CAPÍTULO 8

Flujos de archivos de E/S y archivos de datos

TEMAS

Los datos para los programas que se han utilizado hasta ahora se han asignado en forma interna dentro de los programas o han sido introducidos por el usuario durante la ejecución del programa. Como tales, los datos utilizados en estos programas son almacenados en la memoria principal de la computadora y dejan de existir una vez que el programa que los usa termina de ejecutarse. Este tipo de entrada de datos está bien para cantidades pequeñas de datos. Sin embargo, imagine a una compañía que tiene que pagarle a alguien para mecanografiar los nombres y direcciones de cientos o miles de clientes cada mes cuando se preparan y envían las facturas.

Como aprenderá en este capítulo, almacenar estos datos fuera de un programa en un medio de almacenamiento conveniente es más sensato. Los datos almacenados juntos bajo un nombre común en un medio de almacenamiento distinto a la memoria principal de la computadora se llaman archivos de datos. De manera

típica, los archivos de datos se almacenan en discos, cintas o CD-ROM. Además de proporcionar un almacenamiento permanente para los datos, los archivos de datos pueden ser compartidos entre programas, de modo que la salida de datos de un programa puede ser introducidos de manera directa en otro programa. Se comenzará este capítulo aprendiendo cómo se crean y mantienen los archivos de datos en C++. Una tarea muy importante sobre el uso de archivos de datos es asegurar que su programa abra y se conecte en forma correcta con ellos antes que comience cualquier procesamiento de datos. Por esta razón, aprenderá cómo usar el manejo de excepciones para esta tarea. Este tipo de detección y corrección de errores es de interés primordial en todos los programas escritos de manera profesional.

8.1 OBJETOS Y MÉTODOS EN EL FLUJO DE ARCHIVOS DE E/S

Para almacenar y recuperar datos fuera de un programa en C++, se necesitan dos cosas:

- Un archivo
- Un objeto de flujo de archivos

Aprenderá sobre estos temas importantes en las dos secciones siguientes.

Archivos

Un **archivo** es una colección de datos almacenados juntos bajo un nombre común, por lo general en un disco, cinta magnética o CD-ROM. Por ejemplo, los programas en C++ que se almacenan en un disco son ejemplos de archivos. Los datos almacenados en un archivo de programa son el código del programa que se convierte en datos de entrada para el compilador de C++. Sin embargo, En el contexto del procesamiento de datos, el programa en C++ por lo general no es considerado como datos, y el término "archivo", o "archivo de datos", por lo general sólo se refiere a archivos externos que contienen los datos usados en un programa en C++.

Un archivo se almacena en forma física en un medio externo como un disco. Cada archivo tiene un nombre de archivo único conocido como el **nombre externo** del archivo. El nombre externo es la manera en que es conocido el archivo por el sistema operativo. Cuando se revisa el contenido de un directorio o carpeta (por ejemplo, en el Explorador de Windows), se pueden observar los archivos enlistados por sus nombres externos. Cada sistema operativo de computadora tiene su propia especificación en cuanto al número máximo de caracteres permitidos para un nombre externo de archivo. La tabla 8.1 muestra estas especificaciones para los sistemas operativos más comunes.

Tabla 8.1 Cantidad máxima de caracteres permitidos en el nombre de un archivo

Sistema operativo	Largo máximo del nombre de archivo
DOS	8 caracteres más un punto opcional y una extensión de tres caracteres
Windows 98, 2000, XP	255 caracteres
UNIX Primeras versiones Versiones actuales	 14 caracteres 255 caracteres

Un poco de antecedentes

Privacidad, seguridad y archivos

Los archivos de datos existían mucho antes que se usaran las computadoras, pero se almacenaban sobre todo como registros en papel en los archiveros. Los términos como *abrir, cerrar, registros* y *consulta* que se usan en el manejo de archivos de computadora son remembranzas de las técnicas antiguas para tener acceso a archivos en papel almacenados en cajones.

En la actualidad, la mayor parte de los archivos se almacenan en forma electrónica, y la cantidad de información recopilada y almacenada prolifera. La facilidad para compartir grandes cantidades de datos en forma electrónica ha conducido al aumento en los problemas de privacidad y seguridad.

Cada vez que una persona llena un formulario del gobierno o una solicitud de crédito, envía un pedido por correo, solicita un empleo, expide un cheque o usa una tarjeta de crédito, se crea un rastro electrónico de datos. Cada vez que esos archivos son compartidos entre dependencias gubernamentales o empresas privadas, el individuo pierde algo de su privacidad.

Para ayudar a proteger los derechos constitucionales de los ciudadanos estadounidenses, se aprobó en 1970 la Ley de Información Crediticia Imparcial, seguida por la Ley Federal de Privacidad en 1974. Estas leyes especifican que es ilegal que un negocio conserve archivos secretos, que usted tiene el derecho de examinar y corregir los datos recolectados sobre usted, y que las dependencias gubernamentales y contratistas deben mostrar una justificación para tener acceso a sus registros. Continúan los esfuerzos por crear mecanismos que sirvan para preservar la seguridad y privacidad del individuo.

Para asegurar que los ejemplos presentados en este texto son compatibles con todos los sistemas operativos enlistados en la tabla 8.1, en general, pero no de manera exclusiva, nos apegamos a las especificaciones más restrictivas de DOS. Sin embargo, si usa uno de los otros sistemas operativos, deberá aprovechar las ventajas del aumento en la especificación de longitud para crear nombres de archivo descriptivos. Deberán evitarse los nombres de archivo largos debido a que requieren más tiempo para ser mecanografiados y pueden producir errores en esta tarea. Un largo manejable para un nombre de archivo es de 12 a 14 caracteres, con un máximo de 25 caracteres.

Usando la convención de DOS, todos los siguientes son nombres de archivo de datos de computadora válidos:

```
precios.dat     registro        info.txt
exper1.dat      calif.dat       fisica.mem
```

Elija nombres de archivo que indiquen el tipo de datos que hay en el archivo y la aplicación para la cual se usan. Con frecuencia, los primeros ocho caracteres describen los datos y una extensión (los caracteres después del punto decimal) describen la aplicación. Por ejemplo, el programa de hoja de cálculo Excel aplica de manera automática una extensión "xls" a todos los archivos de hoja de cálculo, los programas de procesamiento de palabras Word de Microsoft y WordPerfect usan las extensiones "doc" y "wp*x*" (donde *x* se refiere al número de la versión), respectivamente, y los compiladores de C++ requieren que un archivo de programa tenga la extensión "cpp". Cuando cree sus propios nombres de archivo, deberá observar esta práctica.

Usando la convención de DOS, el nombre "exper1.dat" es apropiado para describir un archivo de datos correspondiente al experimento número 1.

Punto de información

Flujos de entrada y salida

Un *flujo* es una ruta de transmisión en un solo sentido entre una fuente y un destino. Un flujo de bytes es enviado por esta ruta de transmisión. Una buena analogía para este flujo de bytes es un arroyo de agua que proporciona una ruta unidireccional para que el agua viaje de una fuente a un destino.

Los objetos de flujo se crean a partir de clases de flujo. Dos objetos de flujo que se han usado en forma extensa son el objeto de flujo de entrada llamado `cin` y el objeto de flujo de salida llamado `cout`. El objeto `cin` proporciona una ruta de transmisión del teclado al programa, y el objeto `cout` proporciona una ruta de transmisión del programa a la pantalla de la terminal. Estos dos objetos son creados a partir de las clases de flujo `istream` y `ostream`, respectivamente, las cuales son clases madre de la clase `iostream`. Cuando el archivo de encabezado `iostream` se incluye en un programa usando la directiva `#include <iostream>`, los objetos `cin` y `cout` son declarados y abiertos de manera automática por el compilador de C++ para el programa compilado.

Los objetos de flujo de archivos proporcionan las mismas capacidades que los objetos `cin` y `cout`, excepto que se conectan a un programa por medio de un archivo en lugar de al teclado o a la pantalla de la terminal. Los objetos de flujo de archivo deben ser declarados en forma explícita. Los objetos de flujo de archivo usados para entrada deben ser declarados como objetos de la clase `ifstream`. Los objetos de flujo de archivo usados para salida deben ser declarados como objetos de la clase `ofstream`. Las clases `ifstream` y `ofstream` se ponen a disposición de un programa por inclusión del archivo de encabezado `fstream`, usando la directiva `#include <fstream>`. La clase `fstream` se deriva de las clases `ifstream` y `ofstream` (véase la sección 8.8).

Existen dos tipos de **archivos básicos:** archivos de texto, los cuales se conocen como **archivos basados en caracteres,** y los **archivos basados en binarios**. Ambos tipos de archivos almacenan datos usando un código binario; la diferencia estriba en lo que representan los códigos. En resumen, los archivos basados en texto almacenan cada carácter individual, como una letra, dígito, signo de pesos, punto decimal, etc., usando un código de carácter individual (de manera típica ASCII o UNICODE). El uso de un código de caracteres permite a un procesador de palabras o a un editor de texto desplegar los archivos de modo que puedan ser leidos. Los archivos binarios usan el mismo código que su compilador C++ para sus tipos de datos primarios. Esto significa que los números aparecen en su forma binaria verdadera, mientras las cadenas conservan su forma ASCII o UNICODE. La ventaja de los archivos binarios es su compactibilidad debido a que se usa menos espacio para almacenar más números usando su código binario que como valores de carácter individuales. En general, la mayor parte de los archivos usados por los programadores son archivos de texto debido a que los datos del archivo pueden ser desplegados por programas de procesamiento de palabras y editores de texto simples. El tipo de archivo por omisión en C++ siempre es un archivo de texto y es el tipo de archivo que se presenta en este capítulo.

Objetos de flujo de archivos

Un **flujo de archivos** es una ruta de transmisión unidireccional utilizada para conectar un archivo almacenado en un dispositivo físico, como un disco o un CD-ROM, con un programa. Cada flujo de archivos tiene su propio método, el cual determina la dirección de los datos en la ruta de transmisión; es decir, si la ruta moverá datos de un archivo a un programa o si la ruta moverá datos de un programa a un archivo. Un flujo de archivos

que recibe o lee datos de un archivo a un programa se conoce como **flujo de archivos de entrada.** Un flujo de archivos que envía o escribe datos en un archivo se conoce como **flujo de archivos de salida.** La dirección, o modo, se define en relación con el programa y no al archivo; los datos que van a un programa se consideran datos de entrada, y los datos enviados desde el programa se consideran datos de salida. La figura 8.1 ilustra el flujo de datos desde y hacia un archivo usando flujos de entrada y de salida.

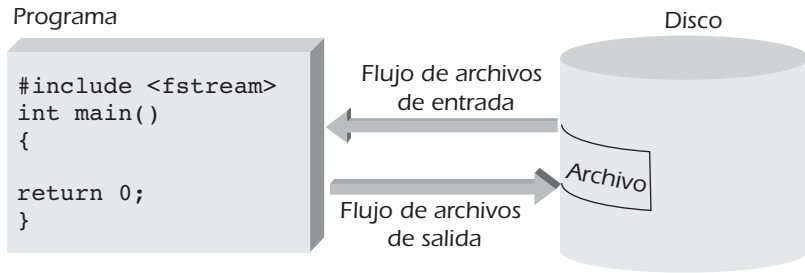


Figura 8.1 *Flujos de archivos de entrada y salida.*

Para cada archivo que utilice su programa, sin importar el tipo de archivo (texto o binario), debe crearse un objeto de flujo de archivos distinto. Si desea que su programa lea y escriba en un archivo, se requieren un objeto de flujo de archivos de entrada y uno de salida. Los objetos de flujo de archivos de entrada se declaran como tipo `ifstream`, y los flujos de archivos de salida como tipo `ofstream`. Por ejemplo, examine la siguiente instrucción de declaración:

```
ifstream archivo_entr;
```

Esta instrucción declara que un objeto de flujo de archivos de entrada llamado `archivo_entr` es un objeto de la clase `ifstream`. Del mismo modo, examine la siguiente instrucción de declaración:

```
ofstream archivo_sal;
```

Esta instrucción declara que un objeto de flujo de archivos de salida llamado `archivo_sal` es un objeto de la clase ofstream. Dentro de C++, se tiene acceso a un flujo de archivos por su nombre de objeto de flujo apropiado: un nombre para leer el archivo y un nombre para escribir en el archivo. Los nombres de objetos, como `archivo_entr` y `archivo_sal`, pueden ser cualquier nombre seleccionado por el programador que se ajuste a las reglas de identificación de C++.

Métodos de flujo de archivos

Cada objeto de flujo de archivos tiene acceso a los métodos definidos por su clase `ifstream` u `ofstream` respectiva. Estos métodos incluyen conectar un nombre de objeto de flujo a un nombre de archivo externo (llamado **abrir un archivo**), determinar si se ha hecho una conexión exitosa, cerrar una conexión (llamado **cerrar un archivo**), obtener el siguiente elemento de datos para el programa desde un flujo de entrada, colocar un elemento de datos nuevo del programa en un flujo de salida y detectar cuando se ha alcanzado el final de un archivo.

Abrir un archivo conecta cada objeto de flujo de archivos con su nombre de archivo externo específico. Esto se logra por medio de un método de apertura de flujo de archivos, el cual tiene dos propósitos. Primero, abrir un archivo que establece el vínculo de cone-

xión física entre un programa y un archivo. Dado que los detalles de este vínculo son manejados por el sistema operativo de la computadora y éstos son transparentes para el programa, por lo general el programador no necesita considerarlos.

Desde la perspectiva de codificación, el segundo propósito de abrir un archivo es más relevante. Además de establecer la conexión física real entre un programa y un archivo de datos, abrir un archivo conecta el nombre externo del archivo en la computadora con el nombre del objeto de flujo usado en forma interna por el programa. El método que realiza esta tarea se llama `open()` y es proporcionado por las clases `ifstream` y `ofstream`.

Al usar el método `open()` para conectar el nombre externo del archivo con su nombre de objeto de flujo interno, sólo se requiere un argumento, el cual es el nombre externo del archivo. Por ejemplo, examine la siguiente instrucción:

```
archivo_entr.open("precios.dat");
```

Conecta el archivo de texto externo llamado `precios.dat` con el objeto de flujo de archivo interno del programa llamado `archivo_entr`. Esto supone, por supuesto, que `archivo_entr` se ha declarado como un objeto `ifstream` u `ofstream`. Si se ha abierto un archivo con la instrucción anterior, el programa tiene acceso al archivo usando el nombre de objeto interno `archivo_entr`, y la computadora guarda el archivo bajo el nombre externo `precios.dat`. El argumento del nombre de archivo externo transmitido a `open()` es una cadena contenida entre comillas. Llamar al método `open()` requiere la notación de objeto estándar donde el nombre del método deseado, es este caso `open()`, es precedido por un punto y un nombre de objeto.

Cuando un archivo existente se conecta a un flujo de archivos de entrada, los datos del archivo quedan disponibles para entrada, empezando por el primer elemento de datos en el archivo. Del mismo modo, un archivo conectado a un flujo de archivos de salida crea un archivo nuevo y hace que el archivo quede disponible para salida. Si existe un archivo con el mismo nombre que el archivo abierto en modo de salida, el archivo antiguo es borrado y todos los datos se pierden.

Cuando se abre un archivo, para entrada o salida, la buena práctica de programación requiere que se compruebe la conexión que se ha establecido antes de intentar usar el archivo. Esto se puede hacer por medio del método `fail()`, el cual devolverá un valor verdadero si el archivo no fue abierto en forma exitosa (es decir, es verdadero en cuanto falló la apertura) o un valor `falso` si la apertura tuvo éxito. Por lo general, el método `fail()` se usa en un código similar al siguiente, el cual intenta abrir un archivo llamado `precios.dat` para entrada, comprueba que se hizo una conexión válida y reporta un mensaje de error si el archivo no fue abierto con éxito para entrada:

```
ifstream archivo_entr; // cualquier nombre de objeto puede
usarse aquí
archivo_entr.open("precios.dat"); // abre el archivo

// comprueba que la conexión se abrió con éxito
if (archivo_entr.fail())
{
  cout << "\nEl archivo no se abrió con éxito"
       << "\n Por favor compruebe que el archivo existe en
       realidad."
       << endl;
  exit(1);
}
```

Si el método `fail()` devuelve un valor booleano `verdadero`, lo cual indica que la apertura falló, el código despliega un mensaje. Además, la función `exit()` es llamada, lo cual es una solicitud para que el sistema operativo termine la ejecución del programa de inmediato. La función `exit()` requiere la inclusión de la función de encabezado `cstdlib` en cualquier programa que utilice esta función, y el argumento de un solo número entero de `exit()` es transmitido en forma directa al sistema operativo para una posible acción posterior o para inspección del usuario. A través del resto del texto se incluirá este tipo de verificación de errores siempre que se abre un archivo. (La sección 8.3 muestra cómo usar el manejo de excepciones para el mismo tipo de comprobación de errores.)

Además del método `fail()`, C++ proporciona otros tres métodos, mostrados en la tabla 8.2, que pueden utilizarse para detectar el estado de un archivo. El uso de estos métodos adicionales se presenta al final de la siguiente sección.

Tabla 8.2 Métodos de estado del archivo

Prototipo	Descripción
`fail()`	Devuelve un valor booleano `verdadero` si el archivo no se ha abierto con éxito; de lo contrario, devuelve un valor booleano `falso`.
`eof()`	Devuelve un valor booleano `verdadero` si se ha intentado leer más allá del final del archivo; de lo contrario, devuelve un valor booleano `falso`. El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido.
`good()`	Devuelve un valor booleano `verdadero` mientras el archivo está disponible para uso del programa. Devuelve un valor booleano `falso` si se ha intentado una lectura después del final del archivo. El valor se vuelve falso sólo cuando se lee el primer carácter después del último carácter de archivo válido.
`bad()`	Devuelve un valor booleano `verdadero` si se ha intentado una lectura después del final del archivo; de lo contrario, devuelve un valor `falso`. El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido.

El programa 8.1 ilustra las instrucciones requeridas para abrir un archivo de entrada, incluyendo una rutina de comprobación de errores para asegurar que se ha obtenido una apertura exitosa. Se dice que un archivo abierto para entrada está en **modo de lectura.**

Programa 8.1

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesaria para exit()
using namespace std;

int main()
{
  ifstream archivo_entr;

  archivo_entr.open("precios.dat");  // abre el archivo con el
                                     // nombre externo precios.dat
  if (archivo_entr.fail())  // comprueba una apertura exitosa
  {
    cout << "\nEl archivo no fue abierto con éxito"
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }

  cout << "\nEl archivo se ha abierto con éxito para lectura."
       << endl;

  // las instrucciones para leer datos de un archivo se colocarán aquí

  return 0;
}
```

Una muestra de la ejecución del programa 8.1 produjo la siguiente salida:

```
El archivo se ha abierto con exito para lectura.
```

Se requiere una comprobación diferente para archivos de salida porque, si existe un archivo que tenga el mismo nombre que el archivo que se va a abrir en modo de salida, el archivo existente es borrado y todos sus datos se pierden. Para evitar esto, el archivo es abierto primero en modo de entrada para ver si existe. Si existe, el usuario tiene la opción de permitir que sea sobrescrito cuando se abra más tarde en modo de salida. El código usado para lograr esto está resaltado en el programa 8.2.

Programa 8.2

```
#include <iostream>
#include <fstream>
#include <cstdlib> // necesario para exit()
using namespace std;

int main()
{
  ifstream archivo_entr;
  ofstream archivo_sal;

  archivo_entr.open("precios.dat"); // intenta abrir el archivo para entrada

  char respuesta;

  if (!archivo_entr.fail()) // si no falla, el archivo existe
  {
    cout << "Existe un archivo con el nombre precios.dat.\n"
         << "Desea continuar y sobrescribirlo\n"
         << " con los datos nuevos (si o no): ";
    cin >> respuesta;
    if (tolower(respuesta) == 'n')
    {
      cout << "El archivo existente no sera sobrescrito." << endl;
       exit(1); // termina la ejecucion del programa
    }
  }
  archivo_sal.open("precios.dat"); // ahora abre el archivo para escritura

  if (archivo_entr.fail()) // comprobar una apertura con exito
  {
    cout << "\nEl archivo no se abrió con éxito"
         << endl;
    exit(1);
  }

  cout << "El archivo se ha abierto con éxito para salida."
       << endl;

  // las instrucciones para escribir en el archivo se colocarán aquí

  return 0;
}
```

Punto de información

Uso de cadenas C como nombres de archivo

Si elige usar una cadena C para almacenar un nombre de archivo externo, debe estar enterado de las siguientes restricciones:

La longitud máxima de la cadena C debe ser especificada dentro de corchetes inmediatamente después de ser declarada. Por ejemplo, examine la siguiente declaración:

```
char nombre_archivo[21] = "precios.dat";
```

El número 21 limita el número de caracteres que pueden ser almacenada en la cadena C. El número entre corchetes (21) representa uno más que el número máximo de caracteres que pueden asignarse a la variable. Esto se debe a que el compilador agrega un carácter de fin de cadena para terminarla. Por tanto, El valor de la cadena `"precios.dat"`, el cual consiste de once caracteres, es almacenado en realidad como 12 caracteres. El carácter extra es un marcador de fin de cadena suministrado por el compilador. En nuestro ejemplo, el valor máximo de la cadena asignable al nombre de archivo de la variable de cadena es un valor de cadena consistente de 20 caracteres.

Las dos ejecuciones siguientes se hicieron con el programa 8.2:

```
Existe un archivo con el nombre precios.dat.
Desea continuar y sobrescribirlo
 con los datos nuevos (si o no): no
El archivo existente no sera sobrescrito.
```

y

```
Existe un archivo con el nombre precios.dat.
Desea continuar y sobrescribirlo
 con los datos nuevos (si o no): si
El archivo se ha abierto con éxito para salida.
```

Aunque los programas 8.1 y 8.2 pueden utilizarse para abrir un archivo existente para lectura y escritura, respectivamente, ambos programas carecen de instrucciones para llevar a cabo una lectura o escritura y cerrar el archivo. Estos temas se expondrán en breve. Antes de continuar, hay que anotar que es posible combinar la declaración de un objeto `ifstream` u `ofstream` y su instrucción de apertura asociada en una instrucción. Por ejemplo, observe las dos instrucciones siguientes en el programa 8.1:

```
ifstream archivo_entr;
archivo_entr.open("precios.dat");
```

Pueden combinarse en una sola instrucción:

```
ifstream archivo_entr("precios.dat");
```

Nombres de archivo incrustados e interactivos

Los programas 8.1 y 8.2 tienen dos problemas:

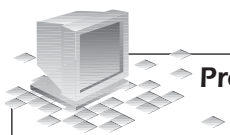
1. El nombre de archivo externo está incrustado dentro del código del programa.
2. No está previsto que un usuario introduzca el nombre de archivo deseado mientras se está ejecutando el programa.

Tal y como están escritos ambos programas, si se va a cambiar el nombre del archivo, el programador debe modificar el nombre de archivo externo en la llamada a `open()` y volver a compilar el programa. Ambos problemas pueden ser resueltos asignando el nombre de archivo a una variable de cadena.

Una variable de cadena como se ha usado a travéz del texto (véase el capítulo 9) es una variable que puede contener un valor de cadena, el cual es cualquier secuencia de cero o más caracteres encerrados entre comillas. Por ejemplo, `"Hola mundo"`, `"precios.dat"` y " " son cadenas. Hay que observar que las cadenas se escriben con comillas que delimitan el inicio y el final de una cadena pero no son almacenadas como parte de la cadena.

Al declarar e inicializar una variable de cadena para usarla en un método `open()`, la cadena es considerada como una cadena C. (Véase el Punto de información en la página anterior para las precauciones que deben tomarse cuando se use una cadena C.) Una alternativa más segura, y que se usará a tarvés de este texto, es usar un objeto de clase de cadena y convertir este objeto en una cadena C utilizando el método `c_str()`.

Una vez que se declara una variable de cadena para almacenar un nombre de archivo, puede utilizarse en una de dos formas. Primera, como se muestra en el programa 8.3a, puede colocarse en la parte superior de un programa para identificar con claridad el nombre externo de un archivo, en lugar de incrustarlo dentro de una llamada al método `open()`.



Programa 8.3a

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string nombre_archivo = "precios.dat"; // coloca el nombre de archivo al frente
  ifstream archivo_entr;

  archivo_entr.open(nombre_archivo.c_str()); // abre el archivo

  if (archivo_entr.fail()) // comprueba una apertura con exito
  {
    cout << "\nEl archivo llamado " << nombre_archivo << " no se abrió con éxito
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }

  cout << "\nEl archivo se ha abierto con éxito para lectura.\n";

  return 0;
}
```

El programa 8.3a muestra que se ha declarado e inicializado el objeto de cadena con el siguiente nombre:

```
nombre_archivo
```

Se usa este nombre al principio de `main()` para una fácil identificación del archivo. Cuando se usa un objeto de cadena, en oposición a una literal de cadena, el nombre de la variable no se encierra entre comillas en la llamada al método `open()`. Dentro de la llamada `open()`, el objeto de cadena es convertido en una cadena C usando la siguiente expresión:

```
nombre_archivo.c_str().
```

Por último, en el código del método `fail()` el nombre externo del archivo es desplegado con facilidad al insertar el nombre del objeto de cadena en el flujo de salida `cout`. Por estas razones, se continuará identificando los nombres externos de los archivos de esta manera.

Otro papel útil que desempeñan los objetos de cadena es permitir al usuario introducir el nombre del archivo mientras se está ejecutando el programa. Por ejemplo, el código:

```
string nombre_archivo;

cout << "Por favor introduzca el nombre del archivo que
desea abrir: ";
cin >> nombre_archivo;
```

permite a un usuario introducir un nombre externo de archivo en tiempo de ejecución. La única restricción en este código es que el usuario no debe encerrar el valor de cadena introducido entre comillas, y el valor de cadena introducido no puede contener ningún espacio en blanco. La razón para ello que cuando se usa `cin`, el compilador terminará la cadena cuando encuentre un espacio en blanco. El programa 8.3b usa este código en el contexto de un programa completo.

La siguiente es una salida de muestra proporcionada por el programa 8.3b:

```
Por favor introduzca el nombre del archivo que desea
abrir: foobar

El archivo llamado foobar no se abrió con éxito
 Por favor compruebe que el archivo existe en realidad.
```

Cerrar un archivo

Un archivo se cierra usando el método `close()`. Este método rompe la conexión entre el nombre externo del archivo y el objeto de flujo de archivos, el cual puede ser usado por otro archivo. Examine la siguiente instrucción:

```
archivo_entr.close();
```

Esta instrucción cierra la conexión del flujo `archivo_entr` con su archivo actual. Como se indica, el método `close()` no utiliza un argumento.

Debido a que todas las computadoras tienen un límite en el número máximo de archivos que pueden abrirse a la vez, cerrar archivos que ya no son necesarios tiene sentido. Los archivos abiertos existentes al final de la ejecución del programa normal serán cerrados de manera automática por el sistema operativo.

Programa 8.3b

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // necesario para exit()
#include <string>
using namespace std;
int main()
{
  string nombre_archivo;
  ifstream archivo_entr;

  cout << "Por favor introduzca el nombre del archivo que desea abrir: ";
  cin >> nombre_archivo;

  archivo_entr.open(nombre_archivo.c_str());  // abre el archivo

  if (archivo_entr.fail())  // comprueba una apertura con éxito
  {
    cout << "\nEl archivo llamado " << nombre_archivo << " no se abrio con exito
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }
  cout << "\nEl archivo se abrió con éxito para lectura.\n";

  return 0;
}
```

Punto de información

Uso de objetos `fstream`

Al usar objetos `ifstream` y `ofstream`, el modo de entrada o salida está implicado por el objeto. Por tanto, los objetos `ifstream` deben usarse para entrada y los objetos `ofstream` deben utilizarse para salida.

Otro medio para crear flujos de archivos es usar objetos `fstream` que puedan utilizarse para entrada o salida, pero este método requiere una designación explícita. Un objeto `fstream` se declara usando la siguiente sintaxis:

```
fstream nombreObjeto;
```

Cuando se usa el método `open()` de la clase `fstream()`, se requieren dos argumentos: un nombre externo de archivo y un indicador de modo. Aquí se muestran los indicadores de modo permisibles:

Indicador	Descripción
`ios::in`	Abre un archivo de texto en modo de entrada
`ios::out`	Abre un archivo de texto en modo de salida
`ios::app`	Abre un archivo de texto en modo anexar
`ios::ate`	Va al final del archivo abierto
`ios::binary`	Abre un archivo binario en modo de entrada (es archivo de texto el valor por omisión)
`ios::trunc`	Elimina el contenido del archivo si existe
`ios::nocreate`	Si el archivo no existe, la apertura falla
`ios::noreplace`	Si el archivo existe, falla la apertura para salida

Como con los objetos `ofstream`, un objeto `fstream` en modo de salida crea un archivo nuevo y hace que el archivo esté disponible para escritura. Si existe un archivo con el mismo nombre que el archivo abierto para salida, el archivo antiguo es borrado. Por ejemplo, suponga que se ha declarado `archivo1` como un objeto de tipo `fstream` usando la siguiente instrucción:

```
fstream archivo1;
```

La instrucción que sigue intenta abrir el archivo de texto nombrado `precios.dat` para salida:

```
archivo1.open("precios.dat"),ios::out);
```

Una vez que se ha abierto este archivo, el programa tiene acceso al archivo usando el nombre de objeto interno `archivo1`, y la computadora guarda el archivo bajo el nombre externo `precios.dat`.

Un objeto de archivo `fstream` abierto en modo anexar significa que un archivo existente está disponible para que se anexen datos al final del archivo. Si el archivo abierto para anexar no existe, se crea un archivo nuevo con el nombre designado y se hace disponible para recibir salida del programa. Por ejemplo, suponga que se ha declarado que `archivo1` es del tipo `fstream`:

```
archivo1.open("precios.dat",ios::app);
```

La instrucción anterior intenta abrir un archivo de texto llamado `precios.dat` y lo hace disponible para que se anexen datos al final del archivo.

Por último, un objeto `fstream` abierto en modo de entrada significa que se ha conectado un archivo externo existente y que sus datos están disponibles como entrada. Por ejemplo, suponga que se ha declarado que `archivo1` es del tipo `fstream`

```
archivo1.open("precios.dat",ios::in);
```

(Continúa)

(Continuación)

La instrucción anterior intenta abrir un archivo de texto llamado `precios.dat` para entrada. Los indicadores de modo pueden combinarse con la operación de bit O (véase la sección 15.2).

```
archivo1.open("precios.dat", ios::in | ios::binary)
```

La instrucción anterior abre el flujo `archivo1`, el cual puede ser un `fstream` o `ifstream`, como un flujo binario de entrada. Si se omite el indicador de modo como el segundo argumento para un objeto `ifstream`, el flujo es abierto, por omisión, como un archivo de entrada de texto; si se omite el indicador de modo para un objeto `ofstream`, el flujo es abierto, por omisión, como un archivo de salida de texto.

Ejercicios 8.1

1. Escriba instrucciones de declaración y apertura individuales que vinculen los siguientes nombres de archivos de datos externos con sus correspondientes nombres de objeto internos. Suponga que todos los archivos se basan en texto.

Nombre externo	Nombre de objeto	Modo
`coba.mem`	`memo`	`salida`
`libro.let`	`carta`	`salida`
`cupones.bnd`	`cupones`	`anexar`
`produce.bnd`	`produce`	`anexar`
`precios.dat`	`archivo_precios`	`entrada`
`indices.dat`	`indices`	`entrada`

2. a. Escriba un conjunto de dos instrucciones que declare primero los siguientes objetos como objetos `ifstream` y luego los abra como archivos de entrada de texto: `enDatos.txt`, `precios.txt`, `cupones.dat` y `exper.dat`.
b. Vuelva a escribir las dos instrucciones para el ejercicio 2a usando una sola instrucción.

3. a. Escriba un conjunto de dos instrucciones que declaren primero los siguientes objetos como objetos `ofstream` y luego los abra como archivos de salida de texto: `salFecha.txt`, `indices.txt`, `trecho.txt` y `archivo2.txt`.
b. Vuelva a escribir las dos instrucciones para el ejercicio 3a usando una sola instrucción.

4. Introduzca y ejecute el programa 8.1 en su computadora.

5. Introduzca y ejecute el programa 8.2 en su computadora.

6. a. Introduzca y ejecute el programa 8.3a en su computadora.
b. Agregue un método `close()` al programa 8.3a y luego ejecute el programa.

7. a. Introduzca y ejecute el programa 8.3b en su computadora.
b. Agregue un método `close()` al programa 8.3b y luego ejecute el programa.

Punto de información

Comprobar una conexión exitosa

Debe comprobarse que el método `open()` estableció con éxito una conexión entre un flujo de archivo y un archivo externo. Esto se debe a que la llamada `open()` es una solicitud al sistema operativo que puede fallar por varias razones. La principal de estas razones puede ser una solicitud para abrir un archivo existente para lectura que el sistema operativo no puede localizar o una solicitud para abrir un archivo para salida en una carpeta inexistente. Si el sistema operativo no puede satisfacer la solicitud de apertura, usted necesita saberlo y terminar el programa. No hacerlo así puede producir un comportamiento anormal del programa o una caída subsiguiente de éste.

Hay dos estilos de codificación para comprobar el valor devuelto. El método más común para comprobar que no ocurrió una falla cuando se intenta usar un archivo de entrada es el codificado en el programa 8.1. Se usa para distinguir la solicitud `open()` de la comprobación hecha por medio de la llamada `fail()` y a continuación se repite por comodidad:

```
archivo_entr.open("precios.dat"); // solicitud para abrir
el archivo

if (archivo_entr.fail()) // comprueba una conexión fallida
{
  cout << "\nEl archivo no se abrió con éxito"
       << "\n Por favor compruebe que el archivo existe
          en realidad."
       << endl;
  exit(1);
}
```

Del mismo modo, la comprobación hecha en el programa 8.2 se incluye por lo general cuando se está abriendo un archivo en modo de salida.

De manera alternativa, pueden encontrarse programas que usan objetos `fstream` en lugar de objetos `ifstream` y `ofstream` (véase el recuadro Punto de información anterior). Cuando se usa el método `open()` de `fstream` se requieren dos argumentos: un nombre externo del archivo y una indicación explícita del modo. Usando un objeto `fstream` la solicitud para abrir y comprobar un archivo de entrada por lo general aparece como sigue:

```
fstream archivo_entr;

archivo_entr.open("nombre de archivo externo", ios::in);
if (archivo_entr.fail())
{
  cout << "\nEl archivo no se abrió con éxito"
       << "\n Por favor compruebe que el archivo existe
          en realidad."
       << endl;
  exit(1);
}
```

Muchas veces la expresión condicional `archivo_entr.fail()` es reemplazada por la expresión equivalente `!archivo_entr`. Aunque nosotros siempre usamos objetos `ifstream` y `ofstream`, debe estar preparado para encontrar los estilos que usan objetos `fstream`.

8. Usando los manuales de referencia proporcionados con el sistema operativo de su computadora, determine lo siguiente:
   a. el número máximo de caracteres que pueden usarse para nombrar un archivo para almacenamiento en el sistema de su computadora
   b. el número máximo de archivos de datos que pueden abrirse al mismo tiempo
9. ¿Sería apropiado llamar archivo a un programa en C++ guardado? ¿Por qué sí o por qué no?
10. a. Escriba instrucciones de declaración y apertura individuales para vincular los siguientes nombres de archivo de datos externos con sus nombres de objeto internos correspondientes. Use sólo objetos `ifstream` y `ofstream`.

Nombre externo	Nombre de objeto	Modo
`coba.mem`	`memo`	`binario y salida`
`cupones.bnd`	`cupones`	`binario y anexar`
`precios.dat`	`archivo_precios`	`binario y entrada`

   b. Vuelva a hacer el ejercicio 10a usando sólo objetos `fstream`.
   c. Escriba instrucciones de cierre para cada uno de los archivos abiertos en el ejercicio 10a.

8.2 LECTURA Y ESCRITURA DE ARCHIVOS BASADOS EN CARACTERES

Leer o escribir en archivos basados en caracteres implica las operaciones casi idénticas para leer la entrada realizada desde un teclado y escribir datos a una pantalla de despliegue. Para escribir en un archivo, el objeto `cout` es reemplazado por el nombre de objeto `ofstream` declarado en el programa. Por ejemplo, si `archivo_sal` es declarado como un objeto de tipo `ofstream`, las siguientes instrucciones de salida son válidas:

```
archivo_sal << 'a';
archivo_sal << "¡Hola mundo!";
archivo_sal << descrip << ' ' << precio;
```

El nombre de archivo en cada una de estas instrucciones, en lugar de `cout`, dirige el flujo de salida a un archivo específico en lugar de al dispositivo de despliegue estándar. El programa 8.4 ilustra el uso del operador de inserción, `<<`, para escribir una lista de descripciones y precios en un archivo.

Programa 8.4

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
#include <iomanip>   // necesario para formatear
using namespace std;

int main()
{
  string nombre_archivo = "precios.dat"; // pone al frente el nombre de archivo
  ofstream archivo_sal;

  archivo_sal.open(nombre_archivo.c_str());

  if (archivo_sal.fail())
  {
    cout << "El archivo no se abrió con éxito" << endl;
    exit(1);
  }

  // establece los formatos del flujo de archivo de salida
  archivo_sal << setiosflags(ios::fixed)
              << setiosflags(ios::showpoint)
              << setprecision(2);

  // envía datos al archivo
  archivo_sal << "Alfombras " << 39.95 << endl
              << "Bombillas " << 3.22 << endl
              << "Fusibles " << 1.08 << endl;

  archivo_sal.close();
  cout << "El archivo " << nombre_archivo
       << " se ha escrito con éxito." << endl;

  return 0;
}
```

Punto de información

Formatear los datos del flujo de salida de un archivo de texto

Los flujos de archivo de salida pueden formatearse de la misma manera que el flujo de salida cout estándar. Por ejemplo, si se ha declarado un flujo de salida nombrado `salidaArchivo`, la siguiente instrucción formatea todos los datos insertados en el flujo `salidaArchivo` en la misma forma en que trabajan estos manipuladores parametrizados para el flujo `cout`:

```
salidaArchivo << setiosflags(ios::fixed)
              << setiosflags(ios::showpoint)
              << setprecision(2);
```

El primer parámetro manipulador, `ios::fixed`, causa que el flujo dé salida a todos los números como si fueran valores de punto flotante. El siguiente parámetro, `ios::showpoint`, le indica al flujo que proporcione un punto decimal. Por tanto, un valor como 1.0 aparecerá como 1.0, y no 1. Por último, el manipulador `setprecision` le indica al flujo que despliegue dos valores decimales después del punto decimal. Por tanto, el número 1.0, por ejemplo, aparecerá como 1.00.

En lugar de manipuladores, también pueden usarse los métodos de flujo `setf()` y `precision()`. Por ejemplo, el formato anterior puede elaborarse usando el siguiente código:

```
salidaArchivo.setf(ios::fixed);
salidaArchivo.setf(ios::showpoint);
salidaArchivo.precision(2);
```

El estilo que se elija es cuestión de preferencia. En ambos casos, sólo es necesario especificar los formatos una vez, y permanecen en efecto para todos los números insertados después en el flujo de archivo.

Cuando se ejecuta el programa 8.4, se crea un archivo, `precios.dat`, y es guardado por la computadora como un archivo de texto. El archivo es un archivo secuencial consistente en los siguientes datos:

```
Alfombras 39.95
Bombillas 3.22
Fusibles 1.08
```

El almacenamiento real de caracteres en el archivo depende de los códigos de carácter usados por la computadora. Aunque sólo parecen estar almacenados 42 caracteres en el archivo, correspondientes a las descripciones, espacios en blanco y precios escritos en el archivo, el archivo contiene 48 caracteres.

Los caracteres extra consisten en la secuencia de escape de línea nueva al final de cada línea creada por el manipulador `endl`, el cual es creado como un carácter de retorno de carro (`cr`) y avance de línea (`lf`). Suponiendo que los caracteres son almacenados usando el código ASCII, el archivo `precios.dat` está almacenado en forma física como se ilustra en la figura 8.2. Por comodidad, el carácter correspondiente a cada código hexadecimal se enlista abajo del código. Un código de 20 representa el carácter de espacio en blanco. Además, C y C++ anexan el byte hexadecimal de valor bajo 0x00 como el centinela de fin de archivo (EOF) cuando se cierra el archivo. Este centinela EOF nunca se cuenta como parte del archivo.

Punto de información

El método `put()`

Todos los flujos de salida tienen acceso al método `put()` de la clase `fstream`, el cual permite la salida carácter por carácter a un flujo. Este método funciona de la misma manera que el operador de inserción de carácter, <<. La sintaxis de esta llamada al método es la siguiente:

```
ofstreamNombre.put(expresión-en-carácter);
```

La *`expresión-en-carácter`* puede ser una variable de carácter o un valor literal. Por ejemplo, el siguiente código puede usarse para dar salida a una 'a' al flujo de salida estándar:

```
cin.put('a');
```

De una manera similar, si `archivo_sal` es un archivo de objeto `ofstream` que se ha abierto, el siguiente código envía el valor de carácter en la variable de carácter llamada `codigoclave` a esta salida:

```
char codigoclave
  .
  .
archivo_sal.put(codigoclave);
```

```
46 75 73 69 62 6C 65 73 20 33 39 2E 39 35 0D 0A 42 6F 6D 62 69 6C 6C 61 73 20
 F  u  s  i  b  l  e  s     3  9  .  9  5 cr 1f  B  o  m  b  i  l  l  a  s

33 2E 32 32 0D 0A 41 6C 66 6F 6D 62 72 61 73 20 31 2E 30 38 0D 0A
 3  .  2  2 cr 1f  A  l  f  o  m  b  r  a  s     1  .  0  8 cr 1f
```

Figura 8.2 *El archivo* `precios.dat` *tal como es almacenado por la computadora.*

Lectura de un archivo de texto

Leer datos de un archivo basado en texto es casi igual a leer datos de un teclado estándar, excepto que el objeto `cin` es reemplazado por el objeto `ifstream` declarado en el programa. Por ejemplo, si `archivo_entr` es declarado como un objeto de tipo `ifstream` que es abierto para entrada, la entrada que sigue a la instrucción leerá los siguientes dos elementos en el archivo y los almacenará en las variables `descrip` y `precio`:

```
archivo_entr >> descrip >> precio;
```

El nombre del flujo de archivos en esta instrucción, en lugar de `cin`, instruye que la entrada provenga del flujo de archivo en lugar de venir del flujo del dispositivo de entrada estándar. Otros métodos que pueden usarse para flujo de entrada se muestran en la tabla 8.3. Cada uno de estos métodos debe ser precedido por un nombre de objeto de flujo.

Tabla 8.3 Métodos `fstream`

Nombre del método	Descripción
`get()`	Devuelve el siguiente carácter extraído del flujo de entrada como un `int`.
`get(charVar)`	Versión sobrecargada de `get()` que extrae el siguiente carácter del flujo de entrada y lo asigna a la variable de carácter especificada, `charVar`.
`getline( strObj, termChar)`	Extrae caracteres del flujo de entrada especificado, `strObj`, hasta que se encuentra el carácter de terminación, `termChar`. Asigna los caracteres al objeto de clase de cadena especificado, `strObj`.
`peek()`	Devuelve el siguiente carácter en el flujo de entrada sin extraerlo del flujo.
`ignore(int n)`	Se salta los siguientes *n* caracteres. Si se omite `n`, el valor por omisión es saltarse el siguiente carácter individual.

El programa 8.5 ilustra cómo puede leerse el archivo `precios.dat` que fue creado en el programa 8.4. El programa ilustra un método para detectar el marcador EOF usando la función `good()` (véase la tabla 8.2). Debido a que esta función devuelve un valor booleano verdadero antes que se haya leído o transmitido el marcador EOF, puede utilizarse para verificar que los datos leídos son datos de archivo válidos. Sólo después que se ha leído o transmitido el marcador EOF esta función devuelve un valor booleano `falso`. Por tanto, la notación `while(archivo_entr.good())` usada en el programa 8.5 asegura que los datos pertenecen al archivo antes que se haya leído EOF.

El despliegue producido por el programa 8.5 es el siguiente:

```
Alfombras 39.95
Bombillas 3.22
Fusibles 1.08
```

Examine la expresión `archivo_entr.good()` usada en la instrucción while. Esta expresión es verdadera en tanto el marcador EOF no haya sido leído. Por tanto, mientras la lectura del elemento fue buena, el ciclo continúa leyendo el archivo. Dentro del ciclo, los elementos que se acaban de leer son desplegados, y luego una cadena nueva y un número de precisión doble son introducidos al programa. Cuando se ha detectado el EOF, la expresión devuelve un valor booleano de `falso` y el ciclo termina. Esto asegura que los datos son leídos y desplegados hasta el marcador EOF, pero sin incluirlo.

Un reemplazo directo para la instrucción `while(archivo_entr.good())` es la instrucción `while(!archivo_entr.eof())`, la cual se lee "mientras el final del archivo no se ha alcanzado". Esto funciona porque la función `eof()` devuelve un valor booleano verdadero sólo después que se ha leído o transmitido el marcador EOF. En efecto, la expresión relacional comprueba que el EOF no ha sido leído; de ahí el uso del operador NOT, `!`.

Otro medio de detectar el EOF es usar el hecho que la operación de extracción, `>>`, devuelve un valor booleano `verdadero` si los datos son extraídos de un flujo; de lo contrario, devuelve un valor booleano `falso`. Usando este valor devuelto, puede utilizarse el siguiente código dentro del programa 8.5 para leer el archivo.

Programa 8.5

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string nombre_archivo = "precios.dat"; // pone al frente el nombre de archivo
  string descrip;
  double precio;

  ifstream archivo_entr;

  archivo_entr.open(nombre_archivo.c_str());

  if (archivo_entr.fail()) // comprueba el éxito en la apertura
  {
    cout << "\nEl archivo no se abrió con éxito"
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }

  // lee y despliega el contenido del archivo
  archivo_entr >> descrip >> precio;
  while (archivo_entr.good()) // comprueba el siguiente carácter
  {
    cout << descrip << ' ' << precio << endl;
    archivo_entr >> descrip >> precio;archivo_entr >> descrip >> precio;
  }

  inFile.close();

  archivo_entr.close();
}
```

Punto de información

Una forma de identificar el nombre y ubicación de un archivo

Durante el desarrollo del programa, los archivos de prueba se colocan por lo general en el mismo directorio que el programa. Por consiguiente, una llamada al método como `archivo_entr.open ("exper.dat")` no causa problemas al sistema operativo. En sistemas de producción, sin embargo, no es poco común que los archivos de datos residan en un directorio mientras los archivos del programa residen en otro. Por esta razón siempre es una buena idea incluir el nombre de la ruta completa de cualquier archivo abierto.

Por ejemplo, si el archivo `exper.dat` reside en el directorio `C:\prueba\archivos`, la llamada `open()` deberá incluir el nombre de la ruta completa: `archivo_entr.open("C:\\prueba \\archivos\\exper.dat")`. Entonces, sin importar desde dónde se ejecute el programa, el sistema operativo sabrá dónde localizar el archivo. Observe el uso de diagonales invertidas dobles, lo cual es necesario.

Otra convención importante es enlistar todos los nombres de archivo al principio del programa en lugar de incrustar los nombres en las profundidades dentro del código. Esto puede lograrse con facilidad con variables de cadena para almacenar cada nombre de archivo.

Por ejemplo, si las instrucciones:

```
string nombre_archivo = "c:\\prueba\\archivos\\exper.dat";
```

se colocan al principio de un archivo de programa, la instrucción de declaración enlista con claridad tanto el nombre del archivo deseado como su ubicación. Luego, si se va a probar algún otro archivo, todo lo que se requiere es un cambio simple de una línea al principio del programa.

Usar una variable de cadena para el nombre del archivo también es útil para la comprobación del método `fail()`. Por ejemplo, considere el siguiente código:

```
string nombre_archivo;
ifstream archivo_entr;

archivo_entr.open(nombre_archivo.c_str());

if (archivo_entr.fail())
{
  cout << "\n El archivo nombrado " << nombre_archivo
       << " no se abrió con éxito"
       << "\n Por favor compruebe que este archivo existe
          en realidad."
       exit(1);
}
```

En este código, el nombre del archivo que falló en abrirse es desplegado en forma directa dentro del mensaje de error sin que el nombre esté incrustado como un valor de cadena.

```
// leer y desplegar el contenido del archivo
  while (archivo_entr >> descrip >> precio) // comprueba
  el siguiente carácter
    cout << descrip << ' ' << precio << endl;
```

Aunque al principio es un poco críptico, este código tiene sentido perfecto cuando se entiende que la expresión que se está probando extrae datos del archivo y devuelve un valor booleano para indicar si la extracción fue exitosa.

Por último, en la instrucción `while` previa o en el programa 8.5, la expresión `archivo_entr >> descrip >>` precio puede reemplazarse con un método `getline()` (véase la sección 7.2). Para entrada de archivo, este método tiene la siguiente sintaxis:

```
getline(objetoArchivo, strObj, carácter-de-terminación)
```

objetoArchivo es el nombre del archivo *ifstream*, *strObj* es un objeto de la clase de cadena y *carácter-de-terminación* es una constante o variable de carácter opcional que especifica el carácter de terminación. Si se omite este tercer argumento opcional, el carácter de terminación por omisión es el carácter de línea nueva (`'\n'`). El programa 8.6 ilustra el uso de *getline()* dentro del contexto de un programa completo.

Programa 8.6

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string nombre_archivo = "precios.dat"; // pone al frente el nombre de archivo
  string linea;
  ifstream archivo_entr;

  archivo_entr.open(nombre_archivo.c_str());

  if (archivo_entr.fail()) // comprueba que se abrió con éxito
  {
    cout << "\nEl archivo no se abrió con éxito"
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }

  // lee y despliega el contenido del archivo
  while (getline(archivo_entr,linea))
    cout << linea << endl;

  archivo_entr.close();

  return 0;
}
```

Punto de información

Los métodos `get()` y `putback()`

Todos los flujos de entrada tienen acceso al método `get()` de la clase fstream, el cual permite la entrada carácter por carácter desde un flujo de entrada. Este método funciona de manera similar a la extracción de carácter usando el operador >>, con dos diferencias importantes: si se encuentra un carácter de línea nueva, `'\n'`, o un carácter de espacio en blanco, `' '`, cada uno de estos caracteres es leído de la misma manera que cualquier otro carácter alfanumérico. La sintaxis de la llamada a este método es la siguiente:

```
istreamNombre.get(variable-de-carácter);
```

Por ejemplo, puede usarse el siguiente código para leer el siguiente carácter del flujo de entrada estándar y almacenar el carácter en la variable `ch`:

```
char ch;
cin.get(ch);
```

De manera similar, si `archivo_entr` es un objeto `ifstream` que se ha abierto a un archivo, el siguiente código lee el siguiente carácter en el flujo y lo asigna al carácter `codigoclave`:

```
char codigoclave;
archivo_entr.get(codigoclave);
```

Además del método `get()`, todos los flujos de entrada tienen un método `putback()` que puede utilizarse para poner el último carácter leído de un flujo de entrada de vuelta en el flujo. Este método tiene la siguiente sintaxis:

```
ifstreamNombre.putback(expresión-de-carácter);
```

`expresión-de-carácter` puede ser cualquier variable de carácter o valor de carácter.

El método `putback()` proporciona una capacidad de salida a un flujo de entrada. El carácter `putback` no necesita ser el último carácter leído; más bien, puede ser cualquier carácter. Sin embargo, todos los caracteres de `putback` no tienen efecto en el archivo de datos sino sólo en el flujo de entrada abierto. Por tanto, los caracteres del archivo de datos permanecen sin cambios aunque puedan cambiar los caracteres leídos en lo subsiguiente desde el flujo de entrada.

El programa 8.6 es un programa de copiado de texto línea por línea, el cual lee una línea de texto desde el archivo y luego la despliega en la terminal. La salida del programa 8.6 es la siguiente:

```
Alfombras 39.95
Bombillas 3.22
Fusibles 1.08
```

Si fuera necesario obtener la descripción y el precio como variables individuales, debería utilizarse el programa 8.5 o procesarse más la cadena devuelta por `getline()` en el programa 8.6 para extraer los elementos de datos individuales. (Véase la sección 8.7 para procedimientos de análisis sintáctico.)

Archivos estándar en dispositivos

Los objetos de flujo de archivos que se han usado han sido objetos de archivo lógicos. Un objeto de archivo lógico es un flujo que conecta un archivo de datos relacionados lógicamente, como un archivo de datos, con un programa. Además de objetos de archivo lógicos, C++ soporta objetos de archivo físicos. Un objeto de archivo físico es un flujo que conecta a un dispositivo de hardware, como un teclado, pantalla o impresora.

El dispositivo físico asignado a su programa para la introducción de datos se llama de manera formal **archivo estándar de entrada**. Por lo general, éste es el teclado. Cuando se encuentra una llamada al método de objeto `cin` en un programa en C++, es una solicitud al sistema operativo para que vaya a este archivo de entrada estándar por la entrada esperada. Del mismo modo, cuando se encuentra una llamada al método de objeto `cout`, la salida se despliega de manera automática o se "escribe" en un dispositivo que ha sido asignado como el **archivo estándar de salida**. Para la mayor parte de los sistemas, éste es una pantalla de computadora, aunque puede ser una impresora.

Cuando se ejecuta un programa, el flujo de entrada estándar `cin` es conectado con el dispositivo estándar de entrada. Del mismo modo, el flujo de salida estándar `cout` es conectado con el dispositivo estándar de salida. Estos dos flujos de objetos están disponibles para uso del programador, como lo están el flujo de error estándar, `cerr`, y el flujo de registro estándar, `clog`. Estos dos flujos se conectan con la pantalla de la terminal.

Otros dispositivos

Los flujos de teclado, despliegue, reporte de errores y registro son conectados en forma automática con los objetos de flujo llamados `cin`, `cout`, `cerr` y `clog`, respectivamente, cuando se incluye el archivo de encabezado `iostream` en un programa. Pueden utilizarse otros dispositivos para entrada o salida si se conoce el nombre asignado por el sistema. Por ejemplo, la mayor parte de las computadoras personales IBM o compatibles con IBM asignan el nombre `prn` a la impresora conectada a la computadora. Para estas computadoras, una instrucción como `archivo_sal.open("prn")` conecta la impresora al objeto `ofstream` llamado `archivo_sal`. Una instrucción subsiguiente, como `archivo_sal << "¡Hola mundo!";` causaría que la cadena `¡Hola mundo!` saliera en forma directa a la impresora. Como el nombre de un archivo real, `prn` debe encerrarse entre comillas en la llamada a la función `open()`.

Ejercicios 8.2

1. **a.** Introduzca y ejecute el programa 8.5.
   b. Modifique el programa 8.5 para usar la expresión `!archivo_entr.eof()` en lugar de la expresión `archivo_entr.good()`, y ejecute el programa para ver que opera en forma correcta.

2. **a.** Introduzca y ejecute el programa 8.6.
   b. Modifique el programa 8.6 reemplazando el identificador `cout` con `cerr`, y verifique que la salida para el flujo de archivo de error estándar es la pantalla.
   c. Modifique el programa 8.6 reemplazando el identificador cout con `clog`, y verifique que la salida para el flujo de registro estándar es la pantalla.

3. a. Escriba un programa en C++ que acepte líneas de texto del teclado y escriba cada línea en un archivo llamado `texto.dat` hasta que se introduzca una línea vacía. Una línea vacía es una línea sin texto que se crea oprimiendo la tecla Entrar (o Retorno).

b. Modifique el programa 8.6 para leer y desplegar los datos almacenados en el archivo `texto.dat` creado en el ejercicio 3a.

4. Determine el comando o procedimiento del sistema operativo proporcionado por su computadora para desplegar el contenido de un archivo guardado.

5. a. Cree un archivo de texto nombrado `empleado.dat` que contenga los siguientes datos:

Anthony	A	10031	7.82	12/18/05
Burrows	W	10067	9.14	6/ 9/04
Fain	B	10083	8.79	5/18/04
Janney	P	10095	10.57	9/28/04
Smith	G	10105	8.50	12/20/03

b. Escriba un programa en C++ para leer el archivo `empleado.dat` creado en el ejercicio 5a y producir una copia duplicada del archivo llamada `empleado.bak`.

c. Modifique el programa escrito en el ejercicio 5b para que acepte los nombres de los archivos original y duplicado como entrada del usuario.

d. El programa escrito para el ejercicio 5c siempre copia datos de un archivo original a un archivo duplicado. ¿Cuál es un mejor método para aceptar los nombres de archivo original y duplicado, otro que no sea indicar al usuario que los introduzca cada vez que se ejecute el programa?

6. a. Escriba un programa en C++ que abra un archivo y despliegue el contenido del archivo con números de línea asociados. Es decir, el programa deberá imprimir el número 1 antes de desplegar la primera línea, imprimir el número 2 antes de desplegar la segunda línea, y así en forma sucesiva para cada línea en el archivo.

b. Modifique el programa escrito en el ejercicio 6a para enlistar el contenido del archivo en la impresora asignada a su computadora.

7. a. Cree un archivo de texto que contenga los siguientes datos (sin los encabezados):

Nombres	Número de seguro social	Tarifa por hora	Horas trabajadas
B Caldwell	555-88-2222	7.32	37
D Memcheck	555-77-4444	8.32	40
R Potter	555-77-6666	6.54	40
W Rosen	555-99-8888	9.80	35

b. Escriba un programa en C++ que lea el archivo de datos creado en el ejercicio 7a y calcule y despliegue una lista de nómina. La salida deberá enlistar el número de seguro social, nombre, pago bruto para cada individuo y donde el pago bruto se calcula se hará como *tarifa por hora x horas trabajadas*.

8. a. Cree un archivo de texto que contenga los siguientes números de automóviles, número de millas recorridas y número de galones de gasolina utilizados en cada automóvil (no incluya los encabezados):

Núm. de automóvil	Millas recorridas	Galones utilizados
54	250	19
62	525	38
71	123	6
85	1 322	86
97	235	14

b. Escriba un programa en C++ que lea los datos en el archivo creado en el ejercicio 8a y despliegue el número de automóvil, las millas recorridas, los galones utilizados y las millas por galón para cada automóvil. La salida deberá contener el total de millas recorridas, el total de galones utilizados y el promedio de millas por galón para todos los automóviles. Estos totales deberán desplegarse al final del reporte de salida.

9. a. Cree un archivo de texto con los siguientes datos (sin los encabezados):

Número de parte	Cantidad inicial	Cantidad vendida	Cantidad mínima
QA310	95	47	50
CM145	320	162	200
MS514	34	20	25
EN212	163	150	160

b. Escriba un programa en C++ para crear un reporte de inventario basado en los datos en el archivo creado en el ejercicio 9a. El despliegue deberá consistir en el número de parte, balance actual y la cantidad que es necesaria para mantener el inventario en el nivel mínimo.

10. a. Cree un archivo de texto que contenga los siguientes datos (sin encabezados):

Nombre	Tarifa	Horas
Callaway,G.	6.00	40
Hanson,P.	5.00	48
Lasard,D.	6.50	35
Stillman,W.	8.00	50

b. Escriba un programa en C++ que use la información contenida en el archivo creado en el ejercicio 10a para producir el siguiente reporte de pagos para cada empleado:

```
Nombre  Tarifa tarifa  Horas  Pago bruto  Pago de tiempo extra  Pago regular
```

El pago regular se calculará como cualesquiera horas trabajadas hasta 40 horas inclusive, multiplicadas por la tarifa de pago. El pago de tiempo extra se calculará como cualesquiera horas trabajadas que rebasen las 40 horas por una tarifa de pa-

go de 1.5 multiplicada por la tarifa regular, y el pago bruto es la suma del pago regular y el pago de tiempo extra. Al final del reporte, el programa deberá desplegar los totales de las columnas de pago regular, de tiempo extra y bruto.

11. a. Almacene los siguientes datos en un archivo:

5 96 87 78 93 21 4 92 82 85 87 6 72 69 85 75 81 73

b. Escriba un programa en C++ para calcular y desplegar el promedio de cada grupo de números en el archivo creado en el ejercicio 11a. Los datos están ordenados en el archivo de modo que cada grupo de números esté precedido por el número de elementos de datos en el grupo. Por tanto, el primer número en el archivo, 5, indica que los siguientes cinco números deberán agruparse juntos. El número 4 indica que los siguientes cuatro números son un grupo, y el 6 indica que los últimos seis números son un grupo. (*Sugerencia:* Use un ciclo anidado. El ciclo exterior deberá terminar cuando se haya encontrado el final del archivo.)

8.3 EXCEPCIONES Y COMPROBACIÓN DE ARCHIVOS[1]

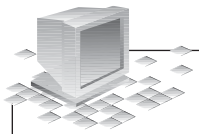
La detección de errores y su procesamiento con el manejo de excepciones se usa en forma extensa dentro de los programas en C++ que usan uno o más archivos. Por ejemplo, si un usuario elimina o cambia el nombre a un archivo usando un comando del sistema operativo, esta acción causará que un programa en C++ falle cuando una llamada a la función `open()` intente abrir el archivo bajo su nombre original.

Se recordará de la sección 7.1 que el código para el manejo de excepciones general luce como éste:

```
try
{
  // una o más instrucciones,
  // al menos una de las cuales deberá
  // lanzar una excepción
}
catch(tipo-de-datos-de-la-excepción nombre_parámetro)
{
  // una o más instrucciones
}
```

En este código, se ejecutan las instrucciones del `bloque try`. Si no ocurren errores, las instrucciones del bloque `catch` se omiten y el procesamiento continúa con la instrucción que sigue al bloque `catch`. Sin embargo, si cualquier instrucción dentro del bloque `try` lanza una excepción, se ejecuta el bloque `catch` cuyo tipo de datos de la excepción corresponda con ésta. Si no se definió un bloque `catch` para un bloque `try`, ocurre un error de compilador. Si no existe un bloque `catch` que atrape un tipo de datos lanzado, ocurre una caída del programa si se lanza la excepción. La mayor parte de las veces, el bloque `catch` despliega un mensaje de error y termina el procesamiento con una llamada a la función `exit()`. El programa 8.7 ilustra las instrucciones requeridas para abrir un archivo en modo de lectura que incluyan manejo de excepciones.

[1]Esta sección puede omitirse en la primera lectura sin perder la continuidad temática.



Programa 8.7

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string nombre_archivo = "precios.dat"; // pone al frente el nombre del archivo
  string descrip;
  double precio;

  ifstream archivo_entr;

  try  // este bloque trata de abrir el archivo, leerlo y desplegar
         los datos del archivo
  {
    archivo_entr.open(nombre_archivo.c_str());

    if (archivo_entr.fail()) throw nombre_archivo; // ésta es la excepción
    que se está comprobando

    // lee y despliega el contenido del archivo
    archivo_entr >> descrip >> precio;
    while (archivo_entr.good()) // comprueba el siguiente carácter
    {
      cout << descrip << ' ' << precio << endl;
      archivo_entr >> descrip >> precio;
    }
    archivo_entr.close();

    return 0;
  }
  catch (string e)
  {
    cout << "\nEl archivo " << e << " no se abrió con éxito"
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
     exit(1);
  }
}
```

Punto de información

Comprobar que un archivo se abrió con éxito

Usando manejo de excepciones, el método más común para comprobar que el sistema operativo localizó el archivo designado es el codificado en el programa 8.7, cuyos puntos clave de codificación se repiten aquí por comodidad:

```
try // este bloque trata de abrir el archivo, leerlo y desplegar los
       datos del archivo
{
  // abre el archivo, lanzando una excepción si la apertura falla
  // ejecuta todo el procesamiento del archivo requerido
  // cierra el archivo
}
catch (string e)
{
  cout << "\nEl archivo " << e << " no se abrió con éxito"
       << "\n Por favor compruebe que el archivo existe en realidad." << endl;
  exit(1);
}
```

El mensaje de excepción producido por el programa 8.7 cuando no se encontró el archivo `precios.dat` es el siguiente:

```
El archivo precios.dat no se abrió con éxito
 Por favor compruebe que el archivo existe en realidad.
```

Aunque el código de manejo de excepciones en el programa 8.7 puede usarse para comprobar una apertura de archivo exitosa para entrada y salida, por lo general se requiere una comprobación más rigurosa para los archivos de salida porque, en la salida, casi está garantizado que el archivo será encontrado. Si existe, el archivo se encontrará; si no existe, el sistema operativo lo creará (a menos que se haya especificado el modo anexar y el archivo exista, o el sistema operativo no pueda encontrar la carpeta indicada). Sin embargo, saber que el archivo se ha encontrado y abierto es insuficiente para propósitos de salida cuando un archivo de salida existente no debe sobrescribirse. En estos casos, el archivo puede abrirse para entrada y, es encontrado, puede hacerse una comprobación posterior para asegurar que el usuario proporciona de manera explícita la aprobación para sobrescribirlo. La forma en que se logra esto se ilustra en código resaltado dentro del programa 8.8.

Programa 8.8

```
#include <iostream>
#include <fstream>
#include <cstdlib> // necesario para exit()
#include <string>
#include <iomanip> // necesario para formatear
using namespace std;

int main()
{
  char respuesta;
  string nombre_archivo = "precios.dat"; // pone al frente el nombre del archivo
  ifstream archivo_entr;
  ofstream archivo_sal;

  try // abre un flujo de entrada básico simplemente para comprobar si
        el archivo existe
  {
    archivo_entr.open(nombre_archivo.c_str());
    if (archivo_entr.fail()) throw 1; // esto significa que el archivo
    no existe
      // sólo llega aquí si se encontró el archivo;
      // de lo contrario el bloque catch toma el control
    cout << "Un archivo con el nombre " << nombre_archivo << " existe
    actualmente.\n"
         << "Desea sobrescribirlo con los datos nuevos (si o no): ";
    cin >> respuesta;
    if (tolower(respuesta) == 'no')
    {
      archivo_entr.close();
      cout << "El archivo existente no se ha sobrescrito." << endl;
      exit(1);
    }
   }
   catch(int e) {};   // un bloque para no hacer nada que permita
                      // que continúe el procesamiento
   try
   {
     // abre el archivo en modo de escritura y continúa con la escritura
        del archivo
     archivo_sal.open(nombre_archivo.c_str());
     if (archivo_sal.fail()) throw nombre_archivo;
     // establece los formatos del flujo de archivo de salida
```

(Continúa)

(Continuación)

```
      archivo_sal << setiosflags(ios::fixed)
                  << setiosflags(ios::showpoint)
                  << setprecision(2);
      // escribe los datos en el archivo
      archivo_sal << "Alfombras " << 39.95 << endl
                  << "Bombillas " << 3.22 << endl
                  << "Fusibles " << 1.08 << endl;
      archivo_sal.close();
      cout << "El archivo " << nombre_archivo
           << " se ha escrito con éxito." << endl;


      return 0;
    }
    catch(string e)
    {
      cout << "El archivo " << nombre_archivo
           << " no se abrió para salida y no se ha escrito."
           << endl;
    }
}
```

En el programa 8.8, los `bloques try` están separados. Debido a que un bloque catch está afiliado con el `bloque try` previo más cercano, no hay ambigüedad respecto a bloques `try` y `catch` sin par.

Apertura de múltiples archivos

Como un ejemplo de la aplicación del manejo de excepciones en la apertura de dos archivos al mismo tiempo, supóngase que se desea leer los datos de un archivo basado en caracteres llamado `info.txt`, un carácter a la vez, y escribir estos datos en un archivo llamado `info.bak`. En esencia, esta aplicación es un programa de copia de archivos que lee los datos de un archivo carácter por carácter y escribe los datos en un segundo archivo. Como ilustración, suponga que los caracteres almacenados en el archivo de entrada son como se muestra en la figura 8.3.

```
Ahora es el momento en que todas las buenas personas
  acudan en ayuda de su partido.
Por favor llame al (555) 888-6666 para
  mayor informacion.
```

Figura 8.3 *Los datos almacenados en el archivo* `info.txt`.

La figura 8.4 ilustra la estructura de los flujos necesarios para producir la copia del archivo. En esta figura, un objeto de flujo de entrada referenciado por la variable `archi-`

vo_entr leerá datos del archivo `info.txt`, y un objeto de flujo de salida referenciado por la variable `archivo_sal` escribirá datos en el archivo `info.bak`.

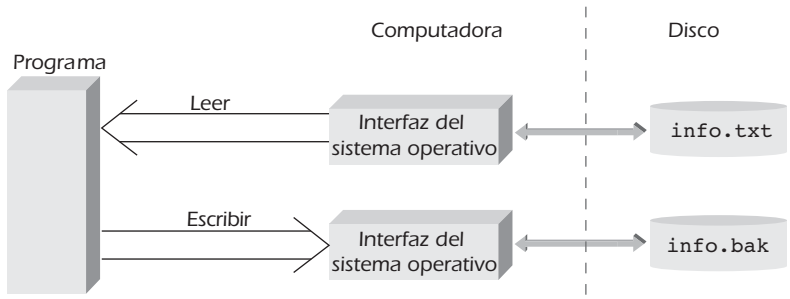


Figura 8.4 *Estructura del flujo de la copia del archivo.*

Ahora considere el programa 8.9, el cual crea el archivo `info.bak` como un duplicado exacto del archivo `info.txt` utilizando el procedimiento ilustrado en la figura 8.4.

Programa 8.9

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string archivoUno = "info.txt"; // pone al frente el nombre del archivo
  string archivoDos = "info.bak";
  char ch;
  ifstream archivo_entr;
  ofstream archivo_sal;

  try   // este bloque trata de abrir el archivo de entrada
  {
    // abre un flujo de entrada básico
    archivo_entr.open(archivoUno.c_str());
      if (archivo_entr.fail()) throw archivoUno;
  } // fin del bloque try exterior
  catch (string entrar) // catch para el bloque try exterior
```

(Continúa)

(Continuación)

```
  {
    cout << "El archivo de entrada " << entrar
         << " no se abrió con éxito." << endl
         << " No se hizo el respaldo." << endl;
    exit(1);
  }

  try // este bloque trata de abrir el archivo de salida y
  {    // ejecuta todo el procesamiento del archivo

    archivo_sal.open(archivoDos.c_str());
    if (archivo_sal.fail()) throw archivoDos;
    while (ch = archivo_entr.get())!= EOF)
       archivo_sal.put(ch);

    archivo_entr.close();
    archivo_sal.close();
   }
   catch (string salir)   // catch para el bloque try interior
   {
     cout << "El archivo de respaldo " << salir
          << " no se abrió con éxito." << endl;
     exit(1);
   }

  cout << "Un respaldo exitoso de " << archivoUno
       << " llamado " << archivoDos << " se realizó correctamente." << endl;

  return 0;
}
```

Por simplicidad, el programa 8.9 intenta abrir los archivos de entrada y salida dentro de bloques `try` separados y no anidados. De manera más general, el segundo archivo será abierto en un bloque `try` interior anidado de modo que el intento de abrir este segundo archivo no se haría si la apertura del primer archivo lanzara una excepción. (El Punto de información sobre anidación de bloques `try` explica cómo se logra esto.)

Punto de información

Anidación de bloques `try`

Cuando está implicado más de un flujo de archivo, abrir cada uno de ellos en su propio bloque `try` permite el aislamiento e identificación exactos de cuál archivo causó una excepción, si ocurre una. Los bloques `try` pueden anidarse. Por ejemplo, considere el programa 8.9, el cual se volvió a escribir aquí usando bloques `try` anidados. Hay que observar que en este caso el bloque `catch` para el bloque `try` interior debe anidarse en el mismo alcance de bloque que su bloque

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // necesario para exit()
#include <string>
using namespace std;

int main()
{
  string archivoUno = "info.txt"; // pone al frente el nombre
                      del archivo
  string archivoDos = "info.bak";

  char ch;
  ifstream archivo_entr;
  ifstream archivo_sal;

  try // este bloque trata de abrir el archivo de entrada
  {
    // abre un flujo de entrada básico
    archivo_entr.open(archivoUno.c_str());
    if (archivo_entr.fail()) throw archivoUno;
    try  // este bloque trata de abrir el archivo de salida y
    {    // ejecuta todo el procesamiento del archivo

         // abre un flujo de salida básico
      archivo_sal.open(archivoDos.c_str());
      if (archivo_sal.fail()) throw archivoDos;
      while ((ch = archivo_entr.get()) != EOF)
        archivo_sal.put(ch);

      archivo_entr.close();
      archivo_sal.close();
    }  // fin del bloque try interior
    catch (string salir) // catch para el bloque try interior
```

(Continúa)

(Continuación)

```
    {
      cout << "El archivo de respaldo " << salir
           << " no se abrió con éxito." << endl
      exit(1);
    }
  }  // fin del bloque try exterior
  catch (string entrar) // catch para el bloque try exterior
  {
    cout << "El archivo de entrada " << entrar
         << " no se abrió con éxito." << endl;
         << "No se hizo ningun respaldo." << endl;
    exit(1);
  }

  cout << "Un respaldo exitoso de " << archivoUno
       << " llamado " << archivoDos << " se realizó correctamente."
          << endl;

  return 0;
}
```

El punto importante que hay que señalar en este programa es el anidamiento de los bloques `try`. Si los dos bloques `try` no estuvieran anidados y la declaración del flujo de entrada, `ifstream archivo_entr;` , se colocara en el primer bloque, no podría utilizarse en el segundo bloque `try` sin producir un error de compilador. La razón para esto es que todas las variables declaradas en un bloque de código, el cual es definido por un par de llaves de apertura y de cierre, son locales para el bloque en el que fueron declaradas.

Al revisar el programa 8.9, ponga particular atención a la instrucción:

```
while((ch = archivo_entr.get())!= EOF)
```

Esta instrucción lee en forma continua un valor del flujo de entrada hasta que se detecta el valor EOF. En tanto el valor devuelto no sea igual al valor EOF, el valor es escrito en el flujo del objeto de salida. Los paréntesis que rodean a la expresión `(ch = archivo_entr.get())` son necesarios para asegurar que el valor es leído primero y asignado a la variable `ch` antes que el valor recuperado se compare con el valor EOF. En su ausencia, la expresión completa sería `ch = archivo_entr.get()!= EOF`. Debido a la precedencia de las operaciones, la expresión relacional `archivo_entr.get()!= EOF` se ejecutaría primero. Debido a que ésta es una expresión relacional, su resultado es un valor booleano verdadero o falso basado en los datos recuperados por el método `get()`. Asignar este resultado booleano a la variable de carácter `ch` es una conversión inválida a través de un operador de asignación.

Ejercicios 8.3

1. Enliste dos condiciones que causen una condición de falla cuando un archivo es abierto para entrada.
2. Enliste dos condiciones que causen una condición de falla cuando un archivo es abierto para salida.
3. Si un archivo que existe es abierto para salida en modo de escritura, ¿qué les sucederá a los datos que están en el archivo?
4. Modifique el programa 8.7 para usar un identificador de su elección, en lugar de la letra `e`, para el nombre del parámetro de excepción del bloque `catch`.
5. Introduzca y ejecute el programa 8.8.
6. Determine por qué los dos bloques `try` en el programa 8.8, los cuales no están anidados, no causan problemas en la compilación o la ejecución. (*Sugerencia:* Coloque la declaración para el nombre del archivo dentro del primer bloque `try` y compile el programa.)
7. **a.** Si los bloques `try` anidados en el Punto de información sobre bloques `try` anidados se separan en bloques no anidados, el programa no se compilará. Determine por qué sucede esto.
   b. ¿Qué cambios adicionales tendría que hacer al programa en el ejercicio 7a que permitirían que se escribiera con bloques no anidados? (*Sugerencia:* Vea el ejercicio 6.)
8. Introduzca los datos para el archivo `info.txt` en la figura 8.3 u obténgalos en el sitio web de este texto (véase el Prefacio para la URL). Luego introduzca y ejecute el programa 8.9 y verifique que haya sido escrito el archivo de respaldo.
9. Modifique el programa 8.9 para usar un método `getline()` en lugar del método `get()` que se encuentra actualmente en el programa.

8.4 ARCHIVOS DE ACCESO ALEATORIO

El término **acceso a archivos** se refiere al proceso de recuperar datos de un archivo. Existen dos tipos de acceso a los archivos: acceso secuencial y acceso aleatorio. Para entender los tipos de acceso a los archivos, es necesario entender algunos conceptos relacionados con la forma en que los datos están organizados dentro de un archivo.

El término **organización de archivos** se refiere a la forma en que los datos están almacenados en un archivo. Los archivos que se han usado, y continuarán usándose, tienen una **organización secuencial.** Esto significa que los caracteres dentro del archivo están almacenados de una manera secuencial.

Además de estar organizados en forma secuencial, se ha leído cada archivo abierto de una manera secuencial. Es decir, se ha tenido acceso a cada carácter en secuencia. Esto se conoce como **acceso secuencial.** Sin embargo, aunque los caracteres en el archivo estén almacenados en forma secuencial, esto no nos obliga a tener acceso secuencial a ellos. De hecho, podemos saltar caracteres y leer un archivo organizado en forma secuencial de una manera no secuencial.

En el **acceso aleatorio,** cualquier carácter en el archivo abierto puede leerse en forma directa sin tener que leer primero en forma secuencial todos los caracteres almacenados antes que él. Para proporcionar acceso aleatorio a los archivos, cada objeto `ifstream` crea en forma automática un marcador de posición de archivo. Este marcador es un número entero largo que representa un desplazamiento desde el principio de cada archivo y le sigue la pista al lugar desde donde se va a leer o a escribir el siguiente carácter. Las funciones usadas para tener acceso y cambiar el marcador de posición del archivo se muestran en la tabla 8.4. Los sufijos `g` y `p` en estos nombres de función denotan `get` y `put`, respectivamente, donde `get` se refiere a un archivo de entrada (obtener desde) y `put` se refiere a un archivo de salida (poner en).

Tabla 8.4 Funciones de marcadores de posición del archivo

Nombre	Descripción
`seekg(offset, mode)`	Para archivos de entrada, se mueve a la posición de desplazamiento indicada por el modo.
`seekp(offset, mode)`	Para archivos de salida, se mueve a la posición de desplazamiento indicada por el modo.
`tellg(void)`	Para archivos de entrada, devuelve el valor actual del marcador de posición del archivo.
`tellp(void)`	Para archivos de salida, devuelve el valor actual del marcador de posición del archivo.

Las funciones `seek()` permiten al programador moverse a cualquier posición en el archivo. Para entender este método, debe entenderse cómo están referenciados los datos en el archivo usando el marcador de posición del archivo.

Cada carácter en un archivo de datos se localiza por su posición en el archivo. El primer carácter en el archivo se localiza en la posición 0, el siguiente en la posición 1, etc. Se hace referencia a la posición de un carácter como su desplazamiento desde el inicio del archivo. Por tanto, el primer carácter tiene un desplazamiento de 0, el segundo carácter tiene un desplazamiento de 1, etc., para cada carácter en el archivo.

Las funciones `seek()` requieren dos argumentos: el primero es el desplazamiento, como un número entero largo, en el archivo; el segundo es desde dónde se va a calcular el desplazamiento, lo que es determinado por el modo. Las tres alternativas posibles para el modo son `ios::beg`, `ios::cur` e `ios::end`, los cuales denotan el principio, la posición actual y el final del archivo, respectivamente. Por tanto, un modo de `ios::beg` significa que el desplazamiento es el desplazamiento verdadero desde el inicio del archivo. Un modo de `ios::cur` significa que el desplazamiento es relativo a la posición actual en el archivo, y un modo `ios::end` significa que el desplazamiento es relativo al final del archivo. Un desplazamiento positivo significa avanzar en el archivo y un desplazamiento negativo significa retroceder. A continuación se muestran ejemplos de llamadas a la función `seek()`. En estos ejemplos, suponga que se ha abierto `archivo_entr` como un archivo de entrada y `archivo_sal` como un archivo de salida. En estos ejemplos, el desplazamiento transmitido a `seekg()` y `seekp()` debe ser un número entero largo.

```
iarchivo_entr.seekg(4L, ios::beg); // va al quinto carácter en el archivo de entrada
archivo_sal.seekp(4L, ios::beg);   // va al quinto carácter en el archivo de salida
archivo_entr.seekg(4L, ios::cur);  // avanza cinco caracteres en el archivo de entrada
```

```
archivo_sal.seekp(4L, ios::cur);   // avanza cinco caracteres en el archivo de salida
archivo_entr.seekg(-4L, ios::cur);  // retrocede cinco caracteres en el archivo de entrada
archivo_sal.seekp(-4L, ios::cur);  // retrocede cinco caracteres en el archivo de salida
archivo_entr.seekg(0L, ios::beg);  // va al inicio del archivo de entrada
archivo_sal.seekp(0L, ios::beg);   // va al inicio del archivo de salida
archivo_entr.seekg(0L, ios::end);  // va al final del archivo de entrada
archivo_sal.seekp(0L, ios::end);   // va al final del archivo de salida
archivo_entr.seekg(-10L, ios::end);  //  va a 10 caracteres antes del final del archivo de
                                        entrada
archivo_sal.seekp(-10L, ios::end);   //  va a 10 caracteres antes del final del archivo de
                                        salida
```

A diferencia de las funciones `seek()` que mueven el marcador de posición del archivo, las funciones `tell()` devuelven el valor de desplazamiento del marcador de posición del archivo. Por ejemplo, si se han leído diez caracteres desde un archivo de entrada llamado `archivo_entr`, la llamada a la función devuelve el número entero largo 10:

```
archivo_entr.tellg();
```

Esto significa que el siguiente carácter que se va a leer está desplazado diez posiciones de byte del inicio del archivo y es el undécimo carácter en el archivo.

El programa 8.10 ilustra el uso de `seekg()` y `tellg()` para leer un archivo en orden inverso, del último carácter al primero. Conforme se lee cada carácter, también se despliega.

Programa 8.10

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;

int main()
{
  string nombre_archivo = "prueba.dat";
  char ch;
  long offset, last;

  ifstream archivo_entr(nombre_archivo.c_str());

  if (archivo_entr.fail())   // comprueba la apertura con exito
  {
    cout << "\nEl archivo no se abrió con éxito."
         << "\n Por favor compruebe que el archivo existe en realidad."
         << endl;
    exit(1);
  }
```

(Continúa)

(Continuación)

```
  archivo_entr.seekg(0L, ios::end);  // se mueve al final del archivo
  last = archivo_entr.tellg(); // guarda el desplazamiento del último carácter

  for(offset = 1L; offset <= last; offset++)
  {
    archivo_entr.seekg(-offset, ios::end);
    ch = archivo_entr.get();
    cout << ch << " : ";
  }

  archivo_entr.close();

  cout << endl;

  return 0;
}
```

Suponga que el archivo `prueba.dat` contiene los siguientes datos:

```
El grado fue 92.5
```

La salida del programa 8.10 es la siguiente:

```
5 : . : 2 : 9 :   : e : u : f :   : o : d : a : r : g :   : I : E :
```

El programa 8.10 va inicialmente al último carácter en el archivo. El desplazamiento de este carácter, el carácter EOF, es guardado en la variable `last`. Dado que `tellg()` devuelve un número entero largo, `last` ha sido declarado como un número entero largo.

Empezando desde el final del archivo, se usa `seekg()` para ubicar el siguiente carácter que se va a leer, referenciado desde el final del archivo. Conforme se lee cada carácter, el carácter es desplegado y el desplazamiento ajustado para tener acceso al siguiente carácter. El primer desplazamiento usado es −1, el cual representa al carácter que precede inmediatamente al marcador EOF.

Ejercicios 8.4

1. **a.** Cree un archivo llamado `prueba.dat` que contenga los datos que hay en el archivo `prueba.dat` usado en el programa 8.10. Puede hacer esto usando un editor de texto.
   b. Introduzca y ejecute el programa 8.10 en su computadora.
2. Vuelva a escribir el programa 8.10 de modo que el origen para la función `seekg()` usada en el ciclo `for` sea el inicio del archivo en lugar del final.
3. Modifique el programa 8.10 para desplegar un mensaje de error si `seekg()` intenta referenciar una posición más allá del final del archivo.

4. Escriba un programa que lea y despliegue cada segundo carácter en un archivo llamado `prueba.dat`.

5. Usando las funciones `seek()` y `tell()`, escriba una función llamada `carac_archivo()` que devuelva el número total de caracteres en un archivo.

6. **a.** Escriba una función llamada `leerBytes()` que lea y despliegue *n* caracteres empezando desde cualquier posición en un archivo. La función deberá aceptar tres argumentos: un nombre de objeto de archivo, el desplazamiento del primer carácter que se va a leer y el número de caracteres que se leerán. (*Nota*: El prototipo para `leerBytes()` deberá ser `void leerBytes(fstream&, long, int)`.)
   b. Modifique la función `leerBytes()` escrita en el ejercicio 6a para almacenar los caracteres leídos en una cadena o arreglo. La función deberá aceptar la dirección del área de almacenamiento como un cuarto argumento.

8.5 FLUJOS DE ARCHIVO COMO ARGUMENTOS DE FUNCIONES

Un objeto de flujo de archivo puede utilizarse como un argumento de función. El único requisito es que el parámetro formal de la función sea una referencia (véase la sección 6.3) al flujo apropiado, ya sea `ifstream&` u `ofstream&`. Por ejemplo, en el programa 8.11, un objeto `ofstream` llamado `archivo_sal` se abre en `main()` y este objeto de flujo es transmitido a la función in`Out()`. El prototipo de la función y la línea de encabezado para `inOut()` declaran el parámetro formal como una referencia a un tipo de objeto `ofstream`. La función in`Out()` se usa entonces para escribir en el archivo cinco líneas de texto introducido por el usuario.

Programa 8.11

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;


int main()
{
  string nombre_archivo = "lista.dat"; // aquí está el archivo con el que
                          estamos trabajando

  void inOut(ofstream&);    // prototipo de la función

  ofstream archivo_sal;
```

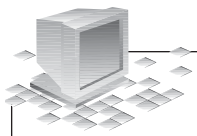
(Continúa)

(Continuación)

```
  archivo_sal.open(nombre_archivo.c_str());
  if (archivo_sal.fail())   // comprueba una apertura exitosa
  {
    cout << "\nEl archivo de salida " << nombre_archivo << " no se abrio
            con exito"

         << endl;
    exit(1);
  }

  inOut(archivo_sal);   // llama a la función

  return 0;
}

void inOut(ofstream& salArchivo)
{

  const int NUMLINEAS = 5; // número de líneas de texto
  string linea;
  int cuenta;

  cout << "Por favor introduzca cinco líneas de texto:" << endl;
  for (cuenta = 0; cuenta < NUMLINEAS; cuenta++)
  {
    getline(cin,linea);
    salArchivo << linea << endl;
  }

  cout << "\nEl archivo se ha escrito con éxito." << endl;
  return;
}
```

Dentro de `main()`, el archivo es un objeto `ostream` llamado `archivo_sal`. Este objeto es transmitido a la función `inOut()` y es aceptado como el parámetro formal llamado `salArchivo`, el cual es declarado como una referencia a un tipo de objeto `ostream`. La función `inOut()` usa entonces su parámetro de referencia `archivo_sal` como un nombre de flujo de archivo de salida de la misma manera que `main()` usaría el objeto de flujo `salArchivo`. El programa 8.11 usa el método `getline()` introducido en la sección 8.2 (véase la tabla 8.3).

En el programa 8.12 se ha expandido el programa 8.11 agregando una función `getOpen()` para ejecutar la apertura. `getOpen()`, como `inOut()`, acepta un argumento de referencia para un objeto `ofstream`. Después que la función `getOpen()` completa la ejecución, esta referencia es transmitida a `inOut()`, como en el programa 8.11. Aunque podría estar tentado a escribir `getOpen()` para que devuelva una referencia a un `ofstream`, esto no funcionará porque produce un intento de asignar una referencia devuelta a una ya existente.



Programa 8.12

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
  int getOpen(ofstream&);  // transmite una referencia a un fstream
  void inOut(ofstream&);   // transmite una referencia a un fstream

  ofstream archivo_sal;   // el nombre de archivo es un objeto fstream

  getOpen(archivo_sal);   // abre el archivo
  inOut(archivo_sal);     // escribe en el

  return 0;
}

int getOpen(ofstream& salArchivo)
{
  string nombre;

  cout << "\nIntroduzca un nombre de archivo: ";
  getline(cin,nombre);

  salArchivo.open(nombre.c_str());   // abre el archivo

  if (salArchivo.fail())   // comprueba una apertura exitosa
  {
    cout << "No se pudo abrir el archivo" << endl;
    exit(1);
  }
  else
    return 1;
}
```

(Continúa)

(Continuación)

```
void inOut(ofstream& salArchivo)
{
  const int NUMLINEAS = 5; // número de líneas
  int cuenta;
  string linea;

  cout << "Por favor introduzca cinco líneas de texto:" << endl;
  for (cuenta = 0; cuenta < NUMLINEAS; cuenta++)
  {
    getline(cin,linea);
    salArchivo << linea << endl;
  }
  cout << "\nEl archivo se ha escrito con éxito." << endl;
  return;
}
```

El programa 8.12 es una versión modificada del programa 8.11 que permite al usuario introducir un nombre de archivo desde el dispositivo de entrada estándar y luego abre la conexión `ofstream` al archivo externo. Si se introduce el nombre de un archivo de datos existente, el archivo será destruido cuando se abra para salida. Un truco útil que puede emplearse para prevenir este tipo de percance es abrir el archivo introducido usando un flujo de archivo de entrada. Si el archivo existe, el método `fail()` indicará una apertura exitosa (es decir, la apertura no fallará), lo cual indica que el archivo está disponible para entrada. Esto puede usarse para alertar al usuario que existe en el sistema un archivo con el nombre introducido y para solicitar confirmación de que los datos en el archivo pueden destruirse y el archivo abrirse para salida. Antes que el archivo sea reabierto para salida, deberá cerrarse el flujo de archivo de entrada. La puesta en práctica de este algoritmo se deja como ejercicio.

Ejercicios 8.5

1. Una función llamada `p_archivo()` recibirá un nombre de archivo como una referencia a un objeto `ifstream`. ¿Qué declaraciones se requieren para transmitir un nombre de archivo a `p_archivo()`?

2. Escriba una función, llamada `revisar_archivo()`, que compruebe si existe un archivo. La función aceptará un objeto `ifstream` como un parámetro de referencia formal. Si el archivo existe, deberá devolver un valor de 1; de lo contrario, la función deberá devolver un valor de cero.

3. Suponga que se ha creado un archivo de datos consistente en un grupo de líneas individuales. Escriba una función llamada `impr_linea()` que leerá y desplegará cualquier línea deseada del archivo. Por ejemplo, la llamada a la función `impr_linea(fstream& nombre_archivo,5)`; deberá desplegar la quinta línea del flujo de objeto transmitido.

4. Vuelva a escribir la función `getOpen()` usada en el programa 8.12 para incorporar los procedimientos de comprobación de archivo descritos en esta sección. De manera específica, si existe el nombre de archivo introducido, deberá desplegarse un mensaje apropiado. Al usuario deberá presentársele la opción de introducir un

nombre de archivo nuevo o permitir que el programa sobrescriba el archivo existente. Use la función escrita para el ejercicio 2 en su programa.

8.6 ERRORES COMUNES DE PROGRAMACIÓN

Los errores comunes de programación con respecto a los archivos son:

1. Usar un nombre externo de archivo en lugar del nombre de objeto de flujo de archivo interno cuando se tiene acceso al archivo. El único método de flujo que usa el nombre externo del archivo de datos es la función `open()`. Como siempre, todos los métodos de flujo presentados en este capítulo deben ser precedidos por un nombre de objeto de flujo y el operador punto.
2. Abrir un archivo para salida sin comprobar primero que ya existe un archivo con el nombre dado. No comprobar un nombre de archivo preexistente asegura que el archivo será sobrescrito.
3. No entender que el fin de archivo sólo se detecta después que se ha leído o transmitido el centinela EOF.
4. Intentar detectar el final de un archivo usando variables de carácter para el marcador EOF. Cualquier variable usada para aceptar el EOF debe ser declarada como una variable en entero. Por ejemplo, si se declara `ch` como una variable de carácter, la siguiente expresión produce un ciclo infinito.[2]

   ```
   while ( (ch = in.archivo.peek()) != EOF )
   ```

   Esto ocurre debido a que una variable de carácter nunca puede tomar un código EOF. El EOF es un valor entero (por lo general −1) que no tiene representación de carácter. Esto asegura que el código EOF nunca pueda confundirse con ningún carácter legítimo encontrado como dato normal en el archivo. Para terminar el ciclo creado por la expresión anterior, la variable `ch` debe declararse como una variable en número entero.
5. Usar un argumento en número entero con las funciones `seekg()` y `seekp()`. Este desplazamiento debe ser una constante o variable en número entero largo. Cualquier otro valor transmitido a estas funciones puede producir un efecto imprevisible.

8.7 RESUMEN DEL CAPÍTULO

1. Un archivo de datos es cualquier colección de datos almacenados juntos en un medio de almacenamiento externo bajo un nombre común.
2. Un archivo de datos se conecta a un flujo de archivos usando el método `open()` de `fstream`. Esta función conecta un nombre externo de archivo con un nombre de objeto interno. Después que se ha abierto el archivo, todos los accesos subsiguientes al archivo requieren el nombre del objeto interno.

[2]Esto no ocurrirá en el sistema UNIX, ya que los caracteres se almacenan como enteros con signo.

3. Un archivo puede abrirse en modo de entrada o de salida. Un flujo de archivo de salida abierto crea un archivo de datos nuevo o elimina los datos en un archivo abierto existente. Un flujo de archivo de entrada abierto hace que los datos en un archivo existente estén disponibles para entrada. Se produce una condición de error si el archivo no existe y puede detectarse usando el método `fail()`.
4. Todos los flujos de archivo deben declararse como objetos de las clases `ifstream` u `ofstream`. Esto significa que debe incluirse una declaración similar a cualquiera de las siguientes con las que se abre el archivo:

   ```
   ifstream archivo_entr;
   ofstream archivo_sal;
   ```

   Los nombres de objeto de flujo `archivo_entr` y `archivo_sal` pueden reemplazarse con cualquier nombre de objeto seleccionado por el usuario.
5. Además de los archivos abiertos dentro de una función, los objetos de flujo estándar `cin`, `cout` y `cerr` son declarados y abiertos en forma automática cuando se ejecuta un programa. `cin` es el nombre de objeto de un flujo de archivo de entrada usado para la introducción de datos (por lo general desde el teclado), `cout` es el nombre de objeto de un flujo de archivo de salida usado para desplegar datos por omisión (por lo general en la pantalla de la computadora) y `cerr` es el nombre de objeto de un flujo de archivo de salida usado para desplegar mensajes de error del sistema (por lo general en la pantalla de la computadora).
6. Puede tenerse acceso aleatorio a los archivos de datos usando los métodos `seekg()`, `seekp()`, `tellg()` y `tellp()`. Las versiones `g` de estas funciones se usan para alterar y buscar el marcador de posición del archivo para flujos de archivo de entrada, y las versiones `p` hacen lo mismo para los flujos de archivo de salida.
7. La tabla 8.5 muestra los métodos suministrados por la clase `fstream` para la manipulación de archivos.

Tabla 8.5 Métodos `fstream`

Nombre del método	Descripción
`get()`	Extrae el siguiente carácter del flujo de entrada y lo devuelve como un `int`.
`get(chrVar)`	Extrae el siguiente carácter del flujo de entrada y lo asigna a `chrVar`.
`getline(fileObj, string, termChar)`	Extrae la siguiente cadena de caracteres del objeto de flujo de entrada y la asigna a la cadena hasta que se detecte el carácter de terminación especificado. Si se omite, el carácter de terminación por omisión es una línea nueva.
`getline(C-stringVar,int n,'\n')`	Extrae y devuelve caracteres del flujo de entrada hasta que se han leído `n-1` caracteres o se encuentra una línea nueva (termina la entrada con un `'\0'`).
`peek()`	Devuelve el siguiente carácter en el flujo de entrada sin extraerlo del flujo.
`put(chrExp)`	Pone el carácter especificado por `chrExp` en el flujo de salida.
`putback(chrExp)`	Pone de vuelta el carácter especificado por `chrExp` en el flujo de entrada. No altera los datos en el archivo.

Tabla 8.5 Métodos `fstream` (continuación)

Nombre del método	Descripción
`ignore(int n)`	Se salta los siguientes *n* caracteres; si se omite `n`, el valor por omisión es saltarse el siguiente carácter individual.
`eof()`	Devuelve un valor booleano `verdadero` si se ha intentado una lectura después del final del archivo; de lo contrario, devuelve un valor booleano `falso`. El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido.
`good()`	Devuelve un valor booleano `verdadero` mientras el archivo está disponible para uso del programa. Devuelve un valor booleano `falso` si se ha intentado una lectura después del final del archivo. El valor se vuelve falso sólo cuando se lee el primer carácter después del último carácter de archivo válido.
`bad()`	Devuelve un valor booleano `verdadero` si se ha intentado una lectura después del final del archivo; de lo contrario, devuelve un `falso`. El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido.
`fail()`	Devuelve un booleano `verdadero` si el archivo no se ha abierto con éxito; de lo contrario, devuelve un valor booleano `falso`.

8.8 COMPLEMENTO DEL CAPÍTULO: LA BIBLIOTECA DE CLASE `iostream`

Como se ha visto, las clases contenidas dentro de la biblioteca de clase `iostream` tienen acceso a los archivos usando entidades llamadas flujos. Para la mayor parte de los sistemas, los bytes de datos transferidos a un flujo representan caracteres ASCII o números binarios.

El mecanismo para leer un flujo de byte de un archivo o escribir un flujo de byte a un archivo está oculto cuando se usa un lenguaje de nivel superior como C++. No obstante, es útil entender este mecanismo de modo que se puedan colocar los servicios proporcionados por la biblioteca de la clase iostream en su contexto apropiado.

Mecanismo de transferencia de flujo de archivos

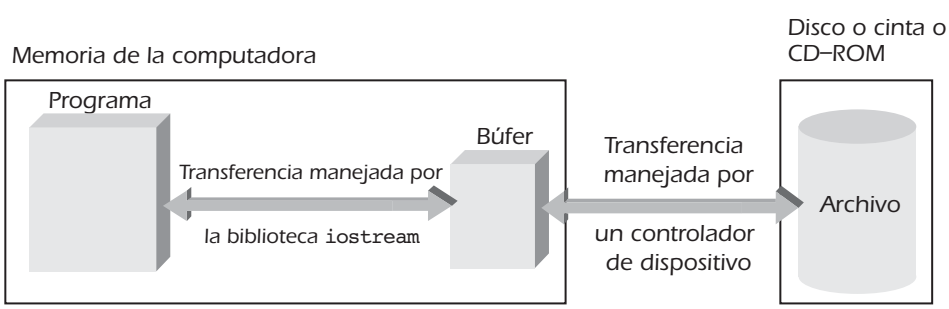


Figura 8.5 *El mecanismo de transferencia de datos.*

El mecanismo para transferir datos entre un programa y un archivo se ilustra en la figura 8.5. Como se puede observar, transferir datos entre un programa y un archivo implica un búfer de archivo intermediario contenido en la memoria de la computadora. A cada archivo abierto se le asigna su propio búfer de archivo, el cual es un área de almacenamiento utilizado por los datos que se están transfiriendo entre el programa y el archivo.

Desde este lado, el programa escribe un conjunto de bytes de datos al búfer del archivo o lee un conjunto de bytes de datos del búfer del archivo usando un objeto de flujo. En el otro lado del búfer, la transferencia de datos entre el dispositivo que almacena el archivo de datos real (por lo general una cinta, disco o CD-ROM) y el búfer del archivo es manejado por programas especiales del sistema operativo a los que se hace referencia como controladores de dispositivo. Los controladores de dispositivo no son programas autónomos sino parte integral del sistema operativo. Un **controlador de dispositivo** es una sección de código del sistema operativo que tiene acceso a un dispositivo de hardware, como una unidad de disco, y maneja la transferencia de datos entre el dispositivo y la memoria de la computadora. Como tal, debe sincronizar en forma correcta la velocidad de los datos transferidos entre la computadora y el dispositivo que envía o recibe los datos. Esto se debe a que la velocidad de transferencia interna de la computadora por lo general es mucho más rápida que cualquier dispositivo conectado a ella.

Por lo general, un controlador de dispositivo de disco sólo transferirá datos entre el disco y el búfer de archivo en tamaños fijos, como 1024 bytes a la vez. Por tanto, el búfer del archivo proporciona un medio conveniente para permitir a un controlador de dispositivo transferir datos en bloques de un tamaño, y el programa puede tener acceso a ellos usando un tamaño diferente (generalmente, como caracteres individuales o como un número fijo de caracteres por línea).

Componentes de la biblioteca de clase `iostream`

La biblioteca de clase `iostream` consiste en dos clases primarias base: la clase `streambuf` y la clase `ios`. La clase `streambuf` proporciona el búfer de archivo, ilustrado en la figura 8.5, y diversas rutinas generales para transferir datos binarios. La clase `ios` contiene un apuntador a los búfer de archivo proporcionados por la clase `streambuf` y diversas rutinas generales para transferir datos de texto. A partir de estas dos clases base, se derivan otras clases diversas y se incluyen en la biblioteca de clase `iostream`.

La figura 8.6 ilustra un diagrama de herencia para la familia de clases `ios` y su relación con las clases `ifstream, ofstream` y `fstream`. El diagrama de herencia para la familia de clases `streambuf` se muestra en la figura 8.7. La convención adoptada para los diagramas de herencia es que las flechas apuntan de una clase derivada a una clase base.

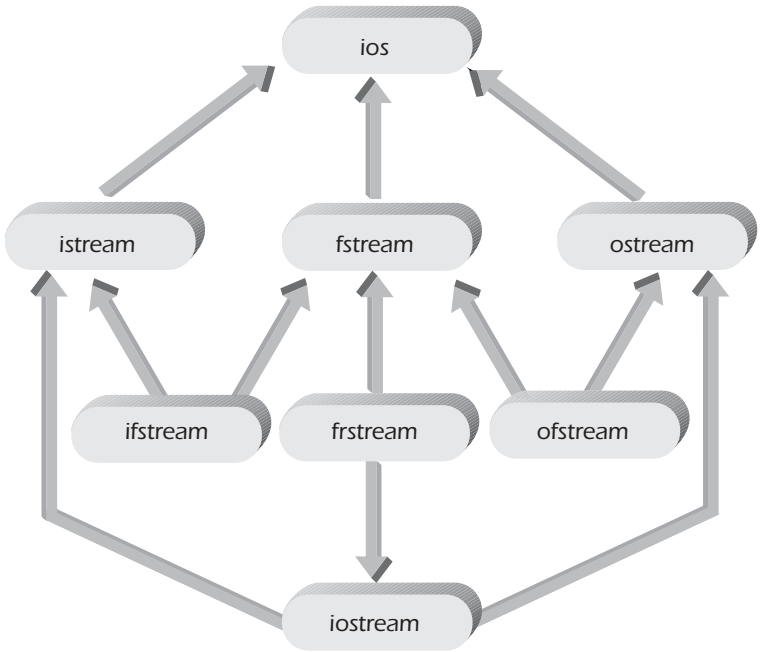


Figura 8.6 *La clase base* `ios` *y sus clases derivadas.*

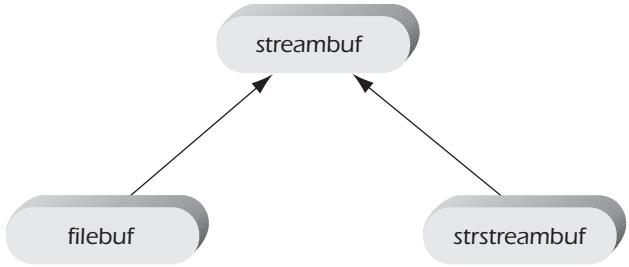


Figure 8.7 *La clase base* `streambuf` *y sus clases derivadas.*

La correspondencia entre las clases ilustradas en las figuras 8.6 y 8.7, incluyendo los archivos de encabezado que definen estas clases, se muestran en la tabla 8.6.

Tabla 8.6 Correspondencia entre las clases ilustradas en las figuras 8.6 y 8.7

Clase ios	Clase streambuf	Archivos de encabezado
istream ostream iostream	streambuf	iostream o fstream
ifstream ofstream fstream	filebuf	fstream

Por tanto, las clases `ifstream, ofstream` y `fstream` que se han usado para el acceso a archivos usan un búfer proporcionado por la clase `filebuf`, definida en el archivo de encabezado `fstream`. Del mismo modo, los objetos `iostream` `cin`, `cout`, `cerr` y `clog` que se han usado a lo largo del texto usan un búfer proporcionado por la clase `streambuf` y definido en el archivo de encabezado `iostream`.

Formateo en memoria

Además de las clases ilustradas en la figura 8.6, también se deriva de la clase `ios` una clase llamada `strstream`. Esta clase usa la clase `strstreambuf` ilustrada en la figura 8.7, requiere el archivo de encabezado `strstream` y proporciona capacidades para escribir y leer cadenas desde y hacia flujos definidos en memoria.

Como flujo de salida, tales flujos se utilizan por lo general para "ensamblar" una cadena de piezas más pequeñas hasta que una línea completa de caracteres está lista para ser escrita, ya sea a `cout` o a un archivo. Anexar un objeto `strstream` a un búfer para este propósito se hace de manera similar a la anexión de un objeto `fstream` a un archivo de salida. Por ejemplo, la instrucción

```
strstream inmem(buf, 72, ios::out);
```

anexa un objeto `strstream` a un búfer existente de 72 bytes en modo de salida. El programa 8.13 ilustra cómo se usa esta instrucción dentro del contexto de un programa completo.

Programa 8.13

```
#include <iostream>
#include <strstream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXCARAC = 81; // uno más que el máximo de caracteres en una línea
  int unidades = 10;
  double precio = 36.85;
  char buf[MAXCARAC];

  strstream inmem(buf, MAXCARAC, ios::out); // abre un flujo en la memoria

  // escribe al bufer a través del flujo
  inmem << "Num. de unidades = "
        << setw(3) << units
        << "  Precio por unidad = $ "
        << setw(6) << setprecision(2) << fixed << precio << '\0';

  cout << '|' << buf << '|';

  cout << endl;

  return 0;
}
```

La salida producida por el programa 8.13 es la siguiente:

```
|Num. de unidades = 10 Precio por unidad = $ 36.85|
```

Esta salida ilustra que el búfer de caracteres se ha llenado en forma correcta por las inserciones al flujo `inmem`. (Hay que observar que el final de la cadena `NULL, '\0'`, el cual es la última inserción en el flujo, se requiere para cerrar en forma correcta la cadena C.) Una vez que se ha llenado el arreglo de caracteres deseado, se escribirá en un archivo como una sola cadena.

De manera similar, un objeto `strstream` puede abrirse en modo de entrada. Dicho flujo se usaría como un área de trabajo de almacenamiento, o búfer, para almacenar una línea completa de texto de un archivo o entrada estándar. Una vez que se ha llenado el búfer, se usará el operador de extracción para "desensamblar" la cadena en sus partes componentes y convertir cada elemento de datos en su tipo de datos designado. Hacer esto permite introducir datos de un archivo línea por línea antes de asignar los elementos de datos individuales a sus variables respectivas.