Ejercicios Adicionales de Ficheros

Ficheros de Texto

1. Escribe la función **CuentaVocalesFichero** a la que le pasamos una cadena con el nombre de un fichero de texto y nos cuenta cuántas veces aparece cada vocal. La función nos devolverá un array de 5 enteros con las veces que se repite cada una de las 5 vocales. Deberá contar todas las vocales, incluyendo minúsculas, mayúsculas y las que estén acentuadas.

   Como ejemplo os paso el "Lazarillo de Tormes" en el fichero lazarillo.txt. Al probarlo con la función, os debería devolver los siguientes valores:

   ```
   [10797, 11205, 4606, 8129, 3749]
   ```

2. Escribe la función **EscribeSumasCSV** a la que le pasamos una cadena con el nombre de un fichero CSV que contiene una tabla con notas de alumnos y copia esa tabla en el otro fichero que le pasamos por parámetro completando los datos que faltan. El formato de cada línea del fichero CSV será el siguiente:
   - En la primera columna, el nombre del alumno.
   - En las tres siguientes columnas, las notas del primer, segundo y tercer trimestre.
   - La quinta columna no debe existir en el primer fichero, y deberá contener la nota media de los tres trimestres en el segundo fichero, redondeada a 2 decimales.
   - La sexta columna también falta del primer fichero y tendrá una cadena que podrá ser "Aprobado" o "Suspenso" dependiendo de si la nota media es mayor o igual que 5 o no.

   Se debe comprobar que el primer fichero existe, que el segundo no existe y, opcionalmente, que cada línea del fichero que leemos tiene 4 campos. Si no se cumplen estas condiciones, se deberá dar un error y no crear el fichero nuevo.

3. Escribe la función **LeeSopaLetras**, que nos leerá una sopa de letras de un archivo de texto cuyo nombre le pasamos por parámetro y nos devolverá un array bidimensional de caracteres con la sopa de letras.

   En el fichero de texto aparecerá cada línea sin separación entre las letras.

   Ej.: PATAK
   IKPMO
   CJDME
   OLOTS

   En este ejemplo, la función nos devolvería un array de tipo char[4][5].

La función deberá comprobar si todas las líneas son iguales. En caso contrario, nos devolverá un array vacío (char[0][0]).

Se recomienda adaptar la función EscribeArrayBidimensional para poder ver si se ha leído correctamente la sopa de letras.

4. Escribe la función **CuatroVocales** a la que le pasamos un nombre de fichero de texto y nos busca las palabras que haya en ese fichero con al menos cuatro vocales. La función nos creará otro fichero que se llamará "4vocales.txt" donde aparecerán las palabras que cumplan esa condición (una palabra en cada línea).
   En el fichero original podrán aparecer símbolos de puntuación, así que hay que tener cuidado con ellos. Como ejemplo, os incluyo el primer capítulo de "El Lazarillo de Tormes".

5. Escribe dos funciones **EscribeArrayBi** y **LeeArrayBi**. A la función EscribeArrayBi le pasaremos el nombre del fichero y un array bidimensional de enteros y nos guardará ese array en un fichero de texto con la siguiente estructura:
   - Primero escribiremos dos enteros correspondientes a la primera dimensión (filas) y a la segunda dimensión (columnas), en una línea separados por una coma.
   - Después escribiremos todos los valores enteros del array, cada uno en una línea.

   A la función LeeArrayBi le pasaremos un nombre de fichero del que leerá un array bidimensional del tamaño que aparece en la primera línea del fichero y lo devolverá.

   Ej.: Si el array es {{1,2,3},{4,5,6}}, en el fichero guardaremos primero 2,3 que son las dos dimensiones y luego en 1, 2, 3, 4, 5 y 6 cada uno en una línea.

6. Escribe la función **PalabrasDistintas** a la que le pasas un nombre de fichero, que será un fichero de texto, y te devuelve un entero con el número de palabras distintas que contiene el fichero. Por ejemplo, si aparece seis veces la palabra "el", sólo habrá que contarla una vez.

   Para que no se escape ninguna palabra, no se os olvide ponerlo todo en minúscula y eliminar todos los signos de puntuación (se recomienda usar la función char.IsLetter). Como ejemplo, os paso un capítulo del Lazarillo de Tormes en el fichero "lazarillo.txt" (que contiene 2139 palabras distintas) y otro del Quijote en el fichero "quijote.txt" (que contiene 718 palabras distintas).

7. Escribe un programa que lea el fichero "dias_de_lluvia.csv" que contiene datos de los días de lluvia mensuales en cada capital española y nos muestre la capital más lluviosa y la menos lluviosa (calculándolo dentro del programa, evidentemente).

   Los datos están sacados de la siguiente tabla Excel:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Ciudad**	**Ene**	**Feb**	**Mar**	**Abr**	**May**	**Jun**	**Jul**	**Ago**	**Sep**	**Oct**	**Nov**	**Dic**
2	Albacete	4	4	5	6	7	4	1	2	3	5	5	5
3	Alicante	4	3	4	4	4	2	1	1	3	4	4	4
4	Almeria	3	3	3	3	2	1	0	0	1	3	3	3
5	Avila	6	5	4	8	9	5	2	2	4	6	6	7
6	Badajoz	7	6	5	7	6	3	1	1	3	7	7	8

8. Escribe una función **ProcesaLogs** a la que le pasaremos un fichero de logs (en este caso os paso un fichero con los logs del servidor web de la NASA de julio de 1995). La función deberá escribirnos por pantalla un listado de las 20 URLs que más hayan accedido al servidor (o sea, las que aparezcan en más líneas), junto con el número de accesos de cada una. Sólo deberán aparecer URLs, debiéndose descartar las líneas en las que aparezcan direcciones IP.
   Para el fichero que os paso, las tres URLs con más accesos son tres subdominios de prodigy.com y tienen 17615, 11581 y 9892 accesos respectivamente. La vigésima URL con más accesos es un proxy de AOL con 3669.

9. Escribe un programa que nos traduzca un fichero de texto a Leet Speak. El programa nos preguntará el nombre del fichero de texto original y el nombre del fichero nuevo y copiará de uno a otro realizando los cambios necesarios.

   Reglas de 1337 5p34k:

   - Las siguientes letras se sustituyen por otro carácter:

A	B	E	G	I	L	O	S	T	Z
4	8	3	6	!	1	0	5	7	2

   - Para el resto de letras, hay un 50% por ciento de posibilidades de que la letra pase a estar en mayúsculas (o si ya estaba en mayúsculas, que pase a estar en minúsculas). Si no, la dejamos como está.

10. Escribe un programa que lea un fichero de texto que le indicamos por teclado y que nos muestre por pantalla todas las palabras de ese fichero que empiecen por mayúscula. Si la palabra lleva pegada algún carácter de puntuación (por ejemplo, una interrogación: "¿Ahora …" o un punto "… fui a ver a María."), quitárselo. También nos mostrará al final una estadística con el número de palabras totales, el número de palabras que empiezan por mayúscula y el porcentaje que representan del total.

11. Escribe el programa **PalabraMasRepetida**, al que le diremos un fichero de texto (se adjunta El Quijote como ejemplo) y nos sacará una lista de las 10 palabras que más se repiten, ordenadas de mayor a menor.
    Como ejemplo, en El Quijote que os he pasado la palabra que más se repite es "que" (sin tilde) con 20628 repeticiones y la décima más repetida es "se" (también sin tilde) con 4691 (nota: estos valores pueden cambiar sin previo aviso).

Notas:

- El Quijote es bastante largo, así que usad sólo un trozo para las pruebas.
- En el fichero que os he pasado (y en cualquier otro en general) las palabras pueden estar separadas por espacios, pero también por un salto de línea, un punto, una coma o un punto y coma.
- La palabra se contará independientemente de que esté en mayúsculas o en minúsculas.
- Si usáis un *Split* para dividir las palabras, muchas de ellas se llevarán de regalo algún símbolo de puntuación. Debéis limpiar cada palabra convenientemente antes de contarla.
- Como ya os podéis suponer por el ejemplo, una palabra con tilde y otra sin ella se cuentan como palabras distintas, como por ejemplo “como” y “cómo” (no sé qué opinarán los lingüistas de esto, pero me la trae al pairo).

12. Escribe un programa de consola que se llamará **SuperEncriptaciónChina**. El programa pedirá por teclado el nombre de un fichero de texto, que leeremos. Para codificarlo, haremos lo siguiente para cada línea:
    - La última letra de cada palabra la intercambiaremos con la primera letra de la siguiente palabra.
    - La última letra de toda la línea la intercambiaremos con la primera letra de la línea.

    Ej.: el cielo es muy bonito -> oc liele om sub yonite

    Una vez codificada cada línea, la guardaremos en un fichero que tendrá el mismo nombre que el fichero original, pero añadiéndole “_codificado” al final del nombre (pero antes de la extensión). P.ej.: texto.txt -> texto_codificado.txt.

    Para que la codificación os quede bien, os aconsejo que lo probéis con un fichero que tenga varias líneas, con todo el texto en minúsculas y sin signos de puntuación. También podéis probar a pasarle un texto codificado y os tendría que devolver el texto original.

Ficheros Binarios

1. Escribe la función **ImprimeJornada** que nos leerá de un fichero **binario** una jornada de la liga de fútbol (por ejemplo) y nos la pondrá por pantalla. A esta función le pasaremos el nombre de un fichero (p.ej.: jornada3) y nos leerá del fichero los datos de los 10 partidos de la jornada. En el fichero estarán los datos de cada partido de manera consecutiva con este formato:
   - Nombre del equipo local (*string*)
   - Nombre del equipo visitante (*string*)
   - Goles del equipo local (*int*)
   - Goles del equipo visitante (*int*)
   - y luego el siguiente partido, etc.

   El resultado deberá quedar tal que así:

```
      Málaga  2-1  Eibar
     Sevilla  1-0  Betis
       Celta  2-1  Sporting
Real Madrid   1-1  Villarreal
              ...
```

   Con los resultados alineados en el centro y los nombres de los equipos a izquierda y derecha de los mismos.

2. Escribe la función **LeeColorinesConsola** que te pinta un cuadrado de 10x10 usando diferentes colores que leerá desde un fichero. El fichero será un fichero binario en el que hay 100 enteros. Cada vez que leemos un entero, cambiaremos el color de fondo de la consola con la siguiente orden:

   Console.BackgroundColor = (ConsoleColor)entero;

   y escribiremos un espacio que aparecerá de ese color.

3. Escribe una función **LeeJuegoEnRaya** que leerá unos ficheros binarios especiales en los que se encuentra una posición del tablero del juego de la 3 en raya (o las 4 en raya, o las 5, etc.). A la función le pasaremos por parámetro el nombre del fichero y nos mostrará por pantalla una representación gráfica del tablero.
   El fichero tendrá extensión .ttt y tendrá la siguiente estructura:
   - En primer lugar, habrá un entero que nos indicará el tamaño del tablero. Por ejemplo, si es un 3, estaremos hablando de un tablero de 3x3.
   - A continuación, aparecerán las posiciones del tablero, teniendo en cuenta que un 0 representa una casilla vacía, un 1 una casilla marcada con una X y un 2 una casilla marcada con una O.

   Ej.: Si en el fichero están los números: 3, 0, 0, 1, 1, 2, 2, 2, 1, 1, el programa nos tendrá que pintar lo siguiente:

```
·  ·  X
X  O  O
O  X  X
```

4. Escribe un programa que nos lea un fichero binario que contiene “ASCII Art” y lo muestre por pantalla. El contenido del fichero será:
    - int
    - int
    - char
    - (repetimos)

    El primer entero será la posición “x” y el segundo será la posición “y” del carácter “c”. Lo que tendréis que hacer es poner el cursor de la consola en la posición que leáis del fichero (con Console.SetCursorPosition) y escribir el carácter en esa posición.

    Se recomienda limpiar la consola antes de ponerse a pintar el dibujito. Incluyo un programita para crear los ficheros binarios a partir de un fichero de texto.

5. Escribe la función **PintaBandera**, que pintará una bandera por pantalla a partir de un fichero binario con extensión “flag” que tendrá las siguientes características:
    - En la primera posición del fichero hay un entero correspondiente al número de filas.
    - En la segunda, otro entero correspondiente al número de columnas.
    - A continuación, habrá tantos enteros como posiciones haya en la bandera (una bandera de 3x5, tendrá 15 posiciones, 5 en cada fila).
    - El color de cada posición dependerá de ese entero, según la siguiente tabla: 1: Rojo, 2: Amarillo, 3: Azul, 4: Verde, 5: Blanco, 6: Negro.

    Para pintar los diferentes colores en la consola, usad: *Console.BackgroundColor*.

6. Escribe un programa **EstadisticasNBA** que nos sacará por pantalla un listado de los mejores anotadores de la NBA sacando los datos de unos ficheros binarios.

    Al ejecutar el programa, en el directorio de trabajo habrá unos ficheros con extensión *stat* que yo os proporciono. Cada fichero tiene como nombre el nombre del jugador en cuestión y dentro hay una serie de números enteros correspondientes a los puntos anotados en los últimos partidos por ese jugador.

    Para cada jugador tendremos que leer su fichero y calcular la media de anotación de sus partidos (p.ej.: si hay 5 enteros: 20, 25, 15, 10, 30 = 100/5 partidos = 20 puntos/partido). El programa nos mostrará por pantalla un listado de todos los jugadores con sus puntos por partido, ordenados de mayor a menor (si no conseguís que salgan ordenados, al menos que salgan).

    Para leer todos los ficheros que haya en el directorio, usad *Directory.GetFiles* (y probad que funciona si quitáis o añadís ficheros).

    Con los ficheros que os he dado, el listado debería ser:

```
Russell Westbrook           33,25
James Harden                30,60
LeBron James                29,38
Stephen Curry               28,00
Anthony Davis               27,54
```

```
Kawhi Leonard                 27,25
John Wall                     27,00
Isaiah Thomas                 26,74
Marc Gasol                    17,40
Pau Gasol                     12,00
José Manuel Calderón           6,60
```

7. Tenemos una serie de ficheros que contienen datos de transacciones bancarias creados con un programa antiguo que guardaba los datos en ficheros de texto con el siguiente formato:

```
+------------------------+------------------------+----------+-----------------+
| Cuenta origen          | Cuenta destino         | Fecha    | Cantidad        |
+------------------------+------------------------+----------+-----------------+
| 2015-1223-15-0000015269 | 1928-0649-71-0004876213 | 30/05/83 |      32.400.000 |
| 4598-0034-72-0080008478 | 1298-0049-82-0000008743 | 13/06/83 |       1.567.200 |
| 1892-0659-12-0000073641 | 3549-0635-41-2000098788 | 16/06/83 |       2.500.000 |
| 8734-0877-26-0070006487 | 1231-0098-64-5000065465 | 19/06/83 |         725.500 |
| 0192-0909-02-3000009138 | 2235-6049-74-0005465054 | 11/07/83 |       2.125.225 |
| 1827-0062-74-0000468125 | 9182-0058-12-3000007468 | 12/08/83 |     100.000.000 |
| 1928-0044-17-0000000366 | 1235-9072-39-0000003876 | 12/09/83 |       3.501.540 |
| 1928-0629-72-0000002382 | 0129-0049-28-0000073490 | 22/09/83 |         859.120 |
+------------------------+------------------------+----------+-----------------+
```

Se pide escribir un programa que se llamará “NuevoFormatoBanco” que te pide el nombre de un fichero que se encuentra en este formato y te crea un nuevo fichero en formato binario en el que se almacenen los mismos datos. El formato de este fichero será el siguiente:

- Como es un fichero binario, todos los registros irán uno después del otro sin ninguna separación. Cada registro se compondrá de:
    - Una cadena para la cuenta de origen y otra cadena para la cuenta de destino. Eliminaremos los guiones que hay en medio de los números de cuenta y dejaremos únicamente los 20 dígitos.
    - Tres enteros para la fecha: primero escribiremos el día, luego el mes y luego el año. Completar las dos cifras del año para conseguir un número de cuatro cifras (suponed que todas las fechas se refieren al siglo pasado).
    - Por último, un decimal para la cantidad, que tendremos que convertir de pesetas (tal y como vienen en el fichero) a euros.

Como al escribir en un fichero binario es difícil comprobar si lo estáis haciendo bien (y digo “estáis” porque yo entiendo el hexadecimal perfectamente :P), os doy las siguientes indicaciones:

- Para comprobar si lo que vais a escribir está bien, podéis empezar mostrando por pantalla los datos que escribiríais en el fichero binario. Si todo está correcto, ya podéis pasar a escribirlos en el fichero.
- También os voy a pasar dos ficheros. El fichero “datosantiguos.txt” contiene los mismos datos del ejemplo, mientras que el fichero “datosnuevos.bin” contiene lo que debería devolver vuestro programa (si lo estáis haciendo bien, os debería salir exactamente el mismo fichero, pero si os da otra cosa, no significa que esté completamente mal, sólo que no está bien :D).
- Ojo, el programa tiene que funcionar para cualquier fichero que tenga este formato, independientemente del número de líneas que tenga (las líneas y los campos sí que tienen que tener todos la misma longitud). No me vale que me lo hagáis para

exactamente 8 líneas, como en el ejemplo. Comprobad, modificando el fichero, que funciona también para menos y más líneas.

8. Escribe el programa **SuperCodificadorPlus**. El programa nos preguntará un par de nombres de fichero, leerá el contenido del fichero1, lo codificará y lo escribirá en el fichero2. El método de codificación será el siguiente: el fichero lo leeremos como un fichero binario (independientemente de si el fichero es de binario o de texto), e iremos leyendo los caracteres como valores numéricos de tipo *Byte*. Para codificar cada valor, si ese número es menos o igual que 127, le sumaremos 127, mientras que, si es mayor o igual que 128, le restaremos 127. El valor codificado lo copiaremos en el segundo fichero.

   Os proporciono los ficheros "Quijote.txt" y "Quijote Codificado.txt" para que comprobéis si funciona correctamente.

   NOTA: Para poder sumarle cosas a un *byte*, hay que escribirlo de la siguiente forma:
   ```
   byte z = (byte)(x + y); // no preguntéis por qué
   ```

Ficheros y Directorios

1. Escribe el programa **MP3Shuffle**, que cogerá todos los ficheros MP3 que haya en el directorio actual y los ordenará aleatoriamente para poder reproducirlos en un reproductor muy chungo que no sepa reproducir canciones aleatoriamente (basado en hechos reales).

   El programa cogerá todos los ficheros con extensión MP3, los meterá en la lista y los desordenará. A continuación, renombrará cada archivo (usando *File.Move()*) y les pondrá un número delante con la posición en la lista de reproducción. Si el archivo ya tenía un número delante, se lo quitará (para permitir reordenar varias veces).

   Ej: los ficheros: "Let It Be.mp3", "Imagine.mp3", "Hold the Line.mp3" y "Stop.mp3" podrían quedar como: "01. Hold the Line.mp3", "02. Let It Be.mp3", "03. Stop.mp3" y "04. Imagine.mp3" (o en otro orden cualquiera).

2. Escribe el programa **OrdenaFicherosPorLineas** que, al ejecutarlo, nos sacará un listado de todos los ficheros de texto que hay en el directorio actual junto con el número de líneas que contiene cada uno. El listado aparecerá ordenado de mayor a menor número de espacios.
   Para conseguir un listado de los ficheros .txt que hay en el directorio actual, podéis usar esta función que se encuentra dentro de System.IO:
   ```
   Directory.GetFiles(".", "*.txt")
   ```
   Esta función nos devolverá un array de cadenas en el que aparecerán todos los ficheros con extensión txt del directorio actual.
   Se podrán crear las funciones necesarias para simplificar la tarea a realizar.

El programa debería escribir por pantalla un listado similar a éste:

```
uno.txt      25
dos.txt      17
siete.txt    5
cinco.txt    5
```

3. Escribe el programa **AutoBackup**. El programa hará lo siguiente: comprobará si en el directorio actual se encuentra el archivo "survival_1".
   - Si lo encuentra, hará una copia de seguridad de ese archivo en un archivo que se llamará "backup_0001". Si ya existe el archivo "backup_0001", la copia se hará en el archivo "backup_0002", y así sucesivamente.
   - Si no lo encuentra, buscará el archivo de *backup* más reciente (el que tiene el número más grande) y restaurará esa copia con el nombre del archivo original.
   - Si no se encuentra ni el archivo "survival_1" ni ningún "backup", el programa escribirá un mensaje de error (no hace falta lanzar una excepción).

   En cualquiera de los tres casos, el programa explicará si se ha encontrado o no el fichero y qué acción se ha realizado.

   NOTA: ¿No he explicado cómo se copian ficheros? Es con *File.Copy()*.

   NOTA2: Como alguno seguro que me lo pregunta: ¿qué pasaría si ya existe el fichero "backup_9999"? Pues que se borra y se machaca con la nueva copia de seguridad.

   NOTA3: Una mejor manera de comprobar cuántos ficheros de *backup* existen que comprobándolos uno a uno es usar la función *Directory.GetFiles()* que te devuelve un *array* de cadenas con los nombres de todos los ficheros que hay en el directorio actual. Si queréis la usáis, si no, no.