8. Manejo de ficheros

8.1. Escritura en un fichero de texto

Cuando queramos manipular un fichero, siempre deberemos hacerlo **en tres pasos**:

- Abrir el fichero.
- Leer datos de él o escribir datos en él.
- Cerrar el fichero.

Eso sí, no siempre será posible completar esas operaciones, por lo que además tendremos que comprobar si ha surgido algún error en el proceso. Por ejemplo, puede ocurrir que intentemos abrir un fichero que realmente no exista, o que tratemos de escribir en un dispositivo que sea sólo de lectura.

Vamos a ver un primer fuente, que cree un fichero de texto y escriba algo en él, todavía sin comprobación de errores:

```
// Ejemplo_08_01a.cs
// Escritura en un fichero de texto (sin comprobación de errores)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamWriter

class Ejemplo_08_01a
{
    static void Main()
    {
        StreamWriter fichero;

        fichero = File.CreateText("prueba.txt");
        fichero.WriteLine("Esto es una línea");
        fichero.Write("Esto es otra");
        fichero.WriteLine(" y esto es continuación de la anterior");
        fichero.Close();
    }
}
```

Hay varias cosas que comentar sobre este programa:

- StreamWriter es la clase que representa a un fichero de texto en el que podemos escribir información.
- El fichero de texto lo creamos con el método (estático) CreateText, que pertenece a la clase File.

- Para escribir en el fichero, empleamos Write y WriteLine, al igual que hacemos en la consola.
- Finalmente, debemos cerrar el fichero con Close, o de lo contrario podría quedar algún dato sin guardar.

Una forma alternativa de crear el fichero es no usar "File.CreateText", sino un constructor que reciba el nombre de fichero, así:

```
// Ejemplo_08_01b.cs
// Escritura en un fichero de texto, con constructor
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamWriter

class Ejemplo_08_01b
{
    static void Main()
    {
        StreamWriter fichero = new StreamWriter("prueba.txt");
        fichero.WriteLine("Esto es una línea");
        fichero.Write("Esto es otra");
        fichero.WriteLine(" y esto es continuación de la anterior");
        fichero.Close();
    }
}
```

Existe una tercera sintaxis, más compacta, que nos permite despreocuparnos (hasta cierto punto) de cerrar el fichero, empleando una sintaxis adicional de la orden "using" para delimitar el bloque que escribe datos en el fichero, de esta forma:

```
// Ejemplo_08_01c.cs
// Escritura en un fichero de texto, con "using"
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamWriter

class Ejemplo_08_01c
{
    static void Main()
    {
        using (StreamWriter fichero = new StreamWriter("prueba.txt"))
        {
            fichero.WriteLine("Esto es una línea");
            fichero.Write("Esto es otra");
            fichero.WriteLine(" y esto es continuación de la anterior");
        }
    }
}
```

Empleando esta construcción, el fichero se cierra automáticamente al salir de bloque "using".

Ejercicios propuestos:

(8.1.1) Crea un programa que vaya leyendo las frases que el usuario teclea y las guarde en un fichero de texto llamado "registroDeUsuario.txt". Terminará cuando la frase introducida sea "fin" (esa frase no deberá guardarse en el fichero).

(8.1.2) Crea una versión de la base de datos de ficheros (ejercicio 5.2.3), de modo que todos los datos se vuelquen a un fichero de texto al terminar la ejecución del programa. Deberás decidir qué formato usar, por ejemplo:

- Cada campo en una línea, con una línea en blanco adicional tras cada registro.
- Cada campo en una línea, pero sin línea en blanco que actúe de separador.
- Todos los campos de cada registro en una misma línea, empleando un carácter poco habitual que actué de separador de campos.
- Cada registro en una línea, además de una primera línea adicional que contenga la cantidad de registros que se van a volcar, para facilitar la lectura.
- Cualquier otra alternativa que te parezca razonable.

(8.1.3) Amplia el proyecto Libro (ejercicio 6.17.8), de modo que todos los datos se vuelquen a un fichero de texto al terminar la ejecución del programa. Nuevamente, tú decides el formato que emplearás.

8.2. Lectura de una línea de un fichero de texto

La estructura mínima de un programa que leyera desde un fichero de texto (nuevamente, sin comprobación de errores) sería parecida a ésta:

```
// Ejemplo_08_02a.cs
// Lectura parcial de un fichero de texto, sin comprobación de errores
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamReader

class Ejemplo_08_02a
{
    static void Main()
    {
        StreamReader fichero;
        string linea;

        fichero = File.OpenText("prueba.txt");
        linea = fichero.ReadLine();
        Console.WriteLine( linea );
        Console.WriteLine( fichero.ReadLine() );
```

```
        fichero.Close();
    }
}
```

Las diferencias son:

- Para leer de un fichero no usaremos StreamWriter, sino StreamReader.
- Como querremos abrir un fichero que ya existe, utilizaremos OpenText, en lugar de CreateText.
- Para leer una línea de texto del fichero, usaríamos ReadLine, de forma parecida a como hacíamos en la consola.
- Nuevamente, deberemos cerrar el fichero al terminar de usarlo.

Al igual que ocurría con la escritura en fichero, podemos emplear un constructor en vez de OpenText:

```
// Ejemplo_08_02b.cs
// Lectura de un fichero de texto, con constructor
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamReader

class Ejemplo_08_02b
{
    static void Main()
    {
        StreamReader fichero = new StreamReader("prueba.txt");
        string linea = fichero.ReadLine();
        Console.WriteLine( linea );
        Console.WriteLine( fichero.ReadLine() );
        fichero.Close();
    }
}
```

o bien utilizar la sintaxis alternativa, con la palabra "using":

```
// Ejemplo_08_02c.cs
// Lectura de un fichero de texto, con "using"
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;   // Para StreamReader

class Ejemplo_08_02c
{
    static void Main()
    {
        using (StreamReader fichero = new StreamReader("prueba.txt"))
        {
            string linea = fichero.ReadLine();
            Console.WriteLine( linea );
            Console.WriteLine( fichero.ReadLine() );
        }
```

```
    }
}
```

Ejercicios propuestos:

(8.2.1) Crea un programa que lea las tres primeras líneas del fichero creado en el ejercicio 8.1.1 y las muestre en pantalla. Usa OpenText para abrirlo.

(8.2.2) Crea una versión alternativa del ejercicio 8.2.1, usando el constructor de StreamReader.

(8.2.3) Crea una versión alternativa del ejercicio 8.2.2, empleando la sintaxis alternativa de "using".

8.3. Lectura hasta el final del fichero

Normalmente no querremos leer sólo una línea del fichero, sino procesar todo su contenido, de principio a fin.

En un fichero de texto, la forma de saber si hemos llegado al final es intentar leer una línea, y comprobar si el resultado ha sido **null**, lo que nos indicaría que no se ha podido leer y que, por tanto estamos en el final del fichero.

Habitualmente, si queremos procesar todo un fichero, esta lectura y comprobación debería ser repetitiva, así:

```
// Ejemplo_08_03a.cs
// Lectura de un fichero de texto completo
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_03a
{
    static void Main()
    {
        StreamReader fichero;
        string linea;

        fichero = File.OpenText("prueba.txt");
        do
        {
            linea = fichero.ReadLine();
            if (linea != null)
                Console.WriteLine( linea );
        }
        while (linea != null);

        fichero.Close();
    }
}
```

En principio, la estructura más natural para repetir una estructura como ésta, en la que intentamos leer y después comprobamos si se ha podido conseguir, es un bucle do-while, como en el ejemplo anterior. También se puede plantear como un "while" si la primera lectura se hace antes de la zona repetitiva del programa, así:

```
// Ejemplo_08_03b.cs
// Lectura de un fichero de texto completo (con "while")
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_03b
{
    static void Main()
    {
        StreamReader fichero;
        string linea;

        fichero = File.OpenText("prueba.txt");
        linea = fichero.ReadLine();

        while (linea != null)
        {
            Console.WriteLine( linea );
            linea = fichero.ReadLine();
        }

        fichero.Close();
    }
}
```

Por supuesto, podrías emplear el constructor o "using" para acceder al fichero. Esos cambios serán parte de los ejercicios propuestos.

Ejercicios propuestos:

(8.3.1) Crea una variante del ejemplo 08_03a, empleando un constructor en vez de "File.OpenText".

(8.3.2) Crea una variante del ejemplo 08_03a, empleando "using" en vez de "Close".

(8.3.3) Prepara un programa que pregunte un nombre de fichero y muestre en pantalla el contenido de ese fichero, haciendo una pausa después de cada 24 líneas, para que dé tiempo a leerlo. Cuando el usuario pulse la tecla Intro, se mostrarán las siguientes 24 líneas, y así sucesivamente hasta que termine el fichero. Pista: puedes usar "Console.ReadLine()" para hacer esa pausa.

(8.3.4) Amplía la base de datos de ficheros (ejercicio 8.1.2), de modo que los datos se lean desde fichero (si existe) en el momento de lanzar el programa (puedes usar try-catch para que el programa no falle en el momento inicial, en el que quizá todavía no existan datos en fichero; dentro de poco veremos formas alternativas

de saber si existe; si no recuerdas el manejo de try-catch, puedes volver al apartado 2.6 o incluso adelantar un momento al 8.7).

(8.3.5) Amplia el proyecto Libro (ejercicio 8.1.3), de modo que los datos se lean desde fichero (si existe) en el momento de poner el programa en marcha.

(8.3.6) Crea un programa que pida al usuario el nombre de un fichero de texto y una frase a buscar, y que muestre en pantalla todas las líneas de ese fichero que contengan esa frase. Cada frase se debe preceder del número de línea (la primera línea del fichero será la 1, la segunda será la 2, y así sucesivamente). Al final de la lectura se debe avisar, en caso de que no se haya encontrado el texto buscado.

(8.3.7) Crea un programa que pida al usuario el nombre de un fichero de texto y muestre en orden inverso las líneas que lo forman, empleando una pila.

(8.3.8) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en una cola, luego muestre este contenido en pantalla y finalmente lo vuelque a otro fichero de texto.

(8.3.9) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en un ArrayList, y luego pregunte de forma repetitiva al usuario qué línea desea ver y la muestre en pantalla. Terminará cuando el usuario introduzca el número de línea "-1".

(8.3.10) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en un ArrayList, y luego pregunte de forma repetitiva al usuario qué texto desea buscar y muestre las líneas que contienen ese texto. Terminará cuando el usuario introduzca una cadena vacía.

(8.3.11) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en un ArrayList, lo ordene y lo muestre ordenado en pantalla.

(8.3.12) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en un ArrayList, lo ordene y vuelque a un nuevo fichero las líneas que no estén duplicadas.

(8.3.13) Crea un programa que lea el contenido de un fichero de texto, lo almacene línea por línea en un ArrayList, luego muestre en pantalla las líneas impares (primera, tercera, etc.) y finalmente vuelque a otro fichero de texto las líneas pares (segunda, cuarta, etc.).

(8.3.14) Crea un programa que lea todo el contenido de un fichero, lo guarde en una lista de strings y luego lo muestre en orden inverso (de la última línea a la primera).

8.4. Añadir a un fichero existente

La forma de conseguir añadir a un fichero que ya existe es sencilla: en vez de abrirlo con CreateText ("crear texto"), usaremos AppendText ("añadir texto"). Por

ejemplo, podríamos crear un fichero, guardar información, cerrarlo, y luego volverlo a abrir para añadir datos, de la siguiente forma:

```
// Ejemplo_08_04a.cs
// Añadir a un fichero de texto
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_04a
{
    static void Main()
    {
        StreamWriter fichero;

        fichero = File.CreateText("prueba2.txt");
        fichero.WriteLine("Primera línea");
        fichero.Close();

        fichero = File.AppendText("prueba2.txt");
        fichero.WriteLine("Segunda línea");
        fichero.Close();
    }
}
```

También podemos usar un constructor para añadir datos a un fichero existente, pero entonces tendremos que emplear un segundo parámetro booleano, que será "true" para añadir (o "false" para sobreescribir el fichero, si ya existiera):

```
// Ejemplo_08_04b.cs
// Añadir a un fichero de texto, con constructor y "using"
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_04b
{
    static void Main()
    {
        using (StreamWriter fichero = new StreamWriter("prueba2.txt", false))
        {
            fichero.WriteLine("Primera línea");
        }

        using (StreamWriter fichero = new StreamWriter("prueba2.txt", true))
        {
            fichero.WriteLine("Segunda línea");
        }
    }
}
```

Ejercicios propuestos:

(8.4.1) Un programa que pida al usuario que teclee frases, y las almacene en el fichero "log.txt", que puede existir anteriormente (y que no deberá borrarse, sino añadir al final de su contenido). Cada sesión acabará cuando el usuario pulse Intro sin teclear nada.

8.5. Leer a un array y guardar desde un array

En la clase File existen dos métodos estáticos que permiten guardar todo un array de strings a fichero con una única orden (WriteAllLines) y cargar todo el contenido de un fichero en un array de strings con una única orden (ReadAllLines):

```
// Ejemplo_08_05a.cs
// Leer todo un array y cargar todo un array
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_05a
{
    static void Main()
    {
        // Primero guardamos datos en el fichero
        string[] frases = { "Esto", "es un", "ejemplo" };
        File.WriteAllLines("ejemplo.txt", frases);

        // Finalmente los cargamos y los mostramos
        string[] contenido = File.ReadAllLines("ejemplo.txt");
        foreach (string s in contenido)
            Console.WriteLine(s);
    }
}
```

Ejercicios propuestos:

(8.5.1) Crea una nueva versión del ejercicio 8.3.6: un programa que pida al usuario el nombre de un fichero de texto y una frase a buscar, y que muestre en pantalla todas las líneas de ese fichero que contengan esa frase. Cada frase se debe preceder del número de línea (la primera línea del fichero será la 1, la segunda será la 2, y así sucesivamente). Al final de la lectura se debe avisar, en caso de que no se haya encontrado el texto buscado. Debes leer todos los datos a un array.

(8.5.2) Crea una nueva versión del ejercicio 8.3.7: un programa que pida al usuario el nombre de un fichero de texto y muestre en orden inverso las líneas que lo forman, tras leer todo su contenido a un array.

(8.5.3) Crea un programa que lea el contenido de un fichero de texto, lo almacene en un array, lo ordene y lo vuelque a un nuevo fichero.

8.6. Leer a una única cadena de texto

De forma alternativa, también se puede leer todo un fichero **a una única cadena de texto**, que contendrá saltos de línea intermedios (\n, \r o ambos, según el sistema operativo), usando dos construcciones alternativas: o bien con el método ReadToEnd de la clase StreamReader:

```
using (StreamReader fichero = new StreamReader("prueba.txt")
{
    string texto = fichero.ReadToEnd( );
}
```

O bien simplemente con File.ReadAllText:

```
string texto = File.ReadAllText("prueba.txt");
```

Ejercicios propuestos:

(8.6.1) Crea un programa que pida al usuario el nombre de un fichero de texto, lea todas sus líneas en una única cadena de texto (con ReadToEnd o ReadAllText), y diga cuantas veces contiene la letra "a" (independientemente de si está en mayúsculas o minúsculas).

8.7. Ficheros en otras carpetas

Si un fichero al que queremos acceder se encuentra en otra carpeta, basta con que el nombre de fichero incluya también la ruta.

Debemos recordar que la barra invertida que se usa en sistemas Windows para separar los nombres de los directorios, coincide con el carácter de control que se usa en las cadenas de C# (las "secuencias de escape", que vimos en el apartado 3.3.2). Por ello, en las rutas deberemos escribir dichas barras invertidas repetidas, así:

```
string nombreFichero = "d:\\ejemplos\\ejemplo1.txt";     // Ruta absoluta
string nombreFichero2 = "..\\datos\\configuracion.txt";  // Ruta relativa
```

Esta sintaxis puede llegar a resultar incómoda, por lo que en C# existe una alternativa: incluir una arroba (@) justo antes de abrir las comillas, y entonces no será necesario "escapar" los caracteres especiales:

```
string nombreFichero = @"d:\ejemplos\ejemplo1.txt";
```

Ejercicios propuestos:

(8.7.1) Crea un programa que pida al usuario pares de números enteros y escriba su suma (con el formato "20 + 3 = 23") en pantalla y en un fichero llamado "sumas.txt", que se encontrará en un subdirectorio llamado "resultados". Cada vez que se ejecute el programa, deberá añadir los nuevos resultados a continuación de los resultados de las ejecuciones anteriores.

8.8. Saber si un fichero existe

Hasta ahora, estamos intentando abrir ficheros para lectura, pero sin comprobar realmente si el fichero existe o no, lo que puede suponer que nuestro programa falle en caso de que el fichero no se encuentre donde nosotros esperamos o de que introduzcamos un nombre incorrecto.

Una primera solución es comprobar si está, antes de intentar abrirlo, usando "File.Exists(nombre)", que devuelve un valor booleano:

```
// Ejemplo_08_08a.cs
// Saber si un fichero existe
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_08a
{
    static void Main()
    {
        StreamReader fichero;
        string nombre;

        while (true) // Interrumpiremos con "break" (no recomendable)
        {
            Console.Write( "Nombre del fichero (\"fin\" para terminar)? ");
            nombre = Console.ReadLine();
            if (nombre == "fin")
                break;
            if ( File.Exists(nombre) )
            {
                fichero = File.OpenText( nombre );
                Console.WriteLine("Su primera linea es: {0}",
                fichero.ReadLine() );
                fichero.Close();
            }
            else
                Console.WriteLine( "No existe!" );
        }
    }
}
```

Ejercicios propuestos:

(8.8.1) Mejora el ejercicio 8.5.1 (buscar texto en un fichero) para que compruebe antes si el fichero existe, y muestre un mensaje de aviso en caso de que no sea así.

(8.8.2) Mejora el ejemplo 08_08a para que no use "while (true)" y "break", sino una variable booleana de control.

Otra forma de comprobar si un fichero existe o no, es usar "**excepciones**", con las que ya nos habíamos encontrado en el tema 2 y que veremos con más detalle en el próximo apartado.

8.9. Más comprobaciones de errores: excepciones

"File.Exists" nos permite saber si un fichero existe, pero ese no es el único problema que podemos encontrar al acceder a un fichero. Puede ocurrir que exista pero no tengamos permiso para acceder a él, o que intentemos escribir en un dispositivo que sea sólo de lectura (como un CD-ROM, un DVD-ROM o ciertas carpetas de red, por ejemplo).

Por ello, una forma más eficaz de comprobar si ha existido algún tipo de error es revisar las posibles "excepciones", con las que ya tuvimos un contacto al final del tema 2.

Típicamente, los pasos que puedan ser problemáticos irán dentro del bloque "try" y las acciones correctoras y/o mensajes de error estarán en el bloque "catch". Así, un primer ejemplo, que mostrara todo el contenido de un fichero de texto y que en caso de error se limitase a mostrar un mensaje y a abandonar el programa, podría ser:

```
// Ejemplo_08_09a.cs
// Excepciones y ficheros (1)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_09a
{
    static void Main()
    {
        StreamReader fichero;
        string nombre;
        string linea;

        Console.WriteLine("Introduzca el nombre del fichero");
        nombre = Console.ReadLine();

        try
```

```
        {
            fichero = File.OpenText(nombre);
            do
            {
                linea = fichero.ReadLine();
                if (linea != null)
                    Console.WriteLine( linea );
            }
            while (linea != null);

            fichero.Close();
        }
        catch (Exception exp)
        {
            Console.WriteLine("Ha habido un error: {0}", exp.Message);
            return;
        }
    }
}
```

El resultado, si ese fichero no existe, sería algo como

```
Introduzca el nombre del fichero
prueba
Ha habido un error: No se pudo encontrar el archivo
'C:\Fuentes\nacho\prueba'.
```

Pero, como ya vimos, generalmente lo razonable no es interceptar "todas las excepciones a la vez", sino crear un análisis para cada caso, que permita recuperarse del error y seguir adelante, para lo que se suelen crear varios bloques "catch". Por ejemplo, en el caso de que queramos crear un fichero, podemos tener excepciones como éstas:

- El fichero existe y es de sólo lectura (se lanzará una excepción del tipo "IOException").
- La ruta del fichero es demasiado larga (excepción de tipo "PathTooLongException").
- El disco puede estar lleno (IOException).

Así, dentro de cada bloque "catch" podríamos indicar una excepción más concreta que una simple "Exception", de forma que el mensaje de aviso sea más concreto, o que podamos dar pasos más adecuados para solucionar el problema:

```
// Ejemplo_08_09b.cs
// Excepciones y ficheros (2)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;
```

```
class Ejemplo_08_09b
{
    static void Main()
    {
        StreamWriter fichero;
        string nombre;
        string linea;

        Console.Write("Introduzca el nombre del fichero: ");
        nombre = Console.ReadLine();
        Console.Write("Introduzca la frase a guardar: ");
        linea = Console.ReadLine();

        try
        {
            fichero = File.CreateText(nombre);
            fichero.WriteLine( linea );
            fichero.Close();
        }
        catch (PathTooLongException e)
        {
            Console.WriteLine("Nombre demasiado largo!");
        }
        catch (IOException e)
        {
            Console.WriteLine("No se ha podido escribir!");
            Console.WriteLine("El error exacto es: {0}", e.Message);
        }
    }
}
```

Las excepciones **más generales** (IOException y, sobre todo, Exception) deberán ser las **últimas** que analicemos, y las más específicas deberán ser las primeras.

Hay que recordar que, como ya vimos, se puede usar "try...catch" para comprobar también otros errores relacionados con la entrada de datos, como los problemas de conversión de datos (por ejemplo, si esperamos que el usuario introduzca un número, pero, en lugar de ello, ha tecleado un texto).

Ejercicios propuestos:

(8.9.1) Un programa que pida al usuario el nombre de un fichero de origen y el de un fichero de destino, y que vuelque al segundo fichero el contenido del primero, convertido a mayúsculas. Se debe controlar los posibles errores, como que el fichero de origen no exista, o que el fichero de destino no se pueda crear.

(8.9.2) Un programa que pida al usuario un número, una operación (+, -, *, /) y un segundo número, y muestre el resultado de la correspondiente operación. Si se teclea un dato no numérico, el programa deberá mostrar un aviso y volver a pedirlo, en vez de interrumpir la ejecución.

(8.9.3) Un programa que pida al usuario repetidamente pares de números y la operación a realizar con ellos (+, -, *, /) y guarde en un fichero "calculadora.txt" el

resultado de dichos cálculos (con la forma "15 * 6 = 90"). Debe controlar los posibles errores, como que los datos no sean numéricos, la división entre cero, o que el fichero no se haya podido crear.

8.10. Conceptos básicos sobre ficheros

Llega el momento de concretar algunos conceptos que hemos pasado por encima, y que es necesario conocer:

- Fichero **físico**. Es un fichero que existe realmente en el disco, como "agenda.txt".
- Fichero **lógico**. Es un identificador que aparece dentro de nuestro programa, y que permite acceder a un fichero físico. Por ejemplo, si declaramos "StreamReader fichero1", esa variable "fichero1" representa un fichero lógico.
- **Equivalencia** entre fichero lógico y fichero físico: no necesariamente tiene por qué existir una correspondencia "1 a 1". Puede que accedamos a un fichero físico mediante dos o más ficheros lógicos (por ejemplo, un StreamReader para leer de él y un StreamWriter para escribir en él), y también podemos usar un único fichero lógico para acceder a distintos ficheros físicos (por ejemplo, los mapas de los niveles de un juego, que estén guardados en distintos ficheros físicos, pero los abramos y los leamos utilizando siempre una misma variable).
- Un **registro** es un bloque de datos que forma un "todo", como el conjunto de los datos de una persona: nombre, dirección, e-mail, teléfono, etc. Cada uno de esos "apartados" de un registro se conoce como "campo".
- Acceso **secuencial**: Cuando debemos "recorrer" todo el contenido de un fichero si queremos llegar hasta cierto punto (como ocurría con las cintas de video o de cassette). Es lo que estamos haciendo hasta ahora en los ficheros de texto.
- Acceso **aleatorio**: Cuando podemos saltar hasta cualquier posición del fichero directamente, sin necesidad de recorrer todo lo anterior (como se puede hacer con un Compact Disc de música o un reproductor de MP3). Es algo que comenzaremos a hacer pronto, en los ficheros binarios. Por el contrario, en un fichero de texto, habitualmente no podremos hacerlo, porque no sabremos el tamaño de cada línea de texto, así que no podremos saltar hasta una cierta línea sin leer las anteriores.

8.11. Leer un byte de un fichero binario: FileStream

Los ficheros de texto son habituales, pero es aún más frecuente encontrarnos con ficheros en los que la información está estructurada como una secuencia de bytes, más o menos ordenada, pero que quizá no sea texto puro (o al menos no por completo). Esto ocurre en las imágenes, los ficheros de sonido, de video, etc.

Vamos a ver cómo leer de un fichero "general", y lo aplicaremos a descifrar la información almacenada en ciertos formatos habituales, como una imagen BMP o un fichero de sonido MP3.

Como primer acercamiento, vamos a ver cómo abrir un fichero (no necesariamente de texto, aunque también podría serlo) y leer el primer byte que contiene. Comenzaremos por utilizar la clase **FileStream**, diseñada para acceder a un byte o a bloques de bytes. Esta clase tiene un método **ReadByte**, que devuelve un número entero, que tendrá valor -1 en caso de error, o bien será un dato que podríamos convertir a byte si la lectura ha sido correcta, así:

```
// Ejemplo_08_11a.cs
// Ficheros binarios: lectura de un byte con FileStream
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_11a
{
    static void Main()
    {
        FileStream fichero;
        string nombre;
        byte unDato;

        Console.WriteLine("Introduzca el nombre del fichero");
        nombre = Console.ReadLine();

        try
        {
            fichero = File.OpenRead(nombre);
            unDato = (byte) fichero.ReadByte();
            Console.WriteLine("El byte leido es un {0}",
                unDato);
            fichero.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("Error: "+e.Message);
            return;
        }
    }
}
```

En vez de File.OpenRead, también podemos usar un **constructor** para acceder al fichero. Este constructor recibirá dos parámetros: el nombre del fichero y el modo de apertura. Por ahora, nos interesará un único modo, que será "FileMode.Open", que nos permite leer datos desde un fichero que ya existe:

```
// Ejemplo_08_11b.cs
// Ficheros binarios: lectura de un byte con FileStream, constructor
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_11b
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        try
        {
            FileStream fichero = new FileStream(nombre, FileMode.Open);
            byte unDato = (byte) fichero.ReadByte();
            Console.WriteLine("El byte leido es un {0}",
                unDato);
            fichero.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("Error: "+e.Message);
            return;
        }
    }
}
```

Y también podemos emplear la sintaxis alternativa, con "**using**", de modo que no sea necesario cerrar el fichero de forma explícita, sino que se cierre automáticamente a la salida del bloque:

```
// Ejemplo_08_10c.cs
// Ficheros binarios: lectura de un byte con FileStream, using
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_10c
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        try
        {
```

```
            using (FileStream fichero = new FileStream(
                nombre, FileMode.Open))
            {
                byte unDato = (byte) fichero.ReadByte();
                Console.WriteLine("El byte leido es un {0}",
                    unDato);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("Error: "+e.Message);
            return;
        }
    }
}
```

Los otros **modos de apertura** posibles (cuyos nombres en inglés son casi autoexplicativos) son:

- **Append**, añade al final o crea si no existe.
- **Create**, crea el fichero o lo sobrescribe si ya existía.
- **CreateNew**, crea si no existía, o falla si existe.
- **Open**, abre si existe o falla si no existe.
- **OpenOrCreate**, abre si existe o lo crea si no existe
- **Truncate**, abre un fichero que debe existir y lo vacía.

Si quisiéramos **leer un segundo dato**, volveríamos a llamar a ReadByte, porque tras cada lectura, avanza automáticamente la posición actual dentro del fichero.

Ejercicios propuestos:

(8.11.1) Abre un fichero con extensión EXE (cuyo nombre introducirá el usuario) y comprueba si realmente se trata de un ejecutable, mirando si los dos primeros bytes del fichero son un 77 (que corresponde a una letra "M", según el estándar que marca el código ASCII) y un 90 (una letra "Z"), respectivamente.

(8.11.2) Abre una imagen BMP (cuyo nombre introducirá el usuario) y comprueba si realmente se trata de un fichero en ese formato, mirando si los dos primeros bytes del fichero corresponden a una letra "B" y una letra "M", respectivamente.

(8.11.3) Abre una imagen GIF y comprueba si sus tres primeros bytes son las letras G, I, F.

8.12. Leer hasta el final de un fichero binario

Tenemos dos formas de leer byte a byte todos datos que forma un FileStream. La primera es descubrir el tamaño del fichero, que podemos obtener con ".**Length**". A partir de entonces, podríamos usar un bucle "for" para recorrer todo el fichero,

como en el siguiente programa, que cuenta la cantidad de veces que aparece "A" dentro de un fichero:

```
// Ejemplo_08_12a.cs
// Ficheros binarios: lectura completa de un FileStream (1)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_12a
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();
        FileStream fichero = File.OpenRead(nombre);
        int cantidad = 0;
        for (int pos = 0; pos < fichero.Length; pos++)
        {
            byte unDato = (byte) fichero.ReadByte();
            char letra = Convert.ToChar( unDato );
            if (letra == 'A')
                cantidad++;
        }
        Console.WriteLine("Tiene {0} letras A", cantidad);
        fichero.Close();
    }
}
```

(Por simplicidad, este fuente no realiza ningún tipo de **comprobación de errores**. Lo mismo ocurrirá en la mayoría de fuentes de ejemplo de este tema, pero un programa real sí debería comprobar si el fichero existe –con File.Exists- y si se ha producido algún error durante la lectura –con try...catch-).

La segunda forma de leer hasta el final del fichero es comprobar si el dato leído es un -1. Cuando esto ocurra, será la señal de que se ha llegado al final del fichero. El fuente correspondiente podría ser:

```
// Ejemplo_08_12b.cs
// Ficheros binarios: lectura completa de un FileStream (2)
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_12b
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();
        FileStream fichero = File.OpenRead(nombre);
        int cantiddad = 0;
```

```
        int dato;
        do
        {
            dato = fichero.ReadByte();
            if (dato != -1)
            {
                char letra = Convert.ToChar( (byte) dato );
                if (letra == 'A')
                    cantidad++;
            }
        }
        while (dato != -1);
        Console.WriteLine("Tiene {0} letras A", cantidad);
        fichero.Close();
    }
}
```

Ejercicios propuestos:

(8.12.1) Crea un programa que compruebe si un fichero (cuyo nombre introducirá el usuario) contiene una cierta letra (también escogida por el usuario).

(8.12.2) Crea un programa que cuente la cantidad de vocales que contiene un fichero binario.

(8.12.3) Crea un programa que diga si un fichero (binario) contiene una cierta palabra que introduzca el usuario.

(8.12.4) Crea un programa que extraiga a un fichero de texto todos los caracteres alfabéticos que contenga un fichero binario, que son los siguientes:

- Códigos 32 a 127 (caracteres visibles del alfabeto inglés y símbolos de puntuación).
- Carácter 9 (TAB, salto de tabulación)
- Carácter 10 (LF, avance de línea)
- Carácter 13 (CR, retorno de carro)

No se deben volcar los demás datos.

8.13. Leer bloques de datos de un fichero binario

Leer byte a byte puede resultar sencillo, pero también es lento. Por eso, en la práctica es más habitual leer grandes bloques de datos. Típicamente, esos datos se guardarán en memoria como un array de bytes.

Para eso, dentro de la clase "FileStream" tenemos un método "**Read**", que nos permite leer una cierta cantidad de datos desde el fichero. Le indicaremos en qué array guardar esos datos, a partir de qué posición del array debe introducir los datos (no la posición en el fichero, sino en el array, de modo que casi siempre será 0, es decir, al principio del array), y qué cantidad de datos se deben leer. Nos

devuelve un valor, que es la cantidad de datos que se han podido leer realmente (porque puede ocurrir que sea menos de lo que le hemos pedido, si hay un error de lectura o si se alcanza el final del fichero).

Así, el siguiente ejemplo lee los 10 primeros bytes de un fichero:

```
// Ejemplo_08_13a.cs
// Ficheros binarios: lectura de un bloque de 10 bytes con FileStream
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_13a
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        FileStream fichero = File.OpenRead(nombre);
        byte[] datos = new byte[10];
        int posicion = 0;
        int cantidadALeer = 10;
        int cantidadLeida = fichero.Read(datos, posicion, cantidadALeer);

        if (cantidadLeida < 10)
            Console.WriteLine("No se han podido leer los 10 bytes!");
        else
        {
            Console.WriteLine("El primer byte leido es {0}", datos[0]);
            Console.WriteLine("El tercero es {0}", datos[2]);
        }
        fichero.Close();
    }
}
```

Ejercicios propuestos:

(8.13.1) Abre un fichero con extensión EXE y comprueba si realmente se trata de un ejecutable, mirando si los dos primeros bytes del fichero corresponden a una letra "M" y una letra "Z", respectivamente. Debes leer ambos bytes a la vez, usando un array.

(8.13.2) Abre una imagen BMP (cuyo nombre introducirá el usuario) y comprueba si realmente se trata de un fichero en ese formato, mirando si los dos primeros bytes del fichero corresponden a una letra "B" y una letra "M", respectivamente. Usa "Read" y un array.

(8.13.3) Abre una imagen GIF y comprueba si sus tres primeros bytes son las letras G, I, F. Guarda los 3 datos en un array.

(8.13.4) Abre una imagen en formato BMP y comprueba si está comprimida, mirando el valor del byte en la posición 30 (empezando a contar desde 0). Si ese

valor es un 0 (que es lo habitual), indicará que el fichero no está comprimido. Deberás leer toda la cabecera (los primeros 54 bytes) con una sola orden.

8.14. La posición en el fichero

Cuando leemos byte a byte (y también a veces en el caso de leer todo un bloque), habitualmente nos interesará situarnos previamente en la posición exacta en la que se encuentra el dato que deseamos leer. En un fichero binario, esto podemos conseguirlo sin necesidad de leer todos los bytes anteriores, accediendo directamente a la posición que buscamos (es lo que se conoce como "acceso aleatorio").

Para ello, tenemos el método "**Seek**". A este método se le indican dos parámetros: la posición a la que queremos saltar, y el punto desde el que queremos que se cuente esa posición, que puede ser:

- SeekOrigin.Begin: desde el comienzo del fichero
- SeekOrigin.Current: desde la posición actual
- SeekOrigin.End: desde el final del fichero

La posición es un Int64, porque puede ser un número muy grande e incluso un número negativo (especialmente habitual si miramos desde el final del fichero, porque desde él habrá que retroceder).

De igual modo, podemos descubrir en qué posición del fichero nos encontramos, consultando la propiedad "**Position**" (y, como ya sabemos, también podemos obtener la longitud del fichero, mirando su propiedad "Length").

```
// Ejemplo_08_14a.cs
// Ficheros binarios: posición
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_14a
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();
        FileStream  fichero = File.OpenRead(nombre);
        byte[] datos = new byte[10];
        int posicion = 0;
        int cantidadALeer = 10;
        int cantidadLeida = fichero.Read(datos, posicion, cantidadALeer);

        if (cantidadLeida < 10)
            Console.WriteLine("No se han podido leer todos los datos!");
        else
```

```
        {
            Console.WriteLine("El primer byte leido es {0}", datos[0]);
            Console.WriteLine("El tercero es {0}", datos[2]);
        }

        if (fichero.Length > 30)
        {
            fichero.Seek(19, SeekOrigin.Begin);
            int nuevoDato = fichero.ReadByte();
            Console.WriteLine("El byte 20 es un {0}", nuevoDato);

            Console.WriteLine("La posición actual es {0}",
                fichero.Position);
            Console.WriteLine("Y el tamaño del fichero es {0}",
                fichero.Length);
        }

        fichero.Close();
    }
}
```

(Nota: existe una propiedad "CanSeek" que nos permite saber si el fichero que hemos abierto permite realmente que nos movamos a unas posiciones u otras –lo habitual es que sí sea posible hacerlo-).

Ejercicios propuestos:

(8.14.1) Abre un fichero con extensión EXE y comprueba si su segundo byte corresponde a una letra "Z", sin leer su primer byte.

(8.14.2) Abre una imagen en formato BMP y comprueba si está comprimida, mirando el valor del byte en la posición 30 (empezando a contar desde 0). Si ese valor es 0 (que es lo habitual), indicará que el fichero no está comprimido. Salta a esa posición directamente, sin leer toda la cabecera.

8.15. Leer datos nativos: BinaryReader

Un FileStream es cómodo para leer byte a byte, pero no tanto cuando queremos leer otros tipos de datos básicos existentes en C# (short, int, float, etc.). Para eso usaremos la clase "BinaryReader". Por ejemplo, podríamos leer el entero corto que forman los dos bytes iniciales de un fichero usando un "BinaryReader" y su método "ReadInt16", así:

```
// Ejemplo_08_15a.cs
// Ficheros binarios: lectura de un short, con BinaryReader
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_15a
{
```

```
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        BinaryReader fichero = new BinaryReader(
            File.Open(nombre, FileMode.Open));
        short dato = fichero.ReadInt16();
        Console.WriteLine("El dato leido es {0}", dato);
        fichero.Close();
    }
}
```

Si preferimos hacer esta lectura byte a byte, podremos usar tanto un FileStream como un BinaryReader, pero deberemos recomponer ese "short" a partir de varios bytes, teniendo en cuenta que, en la mayoría de sistemas actuales, en primer lugar aparece el byte menos significativo y luego el byte más significativo (lo que se conoce como sistema "**little endian**"), de modo que el dato original de 16 bits se obtendría con la operación "byte1 + byte2*256", así:

```
// Ejemplo_08_15b.cs
// Ficheros binarios: lectura de un short, byte a byte
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_15b
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        BinaryReader fichero = new BinaryReader(
            File.Open(nombre, FileMode.Open));
        short dato1 = fichero.ReadByte();
        short dato2 = fichero.ReadByte();
        Console.WriteLine("El dato leido es {0}", dato1 + dato2*256);
        fichero.Close();
    }
}
```

También podemos emplear la cláusula "using" para los BinaryReader, de modo que el fichero se cierre automáticamente, así:

```
// Ejemplo_08_15c.cs
// Ficheros binarios: lectura de un short, con BinaryReader, using
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_15c
```

```
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero");
        string nombre = Console.ReadLine();

        using (BinaryReader fichero = new BinaryReader(
            File.Open(nombre, FileMode.Open)))
        {
            short dato = fichero.ReadInt16();
            Console.WriteLine("El dato leido es {0}", dato);
        }
    }
}
```

Ejercicios propuestos:

(8.15.1) Abre un fichero con extensión EXE y comprueba si comienza con el entero corto 23117.

(8.15.2) Abre una imagen en formato BMP y comprueba si comienza con el entero corto 19778.

También podemos emplear "**Seek**" para movernos a un punto u otro de un "BinaryReader", pero está un poco más escondido que en "FileStream": no se lo pedimos directamente a nuestro fichero, sino al "Stream" (flujo de datos) que hay por debajo, a su "BaseStream", así:

```
ficheroEntrada.BaseStream.Seek( 1, SeekOrigin.Begin);
```

Por ejemplo, la anchura de una imagen en formato BMP es un entero de 32 bits que se encuentra en la posición 18, de modo que podríamos hacer:

```
// Ejemplo_08_15d.cs
// Ficheros binarios: Seek, con BinaryReader
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_15d
{
    static void Main()
    {
        Console.WriteLine("Introduzca el nombre del fichero BMP");
        string nombre = Console.ReadLine();

        BinaryReader fichero = new BinaryReader(
            File.Open(nombre, FileMode.Open));
        fichero.BaseStream.Seek(18, SeekOrigin.Begin);
        int ancho = fichero.ReadInt32();
```

```
        Console.WriteLine("El ancho es {0}", ancho);
        fichero.Close();
    }
}
```

Ejercicios propuestos:

(8.15.3) El alto de un fichero BMP es un entero de 32 bits que se encuentra en la posición 22. Amplía el ejemplo 08_15d, para que muestre también el alto de ese fichero BMP.

8.16. Ejemplo completo: leer información de un fichero BMP

Hemos estado extrayendo algo de información de ficheros BMP. Ahora vamos a ir un poco más allá y a tratar de obtener muchos de los detalles que hay en su cabecera. El formato de dicha cabecera es el siguiente:

Un fichero BMP está compuesto por las siguientes partes: una cabecera de fichero, una cabecera del bitmap, una tabla de colores y los bytes que definirán la imagen.

En concreto, los datos que forman la cabecera de fichero y la cabecera de bitmap son los siguientes:

TIPO DE INFORMACIÓN	POSICIÓN EN EL ARCHIVO
Tipo de fichero (letras BM)	0-1
Tamaño del archivo	2-5
Reservado	6-7
Reservado	8-9
Inicio de los datos de la imagen	10-13
Tamaño de la cabecera de bitmap	14-17
Anchura (píxeles)	18-21
Altura (píxeles)	22-25
Número de planos	26-27
Tamaño de cada punto	28-29
Compresión (0=no comprimido)	30-33
Tamaño de la imagen	34-37
Resolución horizontal	38-41
Resolución vertical	42-45
Tamaño de la tabla de color	46-49
Contador de colores importantes	50-53

Por tanto, será fácil hacer que se nos muestren algunos detalles, como su ancho, su alto, la resolución y si la imagen está comprimida o no:

```
// Ejemplo_08_16a.cs
// Información de un fichero BMP, con BinaryReader
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_16a
{
    static void Main()
    {
        Console.WriteLine("Información de imágenes BMP");
        Console.WriteLine("Dime el nombre del fichero: ");
        string nombre = Console.ReadLine();

        if (! File.Exists( nombre) )
        {
            Console.WriteLine("No encontrado!");
        }
        else
        {
            BinaryReader fichero = new BinaryReader(
                File.Open(nombre, FileMode.Open));

            // Leo los dos primeros bytes
            char marca1 = Convert.ToChar( fichero.ReadByte() );
            char marca2 = Convert.ToChar( fichero.ReadByte() );

            if ((marca1 !='B') || (marca2 !='M'))
                Console.WriteLine("No parece un fichero BMP");
            else
            {
                Console.WriteLine("Marca del fichero: {0}{1}",
                    marca1, marca2);

                // Posición 18: ancho
                fichero.BaseStream.Seek(18, SeekOrigin.Begin);
                int ancho = fichero.ReadInt32();
                Console.WriteLine("Ancho: {0}", ancho);

                // A continuación: alto
                int alto = fichero.ReadInt32();
                Console.WriteLine("Alto: {0}", alto);

                // Posición 30: compresión
                fichero.BaseStream.Seek(30, SeekOrigin.Begin);
                int compresion = fichero.ReadInt32();
                switch (compresion)
                {
                    case 0: Console.WriteLine("Sin compresión");
                            break;
                    case 1: Console.WriteLine("Compresión RLE 8 bits");
                            break;
                    case 2: Console.WriteLine("Compresión RLE 4 bits");
                            break;
                }
```

```
                // 4 bytes después: resolución horizontal
                fichero.BaseStream.Seek(4, SeekOrigin.Current);
                int resolH = fichero.ReadInt32();
                Console.WriteLine("Resolución Horiz.: {0}", resolH);

                // A continuación: resolución vertical
                int resolV = fichero.ReadInt32();
                Console.WriteLine("Resolución Vertical: {0}", resolV);

                fichero.Close();
            }
        }
    }
}
```

Podemos hacerlo también con un FileStream, que simplificará ciertas operaciones, como la lectura, que se puede hacer toda en un bloque, pero a cambio complicará otras operaciones, como el cálculo de los valores enteros, que habrá que componer a partir de los 4 bytes que los forman, recordando que el primer byte es el menos significativo:

```
// Ejemplo_08_16b.cs
// Información de un fichero BMP, con FileStream
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_16b
{
    static void Main()
    {
        Console.WriteLine("Información de imágenes BMP");
        Console.WriteLine("Dime el nombre del fichero: ");
        string nombre = Console.ReadLine();

        if (! File.Exists( nombre) )
        {
            Console.WriteLine("No encontrado!");
        }
        else
        {
            FileStream fichero = File.OpenRead(nombre);
            int tamanyoCabecera = 54;
            byte[] cabecera = new byte[tamanyoCabecera];

            int cantidadLeida = fichero.Read(cabecera, 0, tamanyoCabecera);
            fichero.Close();

            if (cantidadLeida != tamanyoCabecera)
            {
                Console.WriteLine("No se ha podido leer la cabecera");
            }
            else
            {
                // Analizo los dos primeros bytes
```

```
                char marca1 = Convert.ToChar( cabecera[0] );
                char marca2 = Convert.ToChar( cabecera[1] );

                if ((marca1 !='B') || (marca2 !='M'))
                    Console.WriteLine("No parece un fichero BMP");
                else
                {
                    Console.WriteLine("Marca del fichero: {0}{1}",
                        marca1, marca2);

                    int ancho = cabecera[18] + // Convierto 4 bytes a Int32
                        cabecera[19] * 256 +
                        cabecera[20] * 256 * 256 +
                        cabecera[21] * 256 * 256 * 256;
                    Console.WriteLine("Ancho: {0}", ancho);

                    int alto = cabecera[22] +
                        cabecera[23] * 256 +
                        cabecera[24] * 256 * 256 +
                        cabecera[25] * 256 * 256 * 256;
                    Console.WriteLine("Alto: {0}", alto);

                    int compresion = cabecera[30];
                    switch (compresion)
                    {
                        case 0: Console.WriteLine("Sin compresión");
                            break;
                        case 1: Console.WriteLine("Compresión RLE 8 bits");
                            break;
                        case 2: Console.WriteLine("Compresión RLE 4 bits");
                            break;
                    }

                    int resolH = cabecera[38] +
                        cabecera[39] * 256 +
                        cabecera[40] * 256 * 256 +
                        cabecera[41] * 256 * 256 * 256;
                    Console.WriteLine("Resolución Horiz.: {0}", resolH);

                    int resolV = cabecera[42] +
                        cabecera[43] * 256 +
                        cabecera[44] * 256 * 256 +
                        cabecera[45] * 256 * 256 * 256;
                    Console.WriteLine("Resolución Vertical: {0}", resolV);
                }
            }
        }
    }
}
```

Ejercicios propuestos:

(8.16.1) Localiza en Internet información sobre el formato de imágenes PCX. Crea un programa que diga el ancho, alto y número de colores de una imagen PCX.

(8.16.2) Localiza en Internet información sobre el formato de imágenes GIF. Crea un programa que diga el subformato, ancho, alto y número de colores de una imagen GIF.

8.17. Escribir en un fichero binario

Para escribir en un FileStream, usaremos órdenes similares a las que empleábamos para leer de él:

- Un método WriteByte, para escribir sólo un byte, o bien...
- Un método Write, para escribir un bloque de información (desde cierta posición de un array -normalmente 0-, y con cierto tamaño).

Además, a la hora de abrir el fichero, tenemos dos alternativas:

- Abrir un fichero existente con "OpenWrite".
- Crear un nuevo fichero con "Create".

Vamos a ver un ejemplo que junte todo ello: crearemos un fichero, guardaremos datos, lo leeremos para comprobar que todo es correcto, añadiremos al final, y volveremos a leer:

```
// Ejemplo_08_17a.cs
// Ficheros binarios: escritura en FileStream
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_17a
{
    const int TAMANYO_BUFFER = 10;

    static void Main()
    {
        FileStream fichero;
        string nombre;
        byte[] datos;

        nombre = "datos.dat";
        datos = new byte[TAMANYO_BUFFER];

        // Damos valores iniciales al array
        for (byte i=0; i<TAMANYO_BUFFER; i++)
            datos[i] = (byte) (i + 10);

        try
        {
            // Primero creamos el fichero, con algún dato
            fichero = File.Create( nombre );
            fichero.Write(datos, 0, TAMANYO_BUFFER);
            fichero.Close();
```

```
            // Ahora leemos dos datos
            fichero = File.OpenRead(nombre);
            Console.WriteLine("El tamaño es {0}", fichero.Length);

            fichero.Seek(2, SeekOrigin.Begin);
            int nuevoDato = fichero.ReadByte();
            Console.WriteLine("El tercer byte es un {0}", nuevoDato);

            fichero.Seek(-2, SeekOrigin.End);
            nuevoDato = fichero.ReadByte();
            Console.WriteLine("El penultimo byte es un {0}", nuevoDato);

            fichero.Close();

            // Ahora añadimos 10 datos más, al final
            fichero = File.OpenWrite(nombre);
            fichero.Seek(0, SeekOrigin.End);
            fichero.Write(datos, 0, TAMANYO_BUFFER);

            // y modificamos el tercer byte
            fichero.Seek(2, SeekOrigin.Begin);
            fichero.WriteByte( 99 );
            fichero.Close();

            // Volvemos a leer algún dato
            fichero = File.OpenRead(nombre);
            Console.WriteLine("El tamaño es {0}", fichero.Length);
            fichero.Seek(2, SeekOrigin.Begin);
            nuevoDato = fichero.ReadByte();
            Console.WriteLine("El tercer byte es un {0}", nuevoDato);
            fichero.Close();

        }
        catch (Exception e)
        {
            Console.WriteLine("Problemas: "+e.Message);
            return;
        }
    }
}
```

(Nota: existe una propiedad "CanWrite" que nos permite saber si se puede escribir en el fichero).

Si queremos que escribir datos básicos de C# (float, int, etc.) en vez de un array de bytes, podemos usar un "**BinaryWriter**", que se maneja de forma similar a un "BinaryReader", con la diferencia de que no tenemos métodos WriteByte, WriteString y similares, sino un único método "Write", que se encarga de escribir el dato que le indiquemos, sea del tipo que sea (ya que está sobrecargado):

```
// Ejemplo_08_17b.cs
// Ficheros binarios: escritura en BinaryWriter
// Introducción a C#, por Nacho Cabanes
```

```
using System;
using System.IO;

class Ejemplo_08_17b
{
    static void Main()
    {
        BinaryWriter ficheroSalida;
        BinaryReader ficheroEntrada;
        string nombre;

        // Los datos que vamos a guardar/leer
        byte unDatoByte;
        int unDatoInt;
        float unDatoFloat;
        double unDatoDouble;
        string unDatoString;

        Console.Write("Introduzca el nombre del fichero a crear: ");
        nombre = Console.ReadLine();

        Console.WriteLine("Creando fichero...");
        // Primero vamos a grabar datos
        try
        {
            ficheroSalida = new BinaryWriter(
            File.Open(nombre, FileMode.Create));
            unDatoByte = 1;
            unDatoInt = 2;
            unDatoFloat = 3.0f;
            unDatoDouble = 4.0;
            unDatoString = "Hola";
            ficheroSalida.Write(unDatoByte);
            ficheroSalida.Write(unDatoInt);
            ficheroSalida.Write(unDatoFloat);
            ficheroSalida.Write(unDatoDouble);
            ficheroSalida.Write(unDatoString);
            ficheroSalida.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("Problemas al crear: "+e.Message);
            return;
        }

        // Ahora vamos a leerlos
        Console.WriteLine("Leyendo fichero...");
        try
        {
            ficheroEntrada = new BinaryReader(
            File.Open(nombre, FileMode.Open));

            unDatoByte = ficheroEntrada.ReadByte();
            Console.WriteLine("El byte leido es un {0}",
                unDatoByte);

            unDatoInt = ficheroEntrada.ReadInt32();
            Console.WriteLine("El int leido es un {0}",
                unDatoInt);
```

```
            unDatoFloat = ficheroEntrada.ReadSingle();
            Console.WriteLine("El float leido es un {0}",
                unDatoFloat);

            unDatoDouble = ficheroEntrada.ReadDouble();
            Console.WriteLine("El double leido es un {0}",
                unDatoDouble);

            unDatoString = ficheroEntrada.ReadString();
            Console.WriteLine("El string leido es \"{0}\"",
                unDatoString);
            Console.WriteLine("Volvamos a leer el int...");

            ficheroEntrada.BaseStream.Seek(1, SeekOrigin.Begin);
                unDatoInt = ficheroEntrada.ReadInt32();
            Console.WriteLine("El int leido es un {0}",
                unDatoInt);
            ficheroEntrada.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("Problemas al leer: "+e.Message);
            return;
        }
    }
}
```

El resultado de este programa es:

```
Introduzca el nombre del fichero a crear: 1234
Creando fichero...
Leyendo fichero...
El byte leido es un 1
El int leido es un 2
El float leido es un 3
El double leido es un 4
El string leido es "Hola"
Volvamos a leer el int...
El int leido es un 2
```

En este caso hemos usado "FileMode.Create" para indicar que queremos crear el fichero, en vez de abrir un fichero ya existente. Los modos de fichero que podemos emplear en un BinaryReader o en un BinaryWriter son los siguientes:

- CreateNew: Crear un archivo nuevo. Si existe, se produce una excepción IOException.
- Create: Crear un archivo nuevo. Si ya existe, se sobrescribirá.
- Open: Abrir un archivo existente. Si el archivo no existe, se produce una excepción System.IO.FileNotFoundException.
- OpenOrCreate: Se debe abrir un archivo si ya existe; en caso contrario, debe crearse uno nuevo.

- Truncate: Abrir un archivo existente y truncarlo para que su tamaño sea de cero bytes.
- Append: Abre el archivo si existe y se desplaza hasta el final del mismo (para añadir a continuación), o crea un archivo nuevo si no existe.

Ejercicios propuestos:

(8.17.1) Crea una copia de un fichero EXE. La copia debe tener el mismo nombre y extensión BAK.

(8.17.2) Crea un programa que "encripte" el contenido de un fichero BMP, volcando su contenido a un nuevo fichero, en el que intercambiará los dos primeros bytes. Para desencriptar, bastará con volver a intercambiar esos dos bytes, volcando a un tercer fichero.

8.18. Leer y escribir en un mismo fichero binario

También es posible que nos interese leer y escribir en un mismo fichero (por ejemplo, para poder modificar algún dato erróneo, o para encriptar un fichero sin necesidad de crear una copia, o para poder crear utilidades más avanzadas como un editor hexadecimal). Podemos conseguirlo abriendo (en modo de lectura o de escritura) y cerrando el fichero cada vez, pero también tenemos la alternativa de usar un "FileStream", que también tiene un método llamado simplemente "**Open**", al que se le puede indicar el modo de apertura (FileMode, como se vio en el apartado 8.11) y el **modo de acceso** (FileAccess.Read si queremos leer, FileAccess.Write si queremos escribir, o FileAccess.ReadWrite si queremos leer y escribir).

Una vez que hayamos indicado que queremos leer y escribir del fichero, podremos movernos dentro de él con "Seek", leer datos con "Read" o "ReadByte", y grabar datos con "Write" o "WriteByte":

```
// Ejemplo_08_18a.cs
// Lectura y escritura en fichero binario
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO;

class Ejemplo_08_18a
{
  const int TAMANYO_BUFFER = 10;

  static void Main()
  {
    FileStream fichero;
    string nombre;
    byte[] datos;
```

```
    nombre = "datos.dat";
    datos = new byte[TAMANYO_BUFFER];

    // Damos valores iniciales al array
    for (byte i=0; i<TAMANYO_BUFFER; i++)
      datos[i] = (byte) (i + 10);

    try
    {
      int posicion = 0;

      // Primero creamos el fichero, con algún dato
      fichero = File.Create( nombre );
      fichero.Write(datos, posicion, TAMANYO_BUFFER);
      fichero.Close();

      // Ahora leemos dos datos
      fichero = File.Open(nombre, FileMode.Open, FileAccess.ReadWrite);

      fichero.Seek(2, SeekOrigin.Begin);
      int nuevoDato = fichero.ReadByte();
      Console.WriteLine("El tercer byte es un {0}", nuevoDato);

      // Y cambiamos un dato sin cerrar el fichero
      fichero.Seek(2, SeekOrigin.Begin);
      fichero.WriteByte( 4 );

      fichero.Seek(2, SeekOrigin.Begin);
      nuevoDato = fichero.ReadByte();
      Console.WriteLine("Ahora el tercer byte es un {0}", nuevoDato);

      fichero.Close();
    }
    catch (Exception e)
    {
      Console.WriteLine(e.Message);
      return;
    }
  }
}
```

Ejercicios propuestos:

(8.18.1) Crea un programa que "encripte" el contenido de un fichero BMP, intercambiando los dos primeros bytes (y modificando sobre el mismo fichero). Para desencriptar, bastará con volver a intercambiar esos dos bytes.

8.19. Ejercicios de repaso propuestos sobre ficheros

(8.19.1) Crea un programa que vuelque todo el contenido de un fichero de texto a otro, convirtiendo cada frase a mayúsculas. Los nombres de ambos ficheros se deben indican como parámetros en la línea de comandos.

(8.19.2) Haz un programa que vuelque todo el contenido de un fichero binario a otro, convirtiendo cada letra entre la "a" y la "z" a mayúsculas. Los nombres de

ambos ficheros se deben indicar como parámetros en la línea de comandos. Si no hay parámetros, se le preguntarán al usuario.

(8.19.3) Mejora la base de datos de personas (ejercicio 4.6.3) para que los datos almacenados se guarden automáticamente en fichero al terminar una ejecución, y se vuelvan a cargar al principio de la siguiente.

(8.19.4) Crea un "traductor básico de C# a C", que volcará todo el contenido de un fichero de texto a otro, pero reemplazando "Console.WriteLine" con "printf", "Main" con "main", "string" con "char[80]", "Console.ReadLine" con "scanf", y eliminando "static" y "public" y las líneas que comiencen con "using".

(8.19.5) Prepara un programa que pida al usuario el nombre de un fichero y una secuencia de 4 bytes, y diga si el fichero contiene esa secuencia de bytes.

(8.19.6) Haz un programa que duplique un fichero, copiando todo su contenido a un nuevo fichero, byte a byte. El nombre del fichero de salida se tomará del nombre del fichero de entrada, al que se añadirá ".out".

(8.19.7) Crea un programa que duplique un fichero, copiando todo su contenido a un nuevo fichero, volcando para ello todo el contenido en un array. El nombre de ambos ficheros se debe leer de la línea de comandos, o pedir al usuario en caso de no existir parámetros.

(8.19.8) Prepara un programa que compare si dos ficheros son iguales byte a byte, comprobando primero su tamaño y leyendo después cada vez un byte de cada uno de ellos.

(8.19.9) Crea un programa que compare si dos ficheros son iguales, volcando todo su contenido en sendos arrays, que compararás.

(8.19.10) Implementa un programa que muestre el nombre del autor de un fichero de música en formato MP3 (tendrás que localizar en Internet la información sobre dicho formato y sobre la cabecera ID3 V1, que se encuentra -si está presente- en los últimos 128 bytes del fichero).