





Ejercicios Flujos de Entrada y Salida

[Descargar estos ejercicios](#)

Índice

- [Ejercicio 1](#)
- [Ejercicio 2](#)
- [Ejercicio 3](#)
- [Ejercicio 4](#)
-  [Ejercicio 5](#)
-  [Ejercicio 6](#)
- [Ejercicio 7](#)
-  [Ejercicio 8](#)
-  [Ejercicio 9](#)
- [Ejercicio 10](#)

Ejercicio 1

Realiza un programa que pida la ruta con el nombre del fichero incluido. El programa creará el fichero, le introducirá la información que desee el usuario y posteriormente lo abrirá para su lectura y visualización por pantalla.

📌 **Nota:** La lectura se deberá realizar **byte a byte** usando **únicamente `FileStreams`**. Se tendrá que tener en cuenta que el fichero de origen exista y además si la ruta del fichero de destino no existe, se tendrá que crear.

Ejercicio 2

Realiza un programa llamado **mycp** que funcione parecido al comando **cp** del **Linux**. El programa hará la copia de un archivo origen en otro destino. Introduciendo ambos como argumentos de la línea de comando.

📌 **Nota:** La operación de copiar se deberá realizar **byte a byte** usando **únicamente `FileStreams`**.

Ejercicio 3

Vamos a ampliar el **mycp** del ejercicio anterior. Pero en este caso, en lugar de copiar byte a byte se hará mediante un **`BufferedStream`**, con una capacidad de **100000** bytes que utilizaremos para ir copiando de **100 KB en 100 KB**.

Ejercicio 4

Dado un fichero de **texto** con codificación **UTF8**, escribe un programa que convierta los caracteres alfabéticos que aparecen en mayúscula por caracteres alfabéticos en minúscula y viceversa.

Deberás controlar las entradas correctas, como por ejemplo controlar que exista el archivo o controla la excepción que se lanza cuando se intenta modificar un archivo y el acceso no está autorizado.

📌 **Nota:** El cambio deberá realizarse **sobre el mismo stream** usando el método **`Seek`** de la clase **`FileStream`**. Para probar la excepción de modificación, pon el atributo del archivo a solo lectura.

✓ Ejercicio 5

Escribe un programa que cree un fichero de texto de nombre **datos.txt** , que se encuentre en la carpeta **datos** del directorio raíz de la unidad **C:**.

El programa deberá comprobar si la carpeta existe y si no es así la creará.

En ese fichero iremos guardando **carácter a carácter** lo que se introduzca por teclado usando un adaptador **BinaryWriter** .

Finalizaremos la introducción de caracteres al pulsar la tecla **ESC** .

✓ Ejercicio 6

Crea una clase **Microprocesador** con tres atributos privados **modelo, nucleos y frecuencia**. La clase tendrá el constructor necesario para crear un microprocesador con todos los datos y la anulación del ToString para mostrar la instancia de la forma:

Modelo: Intel Core I7

Nucleos: 4

Frecuencia: 3.6

En la clase también crearemos un método **ACSV** que devolverá un string con los datos del microprocesador separados por **;** , para la instancia anterior la salida sería:

Intel Core I7;4;3.6

Otro de los métodos será **AMicroprocesador**, este método será de clase, le llegará una cadena con el formato anterior y se encargará de sacar la información para crear un microprocesador (puedes usar el método Split de cadena). Este método retornará el Microprocesador creado.

Por otro lado, en la clase del programa principal tendremos los métodos necesarios que permitan:

- Devolver un array de microprocesadores con los datos recogidos por teclado.
- Guardar en un fichero llamado **microprocesadores.csv** cada uno de los microprocesadores en líneas distintas y con sus datos separados por **;** , para ello usaremos el método de la clase Microprocesador **ACSV**. Este método guardará un microprocesador cada vez, por lo que tendrá que abrir el fichero para añadir al final.
- Leer todo el fichero **microprocesadores.csv** y devolverá un array con los microprocesadores leídos, para ello usará el método **AMicroprocesador** de la clase Microprocesador.

Realiza un programa principal que permita recoger, guardar y leer de fichero y mostrar los microprocesadores leídos.

Ejercicio 7

Realiza un programa que cuente el número de caracteres de texto **unicode** de un fichero. Para hacer la lectura del fichero debes usar un adaptador **StreamReader**.

El nombre del fichero será pasado como argumento en la **línea de comandos**.

📌 **Nota:** El número de caracteres no tiene porque coincidir con el número de bytes.

✅ Ejercicio 8

Programa que permita **buscar una palabra** en uno o más ficheros de texto (introducidos en la línea de comandos).

Se necesitará un método **BuscaEnFichero(string ruta, string palabra)**, que extraerá las **líneas** del fichero y llamará a la función **BuscaEnCadena(string cadena, string palabra)** que comprobará si las cadenas son iguales.

En la línea de comandos introducirás la palabra y los nombres de fichero a buscar y te mostrará un mensaje para cada fichero, en el que te indicará si ha sido encontrada en ese fichero.

✅ Ejercicio 9

Programa que muestre **el/los número/os de línea/as** que contengan una subcadena pedida al usuario en un fichero indicado.

Además del número de línea, nos indicará el **número de apariciones** de la subcadena en dicha línea.

Si no encuentra la subcadena en todo el fichero, nos mostrará un mensaje de "**CADENA NO ENCONTRADA**".

Para hacer este programa crearemos una **expresión regular** a partir de la cadena que nosotros indiquemos, que será la que se busque con las líneas del fichero.

Ejercicio 10

Un diptongo está formado por dos vocales, una fuerte y una débil, o dos débiles. Las vocales fuertes son **a, e, o** ; las vocales débiles son **i, u** . La acentuación de **u** o **i** destruye el diptongo.

Crea un programa que, a partir de un archivo que nosotros indiquemos desde teclado y usando expresiones regulares, nos permita:

1. Mostrar todas las palabras con diptongo formado por dos vocales débiles, **ordenadas** y **sin repetir**.
2. Mostrar del mismo modo todos los diptongos con **a** .
3. Buscar una **expresión regular mínima** que muestre **todos los diptongos**.