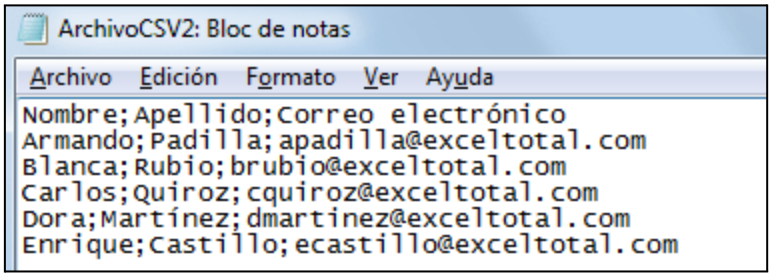
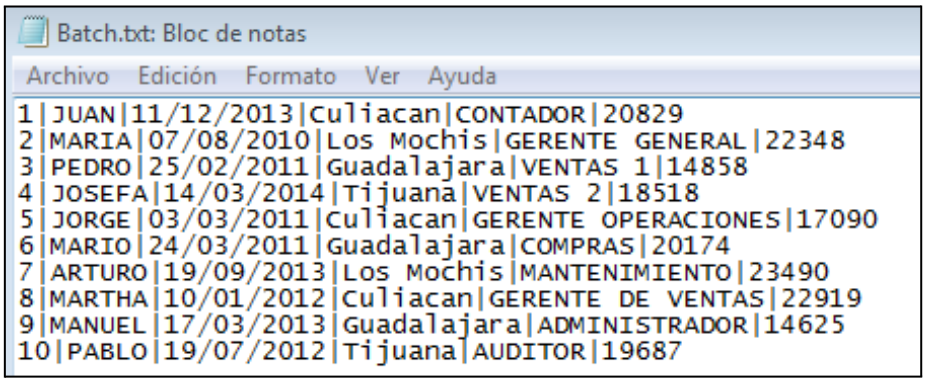
Pre-procesamiento de datos

PRÁCTICA FUNDAMENTOS DE PROGRAMACIÓN

Un fichero CSV (del inglés *'comma separated values'*) es un fichero de texto plano que contiene datos separados por comas. Por extensión, también se puede llamar CSV a los ficheros de texto con datos estructurados en forma de tabla que usan de separador cualquier otro carácter (espacio en blanco, tabulador, punto y coma, etc...), o conjunto de caracteres (por ejemplo, la doble almohadilla '**##**', el doble pipe '**||**' u otros). Un fichero CSV puede tener como primera línea (aunque no siempre) una cabecera (o header) para indicar el nombre de cada una de las columnas de datos del fichero. Ejemplos:



Fichero CSV con separador ';' y cabecera: **Nombre / Apellido / Correo**



Fichero CSV con separador '|' y sin cabecera

En esta práctica se van a considerar solamente ficheros CSV con cabecera, y cuyos datos, a partir de la segunda fila, serán de alguno de los siguientes tipos:

- NUM: datos numéricos, tanto enteros como reales
- DMY: fechas en formato DÍA/MES/AÑO (DD/MM/YYYY)
- YMD: fechas en formato AÑO/MES/DÍA (YYYY/MM/DD)
- TIME: hora y minuto (HH:MM)
- STR: cadena de caracteres (abreviatura de '*string*')
- VOID: tipo vacío, se aplicará cuando haya un valor inexistente

La cabecera indicará el número de columnas de datos que tiene el fichero, si alguna de las filas siguientes a la cabecera no tuviera la misma cantidad de valores se considerará

una fila errónea. Por otra parte, la primera fila de datos (la segunda después de la cabecera) indicará de qué tipo es cada una de las columnas del fichero. También se considerarán filas erróneas aquellas que no tengan los mismos tipos de datos, y en el mismo orden, que la primera fila.

El programa a realizar debe ser capaz de leer un fichero CSV (con cabecera), que ha de contener un conjunto de datos estructurados en forma de tabla, cargar dichos datos en memoria, y realizar diferentes tipos de alteraciones o modificaciones sobre dichos datos. El programa deberá responder a diversos tipos de órdenes (comandos) que el usuario introducirá por consola, mediante los cuales se indicará al programa la operación que debe realizar.

El programa deberá mostrar un "***prompt***" que indique el nombre del fichero CSV que se ha cargado en memoria para trabajar con él, por ejemplo, si se ha cargado un fichero de datos llamado "MisDatos.txt" el prompt deberá mostrar un aspecto parecido a este:

`MisDatos.txt:>>`

Cuando el programa está recién arrancado, y aún no hay un fichero de datos cargado, se indicará poniendo un asterisco en vez del nombre del fichero:

`*:>>`

A continuación se van a indicar los comandos que el programa deberá ser capaz de reconocer y ejecutar. El texto inscrito entre los caracteres 'menor' y 'mayor' (**< . . . >**) se refiere a un argumento o parámetro que será necesario incluir en el comando correspondiente. Si a su vez un argumento aparece inscrito entre corchetes (**[< . . . >]**), eso querrá decir que dicho parámetro es opcional.

Los comando que debe ejecutar el programa son los siguientes:

`exit`

Comando para finalizar el programa (no lleva parámetros), termina el programa liberando la memoria ocupada y sin mostrar por pantalla ningún tipo de mensaje.

`load <name_file> <sep>`

Carga en memoria el fichero de datos indicado por el parámetro "name_file", el cual deberá ser un CSV con datos delimitados por el separador "sep". Si existen datos en memoria de un fichero anterior, dichos datos se eliminan al cargar el nuevo fichero. Esta operación deberá reconocer también las filas erróneas del fichero CSV, mostrándolas en pantalla precedidas de su número, pero sin almacenarlas en memoria. Al terminar de cargar los datos se mostrará en pantalla el número de filas, de columnas y de líneas erróneas. También se modificará el prompt con el

nombre del nuevo CSV. En caso de error en el comando, o si el fichero indicado no se puede leer, se mostrará en pantalla un mensaje indicando el error ocurrido y no se modificará nada, es decir, el prompt seguirá como estaba antes del comando, y los datos que había en memoria previamente, seguirán en memoria.

`save [<name_file> [<sep>]]`

Guarda en un fichero en disco los datos que haya cargados en memoria en ese momento, los comandos “name_file” y “sep” son opcionales, si no se indica el separador (“sep”) el fichero se guardará con el mismo separador con que se leyó originalmente. Si se omite el nombre del fichero (“name_file”) los datos se guardarán con el mismo nombre de fichero que figura en el prompt (machacando el contenido que tuviera anteriormente dicho fichero). En caso de error en el comando, o si el fichero indicado no se pudiera abrir para escritura, se mostrará en pantalla un mensaje indicando el error ocurrido.

`sum <column1> <column2> <column3>`

Realiza la suma de los valores contenidos en las columnas de nombre “column1” y “column2” y crea una columna nueva, de nombre “column3”, para guardar el resultado de la suma. Obviamente las columnas “column1” y “column2” deben existir cargadas en memoria, del mismo modo, no debe existir ninguna columna con el nombre “column3”, de no cumplirse estas condiciones el comando no se podrá ejecutar. Por otro lado, dependiendo del tipo de datos de las columnas “column1” y “column2” el resultado será diferente:

column1	column2	column3 (resultado)
NUM	NUM	NUM
DMY	NUM	DMY (suma NUM como días)
NUM	DMY	DMY (suma NUM como días)
YMD	NUM	YMD (suma NUM como días)
NUM	YMD	YMD (suma NUM como días)
(resto de casos)		**ERROR**

En caso de error, el comando no ejecutará ninguna acción sobre los datos cargados en memoria, pero mostrará por pantalla un mensaje indicando el tipo de error que ha ocurrido.

`sub <column1> <column2> <column3>`

Comando análogo al anterior, pero en este caso realizando la resta (en inglés “subtraction” → sub) de los valores de la columna “column1” menos los de la columna “column2”. Se creará una nueva columna de nombre “column3” para guardar el resultado de la resta. Los resultados según los tipos de datos de “column1” y “column2” serán como sigue:

column1	column2	column3 (resultado)
NUM	NUM	NUM
DMY	NUM	DMY (resta NUM como días)
YMD	NUM	YMD (resta NUM como días)
(resto de casos)		**ERROR**

En caso de error, el comando no ejecutará ninguna acción sobre los datos cargados en memoria, pero mostrará por pantalla un mensaje indicando el tipo de error que ha ocurrido.

`disc <column> <lab1> <lab2> [<lab3>...] <val1> [<val2>...]`

Realiza la discretización (también llamada segmentación) de valores numéricos, la columna indicada en el primer parámetro “column” debe ser de tipo NUM, las etiquetas “lab1”, “lab2”, “lab3”, etc… deben ser de tipo STR y los valores “val1”, “val2”, etc… deben ser de tipo NUM. La cantidad de etiquetas (“lab”) y valores (“val”) es opcional, pero debe haber siempre una etiqueta más que valores, y en total, no deberá haber más de 7 etiquetas y 6 valores. Se creará una nueva columna con el mismo nombre que la columna indicada añadiéndole el sufijo “**_D**”. Los valores de la nueva columna serán de tipo STR y se distribuirán según el siguiente algoritmo en pseudocódigo:

```
SI column1[i] < val1
      column1_D[i] = lab1
SINO Si column1[i] < val2
      column1_D[i] = lab2
SINO Si . . .
      . . .
SINO SI column1[i] < valN
      column1_D[i] = labN
SINO
      column1_D[i] = lab1N+1
```

En caso de que ocurra alguna circunstancia por la que el comando no se pueda ejecutar, el programa mostrará un mensaje en pantalla indicando el tipo de error que haya ocurrido.

`rename <column1> <column2>`

Renombra la columna “column1” que pasará a llamarse “column2”. Para que el comando pueda ejecutarse tiene que existir una columna con el nombre “column1” y no debe existir ninguna columna llamada “column2”. Si la operación no se puede realizar se indicará con un mensaje de error.

`info <column>`

Proporciona información sobre la columna indicada se mostrará en pantalla el nombre y el tipo de dicha columna, y a continuación, según sea ese tipo se mostrará en pantalla la siguiente información:

- Si la columna es de tipo DMY, YMD o TIME, se indicarán los valores máximo y mínimo de la serie correspondiente.
- Si la columna es de tipo NUM, se muestran los valores mínimo, máximo y promedio de la serie, y se realiza el conteo de frecuencias para un *histograma* de cinco intervalos.
- Si la columna es de tipo STR se muestran los valores diferentes que contiene la serie seguido del número de veces que se repite cada valor.

Hasta aquí la descripción de los comandos que el programa debe poder interpretar y ejecutar. Nótese que, salvo los dos primeros comandos (“exit” y “load”) todos los demás requieren que previamente se haya cargado un fichero de datos en memoria, de no ser así, se estará incurriendo en un error que deberá dar como resultado un mensaje en pantalla indicando la falta de datos en memoria para poder ejecutar la acción. También son situaciones de error aquellas en las que un comando va seguido de una cantidad de parámetros incorrecta, distinta a la indicada en cada caso, o cuando los parámetros no son del tipo esperado (p.e.: texto en vez de un número). En estos casos el programa también deberá responder con un mensaje de error adecuado. Otras veces, el comando puede ser correcto, pero tal vez no se pueda ejecutar por diferentes motivos, por ejemplo, puede que el nombre del fichero indicado para cargar (“load”) no exista, o que el nombre indicado de una columna tampoco exista en el conjunto de datos cargados en memoria; en situaciones de este tipo, el comando tampoco se podrá ejecutar y el programa deberá indicarlo con un mensaje de error adecuado.

La estructura general del programa deberá seguir el siguiente esquema:

```
1. Imprimir en pantalla nombre, apellidos y e-mail del autor
2. Imprimir un ‘prompt’ y esperar a que el usuario escriba un
   comando por teclado (leer string con ‘gets()’)
3. Analizar el contenido de la cadena introducida
  3.1. Si se corresponde con un comando incorrecto, se indica
       con un mensaje de error
  3.2. Sino, si se corresponde con un comando correcto pero
       este no se pudiera ejecutar, se indica con el mensaje
       más apropiado posible
  3.3. Sino, se ejecuta el comando según las especificaciones
4. VOLVER AL PASO 2 (excepto si el comando es “exit”)
```

Obviamente, este pseudocódigo indica únicamente como debe ser la estructura principal del programa, pero no dice nada acerca de cómo deben ejecutarse los comandos introducidos por el usuario (paso 3.3 del algoritmo), diseñar y programar esos procedimientos es tarea del alumno.

IMPORTANTE*:** Explícitamente se exige que el programa no muestre por pantalla más texto ni mensajes de los indicados en el enunciado, tampoco deberán hacerse pausas del tipo "Pulse INTRO para continuar...***", tampoco deberá borrarse el contenido de la pantalla en ningún caso.

IMPLEMENTACIÓN: Detalles y normas de obligado cumplimiento

Para la representación y almacenamiento de los datos en memoria se deberán usar **OBLIGATORIAMENTE** las estructuras de datos que se van a describir a continuación. Los diferentes tipos de datos se definen como el siguiente tipo enumerado de C:

```
typedef enum {VOID, NUM, DMY, YMD, TIME, STR} TYPE;
```

Un conjunto de datos se va a representar como una lista de columnas, cada columna será de alguno de los tipos definidos previamente, y contendrá, entre otros elementos, un vector de cadenas de caracteres ("*`*data[]`*") para representar los valores de dicha columna, dichos valores se guardarán como cadenas de caracteres independientemente de que sean de tipo NUM, DMY, TIME, etc...:

```
// ESTRUCTURA 'COLUMNA'
typedef struct col
{
    char *name;        // nombre de la columna
    TYPE t;            // tipo de la columna
    int n;             // número de datos en la columna
    char **data;       // vector dinámico de cadenas para valores
    struct col *next; // puntero a la siguiente columna
} COLUMN;

// ESTRUCTURA 'DATOS'
typedef struct
{
    int nCols; // número de columnas
    COLUMN *p; // puntero a la primera columna de datos
} DATA;
```

La utilización de estas estructuras de datos implica realizar gestión de memoria dinámica, reservando la memoria que se vaya a necesitar y liberándola cuando ya no sea necesaria.

Cada vez que se ejecute un comando que requiera reservar memoria deberá comprobarse que efectivamente dicha memoria se ha reservado correctamente, en caso de que no sea así, el comando no se podrá ejecutar y el programa deberá mostrar en pantalla un mensaje indicando el error por falta de memoria.

En esta práctica hay que hacer un uso intensivo de varias formas de operaciones con cadenas de caracteres por lo que, para abordar la implementación de la aplicación, se necesita una librería de funciones que permitan hacer ciertas operaciones con dichas cadenas, a continuación se describe la funcionalidad de algunas de esas operaciones que se recomienda implementar en forma de librería:

1. Una función que, dada una cadena de caracteres y un separador (que será otra cadena más corta), devuelva el número de campos que hay en dicha cadena.
2. Una función que, dada una cadena, un sepàrador y un número 'i', devuelva el valor del campo i-ésimo contenido en esa cadena.
3. Una función que valide si una cadena tiene formato (o forma) de número entero válido con opción a que pueda empezar con un signo + (número positivo) o - (número negativo).
4. Una función análoga a la anterior para números reales (con decimales).
5. Una función que devuelva el valor numérico de una cadena que representa un valor entero.
6. Una función que devuelva el valor numérico de una cadena que representa un valor real.
7. Una función que determine si una cadena de caracteres representa una fecha válida (para ambos formatos, YMD y DMY).
8. Una función que elimine de una cadena los espacios en blanco, tabuladores '\n' y '\r' que pueda contener al principio y al final (ver función '*trim*' en otros lenguajes de programación, p.e., trim() en PHP)'.

La forma (prototipo) que deben tener estas funciones es algo secundario y queda a elección del alumno, lo importante es disponer de dichas funciones para abordar el resto del problema. Por convenio la librería que contenga dichas funciones deberá llamarse "**cadenas**", es decir, deberá estar formada por los ficheros "*cadenas.h*" y "*cadenas.c*".

Por otra parte, la práctica también debe contener la librería "**datos**" (ficheros "*datos.h*" y "*datos.c*"), para albergar las definiciones de tipos y estructuras descritos anteriormente, así como todas las funciones que el alumno necesite crear para manipular la lista de columnas que debe crearse en memoria.

Como en todo programa que manipula estructuras de memoria dinámica, es muy importante definir y programar funciones para construir y destruir dichas estructuras, es

decir, para reservar y liberar memoria. Algunas de las funciones que debe contener la librería “datos” son las siguientes:

`void InicializaDatos(DATA *d);`

Inicializa las propiedades de la estructura ‘d’, que se pasa por referencia (poner contadores a cero y punteros a NULL).

`int CrearDatos(DATA *d, int c, char *n[], TYPE t[], int f);`

Reserva memoria para un fichero de datos de ‘c’ columnas. El vector de cadenas ‘n’ indica los nombre de cada columna, y el vector ‘t’ indicará los tipos de dichas columnas. El parámetro ‘f’ indica el tamaño o longitud de cada columna de datos (equivale al número de filas de datos correctas en el fichero CSV). La memoria reservada se devolverá en el argumento ‘d’ que se pasa por referencia. La función devuelve 1 (verdadero) si termina correctamente, devolverá 0 (falso) en caso de falta de memoria, si este fuera el caso, la función deberá liberar la memoria que haya podido reservar antes del fallo de memoria e inicializar la estructura ‘d’.

`void EliminarDatos(DATA *d);`

Libera la memoria e inicializa la estructura de datos ‘d’, que se pasa por referencia.

`COLUMN *CrearColumna(char *n, TYPE t, int f);`

Reserva memoria para una columna de nombre ‘n’, de tipo ‘t’ y con una cantidad de datos ‘f’. Devuelve un puntero a la columna reservada o NULL en caso de fallo de memoria. Si esto último ocurriera la función liberará la memoria que hubiera reservado antes de dicho fallo.

`void EliminarColumna(COLUMN *col);`

Libera la memoria de datos ‘col’, que se pasa por referencia.

Estas son 5 funciones que OBLIGATORIAMENTE deberá contener la librería “datos”. Adicionalmente, el alumno podrá añadir las funciones que estime oportunas para manipular la estructura de datos.

FUNDAMENTOS DE PROGRAMACIÓN

EXAMEN - DICIEMBRE 2018

Ejercicio 1 (2 puntos)

Modificar el prompt para que incorpore un contador de cambios, cada vez que se ejecute un comando que modifique los datos, el contador deberá incrementarse, inicialmente, y cada vez que se guarden los datos en disco, deberá ponerse el contador a cero.

`*[0]:>>`

Ejercicio 2 (2 puntos)

Nuevo comando: **`del <column>`**

Este comando debe eliminar de memoria la columna indicada. Si dicha columna no existe, el programa responderá con un mensaje de error apropiado.

Ejercicio 3 (2 puntos)

Nuevo comando: **`see [<num>]`**

Muestra por pantalla todos los datos cargados en memoria o, si se indica el parámetro 'num' (número entero positivo), se mostrarán las 'num' primeras. Si no hay datos para mostrar se indicará con un mensaje de error apropiado.

Ejercicio 4 (2 puntos)

Nuevo comando: **`serial <column1> <column2>`**

La primera columna, 'column1', debe ser de tipo fecha (DMY o YMD), el comando creará una nueva columna en el conjunto de datos de nombre 'column2' de tipo 'NUM' (no deberá existir otra columna con ese nombre) con los números de serie de una fecha tal que el 1 equivalga a 01/01/2000, el 2 a 02/01/2000, etc… En caso de error, indíquese.

Ejercicio 5 (2 puntos)

Nuevo comando: **`sort <column>`**

Ordenar TODO el conjunto de datos de menor a mayor por la columna indicada

---

Avisos:

- La duración del examen es de 3 horas
- Si la práctica contiene algún error, este podría restar a la nota final del examen
- **MUY IMPORTANTE:** MOSTRAR POR PANTALLA AL PRINCIPIO DEL PROGRAMA QUE EJERCICIOS ESTÁN HECHOS Y CUÁLES NO **(no hacerlo restará 1 punto)**
  - **`- Ejercicio 1: HECHO / SIN HACER`**
  - **`- Ejercicio 2: . . .`**

FUNDAMENTOS DE PROGRAMACIÓN

EXAMEN - ENERO 2019

Ejercicio 1 (2.5 puntos)

Modificar el comando **save** para que, en caso de que el fichero donde se va a escribir ya exista, el programa pregunte si se quiere continuar o cancelar. Si el usuario escribe “y” o “yes”, el programa guarda los datos sobreescribiendo el fichero antiguo, y si escribe cualquier otra cosa se cancela la operación.

Ejercicio 2 (2.5 puntos)

Modificar la aplicación en general para que el carácter ‘-’ valga como separador en las fechas, es decir serán fechas válidas todas las siguientes: 27/01/2019, 27-01-2019, 2019/01/27 y 2019-01-27.

Ejercicio 3 (2.5 puntos)

Crear nuevo comando para concatenar cadenas:

```
concat <col1> <col2> <col3>
```

El comando concatena los valores de las cadenas ‘col1’ y ‘col2’ y guardará el resultado en una nueva columna de nombre ‘col3’ (no debe existir una columna con el nombre ‘col3’).

Ejercicio 4 (2.5 puntos)

Crear un comando para añadir datos a los datos que ya hay en memoria :

```
addfile <name_file> <sep>
```

Los datos del fichero ‘name_file’ (con el separador ‘sep’) se añaden como nuevas filas de datos a los datos que ya hay en memoria. Se debe cumplir la condición de que el nuevo fichero debe tener las mismas columnas y del mismo tipo que las que hay en memoria

---

Avisos:

- La duración del examen es de 3 horas
- Si la práctica contiene algún error, este podría restar a la nota final del examen
- **MUY IMPORTANTE:** MOSTRAR POR PANTALLA, INDICAR AL PRINCIPIO DEL PROGRAMA QUE EJERCICIOS ESTÁN HECHOS Y CUÁLES NO **(no hacerlo restará 1 punto)**

```
- Ejercicio 1: HECHO / SIN HACER
- Ejercicio 2: . . .
```

FUNDAMENTOS DE PROGRAMACIÓN

EXAMEN - SEPTIEMBRE 2019

Ejercicio 1 (2.5 puntos)

Modificar comando: **`sort <column> [ASC/DESC]`**

Modificar el comando sort para que admita de forma opcional si se quiere ordenar el conjunto de datos de menor a mayor (ASC) o de mayor a menor (DESC)

Ejercicio 2 (2.5 puntos)

Modificar comandos sum/sub: **`sum <column1> <column3> [<column3>]`** (‘***sub***’ análogo)

Ahora la tercera columna es opcional, en caso de no indicarse dicha tercera columna el resultado de la suma/resta se sobreescribirá en column1.

Ejercicio 3 (2.5 puntos)

Nuevo comando: **`trimester <column1> <column2>`**

La columna <column1> debe ser de tipo fecha (YMD o DMY). Se creará una nueva columna de nombre <column2> (no debe existir otra columna con ese nombre) de tipo STR, que contendrá los valores T1, T2, T3 o T4 según la fecha correspondiente pertenezca al 1º, 2º, 3º o 4º trimestre del año.

Ejercicio 4 (2.5 puntos)

Nuevo comando: **`histogram <column> n`**

Realiza los cálculos para un histograma de ‘n’ intervalos. <column> debe ser de tipo numérico y ‘n’ debe ser un entero positivo mayor que 1 (2 en adelante). Por pantalla se mostrarán los intervalos y el número de valores de cada uno de ellos. Ejemplo para n=3:

```
[min, val1): 13
[val1, val2): 19
[val2, max]: 8
```

Avisos:

- La duración del examen es de 3 horas
- Si la práctica contiene algún error, este podría restar a la nota final del examen
- **MUY IMPORTANTE:** MOSTRAR POR PANTALLA, INDICAR AL PRINCIPIO DEL PROGRAMA QUE EJERCICIOS ESTÁN HECHOS Y CUÁLES NO **(no hacerlo restará 1 punto)**

```
- Ejercicio 1: HECHO / SIN HACER
- Ejercicio 2: . . .
```

PREGUNTAS / DUDAS SOBRE EL ENUNCIADO

Ejemplos de Archivos CSV

Hay infinidad de sitios en la web de donde se pueden descargar datos abiertos en formato CSV (y otros), os paso algunos links:

> Contrataciones en municipios de la comunidad valenciana:
> http://www.dadesobertes.gva.es/storage/f/file/20160413192341/contratos---municipios---genero----2016---01.csv
>
> Estaciones meteorológicas (enero 2017)
> http://www.dadesobertes.gva.es/storage/f/file/20160413143852/contaminacion-atmosferica-y-ozono---promedios-diarios---1997-01.csv
>
> Resultados elecciones locales 2015
> http://www.dadesobertes.gva.es/storage/f/file/20160414174803/resultados-locales----2015.csv

Para localizar esos ficheros me he basado en el catálogo de datos abiertos de http://datos.gob.es, filtrando por "CSV" y "Comunidad Valenciana", en el siguiente link hay más:
http://datos.gob.es/es/catalogo?publisher_display_name=Generalitat+Valenciana&res_format_label=CSV&_publisher_display_name_limit=0

MUY IMPORTANTE
Abrid los ficheros para ver cual es su separador, a veces es la coma (,) otras vece es el punto y coma (;), otras puede ser el tabulador (\t), etc... podría cambiar según el origen de datos.

--