuencia por el telar, las agujas atravesaban las perforaciones y tomaban hilos del color y la textura indicados. El operario podía reacomodar las tarjetas y cambiar la figura por tejer. En esencia, dichas tarjetas programaban un telar para producir figuras sobre tela. Aunque podría parecer que la industria de los hilados y tejidos tiene poco en común con la computación, la idea de almacenar información haciendo perforaciones en una tarjeta resultó de gran importancia para el desarrollo posterior de las computadoras.

A principios y mediados del siglo XIX, el matemático y físico inglés Charles Babbage diseñó dos calculadoras: el motor de diferencias y el motor analítico. El motor de diferencias podía hacer operaciones complejas, como elevar en forma automática números al cuadrado. Babbage produjo un prototipo del motor de diferencias, no el aparato propiamente dicho. El primer motor de diferencias completo se concluyó en Londres en el 2002, 153 años después de haberse concebido. Consta de 8000 partes, pesa cinco toneladas y mide 3.35 metros de largo. Una réplica se llevó a cabo en el 2008, y hoy se exhibe en el Computer History Museum de Mountain View, California (*http://www.computerhistory.org/babbage/*). Casi toda la obra de Babbage se conoce gracias a los textos de su colega Ada Augusta, condesa de Lovelace. Augusta es considerada la primera persona en haber programado una computadora.

A fines del siglo XIX, los funcionarios de la Oficina de Censos de Estados Unidos necesitaban ayuda para tabular con precisión sus datos censales. Herman Hollerith inventó entonces una calculadora que operaba con energía eléctrica y empleaba tarjetas perforadas para almacenar datos. Esta máquina tuvo mucho éxito.

Hollerith fundó la Tabulating Machine Company, que más tarde se convertiría en la corporación de computadoras y tecnología conocida como IBM.

La primera computadora fue la *Mark I*, fabricada conjuntamente en 1944 por IBM y la Universidad de Harvard bajo la dirección de Howard Aiken. Esta máquina era alimentada de datos por medio de tarjetas perforadas. La *Mark I* tenía casi 16 metros de largo, pesaba 50 toneladas y constaba de 750 000 partes. En 1946 se fabricó la ENIAC (*Electronic Numerical Integrator and Calculator*, Computador e Integrador Numérico Electrónico) en la Universidad de Pensilvania. Contenía 18 000 tubos de vacío y pesaba cerca de treinta toneladas.

Las computadoras que conocemos ahora se valen de las reglas de diseño establecidas por John von Neumann a fines de la década de 1940. Este diseño incluía componentes como la unidad de aritmética y lógica, una unidad de control, memoria y dispositivos de entrada/salida. Estos componentes se describen en la siguiente sección. El diseño de Von Neumann permite que una computadora almacene en el mismo espacio de memoria tanto instrucciones de programación como datos. En 1951 se fabricó la UNIVAC (*Universal Automatic Computer*, Computadora Automática Universal), la cual fue vendida a la Oficina de Censos de Estados Unidos.

En 1956, la invención de los transistores redundó en computadoras más pequeñas, veloces, confiables y ahorradoras de energía. Con la aparición de FORTRAN y COBOL, dos de los primeros lenguajes de programación, este periodo también atestiguó el surgimiento de la industria del desarrollo de software. En el adelanto tecnológico subsiguiente, los transistores fueron reemplazados por minúsculos circuitos integrados, o "chips". Éstos son más pequeños y económicos que los transistores, pueden contener miles de circuitos en una sola unidad y dotan a las computadoras de gran velocidad de procesamiento.

En 1970 se inventó el microprocesador, una unidad central de procesamiento (*central processing unit*, CPU) entera en un solo chip. En 1977, Stephen Wozniak y Steven Jobs diseñaron y fabricaron en un garaje la primera computadora Apple. En 1981, IBM presentó su computadora personal (*personal computer*, PC). En esa misma década, clones de la PC de IBM volvieron aún más accesible la computadora personal. Para mediados de la década de 1990, la PC estaba al alcance de todo tipo de personas. Las computadoras siguen siendo cada vez más veloces y menos costosas conforme la tecnología avanza.

Las computadoras de hoy son potentes, confiables y fáciles de usar. Pueden aceptar instrucciones de voz e imitar el razonamiento humano mediante inteligencia artificial. Sistemas expertos asisten a los médicos en la elaboración de diagnósticos. Las aplicaciones móviles de la computación han aumentado significativamente. Usando aparatos de bolsillo, los repartidores de paquetería pueden acceder a satélites de posicionamiento global (*global positioning satellites*, GPS) para precisar la ubicación de clientes para fines de recolección o entrega. Los teléfonos celulares nos permiten consultar nuestro correo electrónico, hacer reservaciones en líneas aéreas, ver cómo se comportan los valores bursátiles y acceder a nuestras cuentas bancarias.

Aunque existen varias categorías de computadoras, como supercomputadoras, medianas y micro, todas comparten algunos elementos básicos, los cuales se describirán en la siguiente sección.

Elementos de un sistema de computación

Una computadora es un dispositivo electrónico capaz de ejecutar comandos. Los comandos básicos que ejecuta son de entrada (obtener datos), salida (presentar resultados), almacena-

miento y realización de operaciones aritméticas y lógicas. Los dos componentes principales de un sistema de computación son el *hardware* y el software. En las secciones siguientes se hará una breve descripción de dichos componentes. Examinemos primero el hardware.

Hardware

Los componentes más importantes del hardware son la unidad central de procesamiento (CPU); la memoria principal (*main memory*, MM), también llamada memoria de acceso aleatorio (*random access memory*, RAM); los dispositivos de entrada/salida, y el almacenamiento secundario. Algunos ejemplos de dispositivos de entrada son el teclado, el ratón y el almacenamiento secundario. Ejemplos de dispositivos de salida son la pantalla, la impresora y el almacenamiento secundario. Examinemos en mayor detalle cada uno de estos componentes.

Unidad central de procesamiento y memoria principal

La **unidad central de procesamiento** es el "cerebro" de la computadora y la pieza de hardware más costosa. Cuanto más potente sea la CPU, más rápida será la computadora. En la CPU se realizan las operaciones aritméticas y lógicas. La figura 1.1*a)* muestra algunos componentes de hardware.

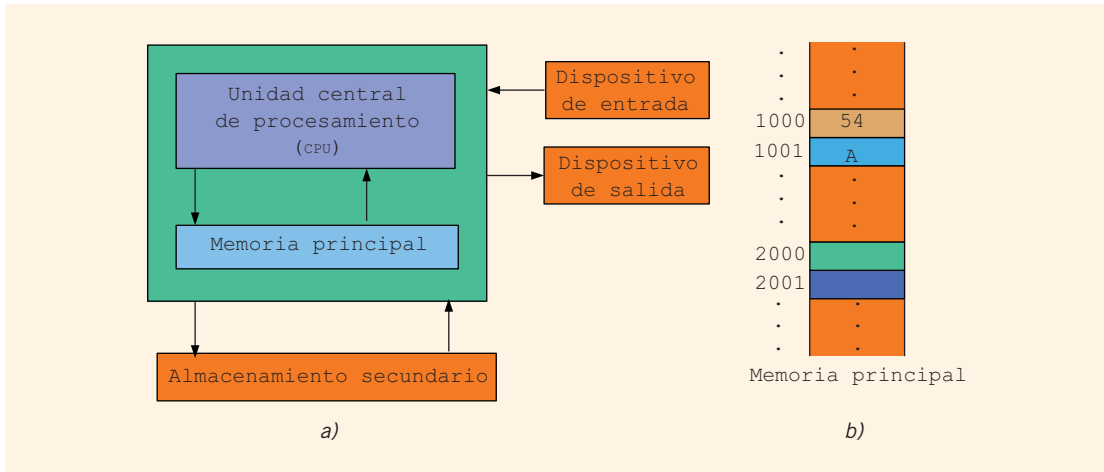

FIGURA 1.1 Componentes de hardware de una computadora y memoria principal

La **memoria principal**, o **memoria de acceso aleatorio**, está directamente conectada con la CPU. Todos los programas deben cargarse en la memoria principal para poder ejecutarse. De igual forma, todos los datos deben introducirse en la memoria principal para que un programa pueda manipularlos. Cuando la computadora se apaga, todo lo alojado en la memoria principal se pierde.

La memoria principal es una secuencia ordenada de celdas, llamadas **celdas de memoria**. Cada una de ellas tiene una ubicación propia en la memoria principal, llamada **dirección** de celda. Estas direcciones permiten acceder a la información almacenada en las celdas. La figura 1.1*b)* presenta la memoria principal junto con algunos datos.

Las computadoras de hoy contienen una memoria principal que consta de entre millones y miles de millones de celdas. Aunque la figura 1.1*b)* muestra datos almacenados en celdas, el contenido de una celda puede ser una instrucción de programación o datos. Además, esa figura exhibe los datos en forma de números y letras. Sin embargo, como se explicará más adelante, la memoria principal almacena todo como secuencias de ceros y unos. Las direcciones de memoria también se expresan como secuencias de ceros y unos.

ALMACENAMIENTO SECUNDARIO

Dado que los programas y datos deben almacenarse en la memoria principal para poder procesarse, y puesto que todo lo alojado en la memoria principal se pierde al apagar la computadora, la información almacenada en la memoria principal debe transferirse a otro dispositivo para su almacenamiento permanente. El mecanismo que almacena información de manera permanente (a menos que el dispositivo se vuelva inutilizable o que se cambie la información reescribiéndola) se llama **almacenamiento secundario**. Para poder transferir información de la memoria principal al almacenamiento secundario, ambos componentes deben estar directamente conectados entre sí. Ejemplos de almacenamiento secundario son los discos duros, las unidades flash, los discos flexibles, los discos ZIP, los CD-ROM y las cintas.

Dispositivos de entrada/salida

Para que una computadora pueda desempeñar una tarea útil debe ser capaz de alojar datos y programas y presentar los resultados de cálculos. Los dispositivos que alimentan de datos y programas a las computadoras se llaman **dispositivos de entrada**. El teclado, el ratón y el almacenamiento secundario son ejemplos de dispositivos de entrada. Los dispositivos que la computadora usa para presentar resultados se llaman **dispositivos de salida**. Un monitor, una impresora y el almacenamiento secundario son ejemplos de dispositivos de salida. En la figura 1.2 aparecen algunos dispositivos de entrada y de salida.

FIGURA 1.2 Algunos dispositivos de entrada y de salida

Software

El software consiste en programas escritos para llevar a cabo tareas específicas. Por ejemplo, los procesadores de palabras son programas que sirven para escribir cartas, artículos, incluso libros. Todo el software está escrito en lenguajes de programación. Existen dos tipos de programas: programas del sistema y programas de aplicación.

Los **programas del sistema** controlan la computadora. El programa del sistema que se carga primero, al encender una PC, se llama sistema operativo. Sin sistema operativo, la computadora es inútil. El **sistema operativo** monitorea la actividad general de la computadora y presta servicios. Algunos de estos servicios son gestión de memoria, actividades de entrada/salida y gestión de almacenamiento. El sistema operativo tiene un programa especial que organiza el almacenamiento secundario para poder acceder convenientemente a la información.

Los **programas de aplicación** desempeñan una tarea específica. Los procesadores de palabras, hojas de cálculo y juegos son ejemplos de programas de aplicación. El sistema operativo es el programa que ejecuta los programas de aplicación.

Lenguaje de una computadora

Al oprimir `A` en un teclado, la computadora presenta `A` en la pantalla. Pero ¿qué es lo realmente almacenado en la memoria principal de la computadora? ¿Cuál es el lenguaje de ésta? ¿Cómo almacena todo lo que se escribe en el teclado?

Hay que recordar que una computadora es un dispositivo electrónico. En ella se usan señales eléctricas para procesar información. Existen dos tipos de señales eléctricas: analógicas y digitales. Las **señales analógicas** son ondas continuas que sirven para representar cosas como sonidos. Las cintas de audio, por ejemplo, almacenan datos en señales analógicas. Las **señales digitales** representan información mediante una secuencia de ceros y unos. Un `0` representa bajo voltaje y un `1`, alto voltaje. Las señales digitales son portadoras de información más confiables que las analógicas y pueden copiarse de un dispositivo a otro con total fidelidad. Quizá hayas notado que cuando haces una copia de una cinta de audio, la calidad de sonido de la copia no es tan buena como la de la cinta original. En cambio, cuando copias un CD, la copia es tan buena como el original. Las computadoras usan señales digitales.

Puesto que en una computadora se procesan señales digitales, el lenguaje de una computadora, llamado **lenguaje de máquina**, es una secuencia de ceros y unos. Los dígitos `0` o `1` se llaman **dígito binario** (*binary digit*), o **bit**. A una secuencia de ceros y unos se le conoce como **código binario** o **número binario**.

Bit: Dígito binario, ya sea `0` o `1`.

Una secuencia de ocho bits se llama **byte**. Asimismo, 2^{10} bytes = 1024 bytes se llama **kilobyte (KB)**. La tabla 1.1 recoge los términos que se emplean para describir diversos números de bytes.

TABLA 1.1 Unidades binarias

Unidad	Símbolo	Bits/bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1 048 576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1 073 741 824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1 099 511 627 776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1 125 899 906 842 624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1 152 921 504 606 846 976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1 180 591 620 717 411 303 424 bytes

Cada letra, número o símbolo especial (como * o {) de tu teclado está codificado como una secuencia de bits, cada una con una representación única. El esquema de codificación de uso más común en las computadoras personales es el **American Standard Code for Information Interchange** (código estadounidense estándar para el intercambio de información, ASCII), de *siete bits*. El conjunto de datos de ASCII consta de 128 caracteres, numerados del 0 al 127. Es decir, en el conjunto de datos de ASCII la posición del primer carácter es 0, la del segundo es 1, y así sucesivamente. En este esquema, A se codifica con el número binario `1000001`. De hecho, A es el sexagésimo sexto carácter en el código de caracteres de ASCII, aunque su posición es la número 65, porque la del primer carácter es 0. Además, el número binario `1000001` es la representación binaria de `65`. El carácter `3` se codifica como `0110011`. Cabe señalar que en el conjunto de caracteres de ASCII, la posición del carácter `3` es 51, así que el carácter `3` es el quincuagésimo segundo carácter en ese conjunto. De esto se desprende asimismo que `0110011` es la representación binaria de `51`. Para una lista completa del conjunto de caracteres imprimibles de ASCII, consulta el apéndice C.

NOTA

El sistema numérico que usamos en la vida diaria se llama **sistema decimal**, o de **base 10**. Dado que en una computadora todo se representa como una secuencia de ceros y unos, es decir, con números binarios, el sistema numérico que emplea una computadora se llama **binario**, o de **base 2**. En el párrafo anterior se indicó que el número 1000001 es la representación binaria de 65. En el apéndice E se describe cómo convertir un número de base 10 a base 2, y viceversa.

En una computadora, cada carácter se representa como una secuencia de *ocho* bits, es decir, como un byte. Así, la representación binaria de ocho dígitos de 65 es `01000001`. Nota que se ha añadido un `0` a la izquierda de la representación de siete bits de 65 para convertirla en una representación de ocho bits. De igual forma, la representación binaria de ocho bits de 51 es `00110011`.

ASCII es un código de siete bits. Por tanto, para representar cada uno de sus caracteres en una computadora la representación binaria de siete bits debe convertirse en una representación binaria de ocho dígitos. Esto se hace añadiendo un `0` a la izquierda de la codificación de siete bits de un carácter de ASCII. De ahí que en una computadora el carácter `A` se represente como `01000001`, y el carácter `3`, como `00110011`.

Existen otros esquemas de codificación, como EBCDIC (*Extended Binary-coded Decimal Interchange Code*, código ampliado para el intercambio de decimales con codificación binaria, usado por IBM) y Unicode, el cual es un desarrollo más reciente. EBCDIC consta de 256 caracteres; Unicode, de 65 536. Para almacenar un carácter de Unicode se necesitan dos bytes.

Evolución de los lenguajes de programación

El lenguaje básico de una computadora, el lenguaje de máquina, da instrucciones de programas en bits. Aunque la mayoría de las computadoras ejecutan el mismo tipo de operaciones, los diseñadores de un tipo particular de máquinas pueden haber elegido conjuntos diferentes de códigos binarios para efectuar dichas operaciones. En consecuencia, el lenguaje de máquina de una computadora no necesariamente es igual al de otra. La única regularidad entre máquinas es que en cualquier computadora moderna todos los datos se almacenan y manipulan como códigos binarios.

Las primeras computadoras se programaron en lenguaje de máquina. Para ver cómo se escriben las instrucciones en lenguaje de máquina, supongamos que deseas usar la ecuación:

```
salario = tasa * horas
```

para calcular el salario semanal. Supongamos además que el código binario `100100` significa cargar (*load*), `100110` significa multiplicar (*multiply*) y `100010` significa almacenar (*store*). En lenguaje de máquina, podrías necesitar la siguiente secuencia de instrucciones para calcular el salario semanal:

```
100100 010001
100110 010010
100010 010011
```

Para representar la ecuación de salario semanal en lenguaje de máquina, el programador tuvo que recordar los códigos del lenguaje de máquina relativos a diversas operaciones. Asimismo, para manipular datos tuvo que recordar la ubicación de los datos en la memoria principal. La necesidad de recordar códigos específicos hacía que la programación fuera no sólo difícil, sino también propensa a errores.

Se desarrollaron entonces lenguajes ensambladores para facilitar el trabajo del desarrollador. En lenguaje ensamblador una instrucción es una forma fácil de recordar, llamada recurso **nemotécnico**. La tabla 1.2 muestra algunos ejemplos de instrucciones en lenguaje ensamblador y su código correspondiente en lenguaje de máquina.

CAPÍTULO 3

ENTRADA/SALIDA

EN ESTE CAPÍTULO:

- Aprenderás qué es un flujo y examinarás flujos de entrada y de salida
- Explorarás cómo leer datos desde el dispositivo de entrada estándar
- Aprenderás a usar funciones predefinidas en un programa
- Explorarás cómo usar las funciones de flujo de entrada `get`, `ignore`, `putback` y `peek`
- Te familiarizarás con las fallas de entrada
- Aprenderás a escribir datos en el dispositivo de salida estándar
- Descubrirás cómo usar manipuladores en un programa para formatear la salida
- Aprenderás a realizar operaciones de entrada y salida con el tipo de datos `string`
- Aprenderás a depurar errores lógicos
- Te familiarizarás con la entrada y salida en archivos

En el capítulo 2 se presentaron algunas de las instrucciones de entrada/salida (E/S) de C++, las cuales introducen datos en un programa e imprimen los resultados en la pantalla. Usaste `cin` y el operador de extracción >> para obtener datos del teclado, y `cout` y el operador de inserción << para enviar salida a la pantalla. Puesto que las operaciones de E/S son fundamentales para cualquier lenguaje de programación, en este capítulo aprenderás más detalladamente las operaciones de E/S de C++. Primero conocerás las instrucciones que extraen los datos de entrada del dispositivo de entrada estándar y envían salida al dispositivo de salida estándar. Después aprenderás a formatear la salida usando manipuladores. Además, conocerás las limitaciones de las operaciones de E/S asociadas con los dispositivos de entrada/salida estándar y aprenderás a extender estas operaciones a otros dispositivos.

Flujos de E/S y dispositivos de E/S estándar

Un programa realiza tres operaciones básicas: obtiene datos, los manipula y da resultados. En el capítulo 2 aprendiste a manipular datos numéricos usando operaciones aritméticas. En capítulos posteriores aprenderás a manipular datos no numéricos. Debido a que escribir programas para E/S es muy complejo, C++ ofrece amplio apoyo para operaciones de E/S proporcionando operaciones sustanciales de E/S preescritas, algunas de éstas las encontraste en el capítulo 2. En este capítulo aprenderás varias operaciones de E/S que pueden aumentar enormemente la flexibilidad de tus programas.

En C++, la E/S es una secuencia de bytes, llamada flujo, de la fuente al destino. Los bytes son usualmente caracteres, a menos que el programa requiera otros tipos de información, como una imagen gráfica o habla digital. Por tanto, un **flujo** es una secuencia de caracteres de la fuente al destino. Hay dos tipos de flujos:

Flujo de entrada: Secuencia de caracteres de un dispositivo de entrada a la computadora.

Flujo de salida: Secuencia de caracteres de la computadora a un dispositivo de salida.

Como recordarás, el dispositivo de entrada estándar suele ser el teclado, y el dispositivo de salida estándar suele ser la pantalla. Para recibir datos del teclado y enviar salida a la pantalla, cada programa en C++ debe usar el archivo de encabezado `iostream`. Este archivo de encabezado contiene, entre otras cosas, las definiciones de dos tipos de datos: `istream` (flujo de entrada) y `ostream` (flujo de salida). Ese archivo de encabezado también contiene dos declaraciones de variables: una para `cin`, que significa **entrada común** (*common input*), y otra para `cout`, que significa **salida común** (*common output*).

Estas declaraciones de variables son similares a las siguientes instrucciones en C++:

```
istream cin;
ostream cout;
```

Para usar `cin` y `cout`, todos los programas en C++ deben usar la directiva del preprocesador:

```
#include <iostream>
```

NOTA

Recuerda que en el capítulo 2 se usó la instrucción `using namespace std;`, además de incluir el archivo de encabezado `iostream`, para usar `cin` y `cout`. Sin la instrucción `using namespace std;`, se debe hacer referencia a esos identificadores como `std::cin` y `std::cout`. En el capítulo 7 conocerás en detalle el significado de la instrucción `using namespace std;`.

Las variables de tipo `istream` se llaman **variables de flujos de entrada**; las variables de tipo `ostream` se llaman **variables de flujos de salida**. Una **variable de flujos** es una variable de flujos de entrada o una variable de flujos de salida.

Debido a que `cin` y `cout` ya están definidos y tienen significados específicos, para evitar confusiones nunca se debe redefinir en programas.

La variable `cin` tiene acceso a operadores y funciones que pueden usarse para extraer datos del dispositivo de entrada estándar. Tú ya has usado brevemente el operador de extracción >> para introducir datos procedentes del dispositivo de entrada estándar. En la siguiente sección se describirá en detalle cómo funciona el operador de extracción >>. En las secciones subsecuentes aprenderás a usar las funciones `get, ignore, peek` y `putback` para introducir datos en una forma específica.

`cin` y el operador de extracción >>

En el capítulo 2 vimos cómo introducir datos desde el dispositivo de entrada estándar usando `cin` y el operador de extracción >>. Supongamos que `payRate` es una variable `double`. Considera la siguiente instrucción en C++:

```
cin >> payRate;
```

Cuando la computadora ejecuta esta instrucción, introduce el siguiente número que se escribió con el teclado y lo almacena en `payRate`. Así, si el usuario escribe `15.50`, el valor almacenado en `payRate` es `15.50`.

El operador de extracción >> es binario y, por tanto, acepta dos operandos. El operando izquierdo debe ser una variable de flujos de entrada, como `cin`. Debido a que el propósito de una instrucción de entrada es leer y almacenar valores en una ubicación de memoria, y puesto que sólo variables remiten a ubicaciones de memoria, el operando de la derecha es una variable.

NOTA

El operador de extracción >> únicamente está definido para introducir datos en variables de tipos de datos simples. Por tanto, el operando derecho del operador de extracción >> es una variable del tipo de datos simple. Sin embargo, C++ permite al programador extender la definición del operador de extracción >> para que los datos también puedan introducirse en otros tipos de variables usando una instrucción de entrada. Conocerás este mecanismo en el capítulo 13.

La sintaxis de una instrucción de entrada que usa `cin` y el operador de extracción >> es:

```
cin >> variable >> variable...;
```

Como puedes ver en la sintaxis anterior, una instrucción de entrada puede leer más de un elemento de datos usando varias veces el operador >>. Cada ocurrencia de >> extrae el siguiente elemento de datos del flujo de entrada. Por ejemplo, puedes leer tanto `payRate` como `hoursWorked` por medio de una sola instrucción de entrada usando el siguiente código:

```
cin >> payRate >> hoursWorked;
```

No hay ninguna diferencia entre la instrucción de entrada anterior y las dos instrucciones de entrada que aparecen a continuación. Decidir qué forma usar es cuestión de conveniencia y estilo.

```
cin >> payRate;
cin >> hoursWorked;
```

¿Cómo funciona el operador de extracción >>? Al buscar la entrada siguiente, >> se salta todos los caracteres de espacio en blanco. Recuerda que los caracteres de espacio en blanco constan de blancos y ciertos caracteres no imprimibles, como tabuladores y el carácter de línea nueva. Así, si separas los datos de entrada con líneas o blancos, el operador de extracción >> simplemente busca el siguiente dato de entrada en el flujo de entrada. Por ejemplo, supongamos que `payRate` y `hoursWorked` son variables `double`. Considera esta instrucción de entrada:

```
cin >> payRate >> hoursWorked;
```

Si la entrada es:

```
15.50 48.30
```

o

```
15.50  48.30
```

o

```
15.50
48.30
```

la instrucción de entrada precedente almacenaría `15.50` en `payRate` y `48.30` en `hoursWorked`. Nota que la primera entrada está separada por un blanco; la segunda, por un tabulador, y la tercera, por una línea.

Supongamos ahora que la entrada es `2`. ¿Cómo distingue el operador de extracción >> entre el carácter `2` y el número `2`? El operando a la derecha del operador de extracción >> hace esta distinción. Si el operando de la derecha es una variable del tipo de datos `char`, la entrada `2` se trata como el carácter `2` y, en este caso, se almacena el valor ASCII de `2`. Si el operando de la derecha es una variable del tipo de datos `int` o `double`, la entrada `2` se trata como el número `2`.

Considera ahora la entrada `25` y la instrucción:

```
cin >> a;
```

donde `a` es una variable de algún tipo de datos simple. Si `a` es del tipo de datos `char`, en `a` se almacena únicamente el carácter `2`. Si `a` es del tipo de datos `int`, en `a` se almacena `25`. Si `a` es

del tipo de datos `double`, la entrada `25` se convierte en el número decimal `25.0`. La tabla 3.1 resume este análisis mostrando la entrada válida para una variable del tipo de datos simple.

TABLA 3.1 Entrada válida para una variable del tipo de datos simple

Tipo de datos de a	Entrada válida para a
`char`	Un carácter imprimible, excepto el blanco
`int`	Un entero, quizá precedido por un signo + o −
`double`	Un número decimal, quizá precedido por un signo + o −. Si la entrada de datos real es un entero, se convierte en número decimal con decimal cero.

Al introducir datos en una variable `char`, y después de saltarse todos los caracteres de espacio en blanco, el operador de extracción >> busca y almacena únicamente el siguiente carácter; la lectura se detiene después de un solo carácter. Para leer datos en una variable `int` o `double` y después de saltarse todos los caracteres de espacio en blanco y leer el signo más o menos (si lo hay), el operador de extracción >> lee los dígitos del número, incluido el punto decimal de las variables de punto flotante, y se detiene al encontrar un carácter de espacio en blanco o un carácter que no sea un dígito.

EJEMPLO 3.1

Supongamos que se tienen las siguientes declaraciones de variables:

```
int a, b;
double z;
char ch;
```

Las siguientes instrucciones muestran cómo funciona el operador de extracción >>.

	Instrucción	Entrada	Valor almacenado en memoria
1	`cin >> ch;`	`A`	`ch = 'A'`
2	`cin >> ch;`	`AB`	`ch = 'A', 'B'` se guarda para una entrada posterior
3	`cin >> a;`	`48`	`a = 48`
4	`cin >> a;`	`46.35`	`a = 46, .35` se guarda para una entrada posterior
5	`cin >> z;`	`74.35`	`z = 74.35`
6	`cin >> z;`	`39`	`z = 39.0`
7	`cin >> z >> a;`	`65.78 38`	`z = 65.78, a = 38`

	Instrucción	Entrada	Valor almacenado en memoria
8	`cin >> a >> b;`	`4 60`	`a = 4, b = 60`
9	`cin >> a >> z;`	`46 32.4 68`	`a = 46, z = 32.4, 68` se guarda para una entrada posterior

EJEMPLO 3.2

Supongamos que se tienen las siguientes declaraciones de variables:

```
int a;
double z;
char ch;
```

Las siguientes instrucciones muestran cómo funciona el operador de extracción >>.

	Instrucción	Entrada	Valor almacenado en memoria
1	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`
2	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A', z = 26.9`
3	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A', z = 26.9`
4	`cin >> a >> ch >> z;`	`57A26.9`	`a = 57, ch = 'A', z = 26.9`

Observa que las instrucciones de entrada 1 a 4 son iguales; sin embargo, los datos entran de manera diferente. En el caso de la instrucción 1 los datos entran en la misma línea separados por blancos. En el de la instrucción 2 entran en dos líneas; los dos primeros valores de entrada están separados por dos espacios en blanco y la tercera entrada está en la siguiente línea. En el caso de la instrucción 3 los tres valores de entrada están separados por líneas, y en cuanto a la instrucción 4, los tres valores de entrada están en la misma línea, pero no hay espacio entre ellos. Nota que la segunda entrada es un carácter no numérico. Estas instrucciones funcionan de la siguiente manera.

Las instrucciones 1, 2 y 3 son fáciles de seguir. Examinemos la instrucción 4.

En la instrucción 4, primero el operador de extracción >> extrae `57` del flujo de entrada y lo almacena en `a`. Luego, el operador de extracción >> extrae el carácter `'A'` del flujo de entrada y lo almacena en `ch`. Después, se extrae `26.9` y se almacena en `z`.

Advierte que las instrucciones 1, 2 y 3 ilustran que, independientemente de que la entrada esté separada por blancos o por líneas, el operador de extracción siempre encuentra la próxima entrada.

EJEMPLO 3.3

Supongamos que se tienen las siguientes declaraciones de variables:

```
int a, b;
double z;
char ch, ch1, ch2;
```

Las siguientes instrucciones muestran cómo funciona el operador de extracción >>.

	Instrucción	Entrada	Valor almacenado en memoria
1	`cin >> z >> ch >> a;`	`36.78B34`	`z = 36.78, ch = 'B', a = 34`
2	`cin >> z >> ch >> a;`	`36.78` `B34`	`z = 36.78, ch = 'B', a = 34`
3	`cin >> a >> b >> z;`	`11 34`	`a = 11, b = 34,` la computadora espera el número contiguo
4	`cin >> a >> z;`	`78.49`	`a = 78, z = 0.49`
5	`cin >> ch >> a;`	`256`	`ch = '2', a = 56`
6	`cin >> a >> ch;`	`256`	`a = 256,` la computadora espera el valor de entrada de `ch`
7	`cin >> ch1 >> ch2;`	`A B`	`ch1 = 'A', ch2 = 'B'`

En la instrucción 1, y puesto que el primer operando a la derecha de >> es `z`, el cual es una variable `double`, `36.78` se extrae del flujo de entrada, y el valor `36.78` se almacena en `z`. Luego, se extrae `'B'` y se almacena en `ch`. Finalmente, se extrae `34` y se almacena en `a`. La instrucción 2 funciona de modo similar.

En la instrucción 3, `11` se almacena en `a` y `34` en `b`, pero el flujo de entrada no tiene suficientes datos de entrada para llenar cada variable. En este caso la computadora espera (y espera, y espera…) a que se introduzca la próxima entrada. La computadora no sigue ejecutando sino hasta que se introduce el valor que sigue.

En la instrucción 4, el primer operando a la derecha del operador de extracción >> es una variable del tipo `int` y la entrada es `78.49`. Ahora, en el caso de las variables `int`, después de introducir los dígitos del número, la lectura se detiene en el primer carácter de espacio en blanco o en un carácter que no sea un dígito. Por tanto, el operador >> almacena `78` en `a`. El siguiente operador a la derecha de >> es la variable z, la cual es del tipo `double`. Por ende, el operador >> almacena el valor `.49` como `0.49` en `z`.

En la instrucción 5, el primer operando a la derecha del operador de extracción >> es una variable `char`, por lo que el primer carácter que no es un espacio en blanco, `'2'`, se extrae del flujo de entrada. El carácter `'2'` se almacena en la variable `ch`. El siguiente operando a la derecha del operador de extracción >> es una variable `int`, por lo que el siguiente valor de entrada, `56`, se extrae y se almacena en `a`.

SEXTA EDICIÓN

Programación C++:

del Análisis de Problemas al Diseño de Programas

D.S. MALIK

Programación C++: del Análisis de Problemas al Diseño de Programas, sexta edición, sigue siendo el texto definitivo para un primer curso de lenguajes de programación. Probada por el tiempo, la metodología centrada en el estudiante desarrollada por D.S. Malik aborda ampliamente la resolución de problemas e incluye ejemplos con código completo para mostrar en forma vívida el cómo y el porqué de aplicar conceptos de programación y utilizar C++ para resolver un problema. Esta nueva edición incluye ejercicios de final de capítulo actualizados, nuevos ejercicios de depuración, una introducción a las variables desde los primeros capítulos y un análisis más moderno de las funciones definidas por el usuario para satisfacer mejor las necesidades del curso actual de ciencias de la computación.

Características del texto:

- Un contenido reorganizado presenta desde el inicio las variables y ofrece una visión actualizada de las funciones definidas por el usuario en respuesta a las necesidades de los profesores.
- Los ejercicios de final de capítulo están basados en problemas actuales y relevantes, ofreciendo amplias oportunidades para la práctica.
- El interior a color muestra con precisión la sintaxis de las instrucciones y enfatiza las palabras clave de C++ y los comentarios para programadores principiantes. Más de 300 diagramas visuales esclarecen conceptos difíciles.
- Los ejemplos numerados con código completo en todo el libro guían a los estudiantes por las etapas de entrada, salida, análisis de problemas y diseño de algoritmos para ilustrar temas clave en cada capítulo. Cada ejemplo de programación incluye una explicación completa y una corrida de muestra.

Acerca del autor

D.S. Malik es profesor de matemáticas y ciencias de la computación en Creighton University. Recibió su doctorado en Ohio University en 1985. Ha publicado más de 45 artículos especializados y 18 libros sobre álgebra abstracta, matemáticas aplicadas, teoría y lenguajes de automatización, lógica difusa y sus aplicaciones, programación, estructuras de datos y matemáticas discretas.

ISBN-13: 978-607481921-2
ISBN-10: 607481921-1

CENGAGE Learning®

Visite nuestro sitio en http://latinoamerica.cengage.com