Procesamiento de tablas de datos

PRÁCTICA FUNDAMENTOS DE PROGRAMACIÓN

Un fichero CSV (del inglés *'comma-separated values'*) es un fichero de texto plano que contiene una tabla de datos separados por comas. Por extensión, también se llama CSV a cualquier fichero de texto que contenga datos estructurados en forma de tabla aunque use de separador entre valores cualquier otro carácter (espacio en blanco, tabulador, punto y coma, etc...) o conjunto de caracteres (por ejemplo, la doble almohadilla '##' u otros).

Del conjunto de datos de una tabla (o fichero CSV) cada columna será de un determinado tipo, concretamente los datos podrán ser de tipo 'ENTERO', 'REAL', 'FECHA' o 'TEXTO'. Para expresar estos tipos en C/C++ se usará la siguiente definición de tipo enumerado:

```
typedef enum {NULO, ENTERO, REAL, FECHA, TEXTO} TIPO;
```

El tipo 'NULO' se usará para indicar situaciones anómalas, es decir, para cuando el tipo que se deba considerar no coincida con ninguno de los otros cuatro.

Obviamente, todas las filas de un fichero CSV deben tener el mismo número de campos (o columnas) y dichos campos deben ser del mismo tipo. Por ejemplo, si la primera fila de un fichero tiene esta forma:

23, 35.6, ALTO, -52, 12/06/2012, OK

Considerando que el separador de valores es el carácter coma (,) se ve claramente que hay 6 campos (o columnas) cuyos tipos son:

ENTERO, REAL, TEXTO, ENTERO, FECHA, TEXTO

El resto de filas del fichero deberá tener el mismo número de campos y además del mismo tipo y en el mismo orden. Cualquier fila que tenga más o menos de 6 campos se considerará una fila errónea (también si la fila está vacía y tiene cero campos); igualmente serán erróneas las filas que, teniendo 6 campos, sus tipos no coincidan (en el mismo orden) con los tipos de esa primera línea, es decir sería errónea una fila, también con 6 campos, como esta:

44, -88.9, 96, BAJO, 21/12/2011, FALLO
(ENTERO, REAL, ENTERO, TEXTO, FECHA, TEXTO)

A partir de esta descripción, se pide implementar en C++ una clase "TABLA" que permita cargar en memoria los datos contenidos en un fichero CSV y procesarlos, así como desarrollar una aplicación

que permita hacer uso de la funcionalidad de esa clase; en los apartados siguientes se pasa a describir detalladamente la funcionalidad de la clase “TABLA” y las operaciones que debe poder ejecutar la aplicación.

Descripción de la clase

La definición de la clase TABLA debe ser como sigue (*exactamente igual*):

```
class TABLA
{
      public:
            TABLA();
            ~TABLA();
            int   Cargar(const char *nomFichero, const char *sep);
            int   Guardar(const char *nomFichero, const char *sep);

            int   NumFilas();
            int   NumColumnas();
            TIPO  TipoColumna(int col);
            TIPO  Valor(int fil, int col, char *val);

            int   ValoresColumna(int col);
            int   EliminarColumna(int col);
            int   OrdenarPor(int col);

            int   NumErrores();
            int   LineaError(int e);
            char* Error(int e);

      private:
            int     columnas, filas;
            TIPO   *tipos;
            char ***datos; /* triple puntero a char → matriz de cadenas */

            int    numErrores;
            int   *filasError;
            char **errores;
};
```

Los parámetros de la parte privada tienen la siguiente utilidad:

- **columnas, filas:**
  Almacenan el número de columnas (campos) y filas que tiene la matriz.

- **tipos:**
  Vector dinámico de tantas componentes como columnas haya en el fichero, debe almacenar de que tipo es cada columna.

- **datos:**
  Es un triple puntero a ‘char’ esto debe considerarse como una matriz de dos dimensiones (de ahí los dos primeros asteriscos) donde cada uno de los valores de esta es una cadena de caracteres (de ahí el tercer asterisco), por lo tanto, dado un fichero CSV de 5 columnas y 20 filas, esta clase deberá reservar dinámicamente memoria para una matriz de 5 x 20 (5 columnas y 20 filas) y posteriormente en cada valor habrá que reservar memoria y copiar, como cadena de caracteres, los 100 valores (5x20) del fichero, cada uno en la posición que le corresponda.

- **numErrores:**
  Valor entero para guardar el número de filas erróneas que se han encontrado en el fichero CSV.

- **filasError:**
  Es un vector dinámico de tamaño ‘numErrores’ (parámetro anterior) para guardar la posición, dentro del fichero CSV, que ocupa cada una de las filas erróneas. Las líneas deben empezar a contarse desde la número uno.

- **errores:**
  Es un vector dinámico, también de tamaño ‘numErrores’, de cadenas de caracteres donde se almacenarán todas las líneas del fichero que hayan sido erróneas

A continuación se va a describir el funcionamiento que debe tener cada uno de los métodos.

- **Cargar:**
  Abre el fichero con nombre igual al valor del parámetro ‘nomFichero’ y lo procesa almacenando en los parámetros de la parte privada toda la información relevante, número de filas y columnas, los tipos de cada columna, los datos y los errores detectados. Se debe usar como separador el valor indicado por el parámetro ‘sep’. El método devuelve verdadero (1) si la operación se realiza sin problemas o falso (0) si se produce algún error (el fichero no existe o no se puede abrir, ocurre un error al reservar memoria dinámica, etc...).

  IMPORTANTE: se debe hacer una gestión adecuada de la memoria, reservando de forma dinámica solamente la que se necesite, tanto para guardar los datos de la tabla como los posibles errores detectados.

- **Guardar:**
  Guarda en un fichero los datos de la tabla (no los errores), el fichero se deberá llamar como indique el valor del parámetro ‘nomFichero’ y como separador se usará el valor del parámetro ‘sep’.

- **NumFilas:**
  Devuelve el número de filas de la tabla.

- **NumColumnas:**

Devuelve el número de columnas de la tabla.

- **TipoColumna:**
  Devuelve el tipo de dato de la columna indicada en el parámetro ‘col’. Si ‘col’ se sale de las dimensiones de la tabla, es decir, si es menor que cero o mayor o igual que el número de columnas devuelve ‘NULO’.

- **Valor:**
  Devuelve por referencia en el parámetro ‘val’ una copia del dato que se encuentra en la tabla en la fila ‘fil’ y columna ‘col’, tras lo cual se devuelve el tipo de dato a que se refiere el valor. Si los parámetros ‘fil’ y/o ‘col’ se salen de los límites de la tabla, ‘val’ no se modifica y se devuelve ‘NULO’.

- **ValoresColumna:**
  Devuelve la cuenta de cuántos valores distintos hay en una columna si esta es de tipo ‘TEXTO’. Si la columna es de otro tipo o ‘col’ se sale de los límites de la tabla devuelve ‘0’.

- **EliminarColumna:**
  Elimina de la tabla de datos la columna indicada por el parámetro ‘col’, liberando la memoria correspondiente y reordenando el resto de columnas (si se elimina la columna 3, la 4 pasa a ser la nueva 3, la 5 la nueva 4, etc...). Si la operación se realiza correctamente el método devuelve verdadero, y si algo falla y no se pudiera realizar la operación devuelve falso.

- **OrdenarPor:**
  Ordena todas las filas del fichero de menor a mayor usando como criterio el valor numérico de la columna indicada por ‘col’ y devuelve verdadero. Si la columna no es numérica (ENTERO o REAL) o se sale de los límites de la tabla no se ordena nada y se devuelve falso.

- **NumErrores:**
  Devuelve el número de filas erróneas encontradas en el fichero.

- **lineaError:**
  Devuelve el número de la línea donde se encuentra el error i-esimo (parámetro ‘e’)

- **Error:**
  Devuelve el puntero a la cadena de caracteres que contiene la línea errónea número ‘e’. Si el valor de ‘e’ se sale del rango permitido (entre 1 y el número de errores) se devuelve NULL.

OJO: no confundir ‘**NULL**’, definición de un puntero nulo en C/C++ con ‘**NULO**’, tipo de dato nulo definido en esta práctica dentro del tipo enumerado ‘**TIPO**’. El primero es una definición estándar de C/C++ y el segundo es una definición particular solo válida en esta práctica.

Descripción de la aplicación

La aplicación a implementar consistirá en un programa que, se ejecutará por línea de comandos recibiendo diversos parámetros, leerá y procesará un fichero de datos en función de lo indicado en dichos parámetros. Obviamente, esta aplicación deberá hacer uso de la clase TABLA para realizar la lectura y procesamiento del fichero.

Para que una aplicación pueda recibir parámetros por línea de comandos deberá construirse una función '*main( )*' con argumentos 'argc' y 'argv' como se muestra en el ejemplo siguiente:

```
int main(int argc, char *argv[]);
{
       . . .

}
```

Donde los argumentos 'argc' y 'argv' son, el primero, el número de argumentos introducidos por el usuario, y el segundo, un vector de cadenas con los valores de dichos parámetros.

A continuación, para detallar el funcionamiento esperado de la aplicación. Se describe una serie de comandos que la aplicación debe poder entender y ejecutar adoptando los siguientes convenios:

- **app** (o app.exe): nombre de la aplicación, es decir, el ejecutable (después de compilar).
- **file, file1, file2**: nombres de ficheros de texto.
- **sep, sep1, sep1**: valores a utilizar como separadores.
- **n, m**: valores de tipo entero.

Considerando estos convenios, la aplicación tendrá que soportar los siguientes comandos:

app VER file sep

Muestra por pantalla los datos (no los errores) del fichero en forma de tabular, tal cual han sido leídos del fichero original (considerando 'sep' como separador de campo en dicho fichero).

app VER file sep n

Muestra por pantalla los datos (no los errores) de las 'n' primeras filas del fichero en forma tabular, tal cual han sido leídos del fichero original ('sep' es el separador de campo).

app ERRORES file sep

Muestra por pantalla un listado de los errores detectados en el fichero 'file'. Por cada error se imprimirá una línea como esta:

nº : linea_con_el_error

Donde 'nº' es el número de línea donde está el error y 'linea_con_el_error' es la mencionada línea que contiene el error.

app COPIAR file1 sep1 file2 sep2

Hace una copia de los valores del fichero file1 (con separador 'sep1') en otro fichero file2 (con separador 'sep2'). Se copian solo los valores de las filas correctas, los errores se ignoran.

app INFO file sep

Muestra en pantalla la siguiente información del fichero:

matriz: filas n, columnas m
columna 1: tipo_1 (info*)
columna 2: tipo_2 (info*)
… (para todas las columnas del fichero indicar de qué tipo son)
Nº errores: valor_parametro_numErrores

Esta 'info*' varía según el tipo de la columna , si es de tipo FECHA no se pone nada, si es de tipo TEXTO se pone el número de valores diferentes y si es de tipo ENTERO o REAL se indican cuales son los valores máximo y mínimo de la columna.

app ELIMINAR file1 sep1 n file2 sep2

Elimina de file1 (con separador 'sep1') la columna 'n' y vuelve a guardar la tabla de datos resultante en el fichero file2 (con separador 'sep2')

Importante:
Nótese que al tratar de ejecutar estos comandos es posible que haya problemas que impidan su ejecución, por ejemplo:

- Un fichero podría no existir y por tanto no se podría abrir.
- Un valor 'n' podría ser cero o negativo o mayor del rango esperado.
- Alguno de los métodos de la clase TABLA utilizada podría devolver falso (0) al no poder hacer la operación solicitada.
- etc...

Además, puede ser que al introducir un comando manualmente por teclado este se introduzca mal, los argumentos sean incorrectos y/o no sean del tipo adecuado (por ejemplo: un texto donde debería haber un número).

Se valorará que el programa detecte estas situaciones y no se cuelgue, y además, en cada caso muestre un mensaje de error por pantalla indicando cuál ha sido el problema. Estos mensajes de error deberá darlos la propia función 'main' NUNCA LOS MÉTODOS DE LA CLASE.

Sugerencias:

En esta práctica hay que hacer un uso intensivo de varias formas de operaciones con cadenas de caracteres por lo que se recomienda, antes de abordar la implementación de la clase o la aplicación disponer de una librería de funciones que permitan hacer algunas operaciones con dichas cadenas, a continuación se describe la funcionalidad de algunas de esas operaciones que se recomienda implementar primero 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 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 dicha cadena 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.
8. Una función que permita comparar dos cadenas en forma de fecha y determine cual es mayor, menor o si son iguales.
9. 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)'.

La forma (prototipo) que deben tener estas funciones es algo secundario, lo importante es poder disponer de ellas para abordar el resto del problema, se deja a elección de cada alumno el diseño de los prototipos para estas funciones según se estime más conveniente.

Librería sugerida:
(ojo, es una sugerencia, cada cual en su práctica podrá hacer otra más o menos parecida)

```
int FechaValida(const char *cad);
```
Devuelve 1 si la cadena "cad" es una fecha válida y 0 si no lo es.

```
int EnteroValido(const char *cad, int *val);
```
Devuelve 1 si la cadena "cad" es un entero válido y 0 si no lo es. En caso de que el entero sea correcto se devuelve su valor en el parámetro "val" que se pasa por referencia.

```
int RealValido(const char *cad, double *val);
```
Devuelve 1 si la cadena "cad" es un entero válido y 0 si no lo es. En caso de que el entero sea correcto se devuelve su valor en el parámetro "val" que se pasa por referencia.

```
int FechaComp(const char *cad1, const char *cad2);
```

Partiendo de que los parámetros “cad1” y “cad2” son fechas, esta función devuelve -1 si “cad1” es menor que “cad2”, devuelve 0 si son fechas iguales y devuelve 1 (positivo) si “cad1” es mayor que “cad2”.

`TIPO ObtenerTipo(const char *valor);`
Devuelve el tipo (FECHA, REAL, ENTERO o TEXTO) al que pertenece el valor que se para como argumento. siendo ‘TIPO’ el tipo enumerado definido al principio del enunciado de la práctica.

`void Trim(char *cad);`
Modifica la cadena que se pasa como parámetro eliminando todos los caracteres no visibles del principio y final de la misma. Por caracteres no visibles se entiende el espacio en blanco, el tabulador (‘\t’), el salto de línea (‘\n’) y el retorno de carro (‘\r’).

`int NumCampos(const char *lin, const char *sep);`
Devuelve el número de campos que hay en una línea dada (‘lin’) considerando que dichos campos están separados por el separador (‘sep’) indicado.

`int ObtenerCampo(const char *lin, const char *sep, int pos, char *campo);`
Trata de obtener un campo determinado de la línea (‘lin’) indicada. Se considera que el separador de campos es el parámetro ‘sep’ y que el campo que se quiere obtener está en la posición ‘pos’, empezando a contar desde cero. Al final, el campo obtenido se guarda en el parámetro que se pasa por referencia ‘campo’. Si toda la operación se ha podido hacer sin problema la función devuelve verdadero (1), si ocurre algún problema, no se obtiene ningún campo y se devuelve falso (0).

ANEXO 1: ENUNCIADO DEL EXAMEN DE DICIEMBRE DE 2012
TIEMPO 2h. 00min.

Método para guardar los errores detectados:

int GuardarErrores(const char *nomFich);
(2,5 puntos)

Guardar en un fichero de texto los errores encontrados en el fichero de datos leído, si no hay errores que guardar devuelve 'falso'. El fichero de error tendrá la forma:

[4] - fila_erronea_numero_4
[13] - fila_erronea_numero_13
[55] - fila_erronea_numero_55
...

Método para generar columna trimestre (valores: "T1", "T2", "T3" y "T4"):

int AddCuatrimestre(int colFecha);
(2,5 puntos)

A partir de una columna de tipo fecha, se crea una nueva columna de nombre "Trimestre" (de 'TIPO' texto) que, para cada fila, contenga uno de los siguientes valores, "T1", "T2", "T3" o "T4", según la fecha se corresponda con una del trimestre 1º, 2º, 3º o 4º.

Método para añadir registros de datos a una tabla ya cargada en pantalla:

int MasRegistros(const char *nomFich);
(2,5 puntos)

Una vez cargado un fichero de datos, esta función carga otro nuevo añadiendo los registros de este a los que ya tenía en memoria, reservando la memoria que sea necesario para guardar dichos registros y los nuevos errores que se hayan detectado. El nuevo fichero debe tener las mismas columnas y del mismo tipo que el que se ha leído previamente, de no ser así no se añade nada y se devuelve 'falso'

Los 2,5 puntos que faltan se evaluaron con la corrección de la práctica

ANEXO 2: ENUNCIADO DEL EXAMEN DE FEBRERO DE 2013
TIEMPO 2h. 30min.

EXAMEN
FUNDAMENTOS DE PROGRAMACIÓN

Método para guardar los errores detectados:
(2,0 puntos)

int GuardarErrores(const char *nomFich);

Guardar en un fichero de texto los errores encontrados en el fichero de datos leído, si no hay errores que guardar devuelve 'falso'. El fichero de error tendrá la forma:

[4] - fila_erronea_numero_4
[13] - fila_erronea_numero_13
[55] - fila_erronea_numero_55
...

Cálculos sobre columnas numéricas:
(2,0 puntos)

int Calculos(int numCol, TValores &v);

Siendo 'numCol' una columna de tipo numérico (REAL o ENTERO), devuelve '1' (verdadero) y pone en el parámetro 'v' la suma y el promedio de dichos valores. Si 'numCol' no es una columna numérica devuelve '0' (falso). 'TValores' debe ser una estructura definida del siguiente modo:

```
typedef struct
{
     double sum; // sumatorio de la columna
     double med; // promedio de los valores de la columna
} TValores;
```

Ordenar los datos, también, por columnas de tipo TEXTO:
(2,0 puntos)

Modificar el método ‘Ordenar’ para que también ordene el conjunto de datos usando columnas de tipo TEXTO (ver función ‘strcmp’ de la librería string.h)

Discretizar una columna numérica en 2 valores ‘BAJO’/‘ALTO’:
(2,0 puntos)

int DiscretizarEnDos(int numCol, double umbral);

A partir de la columna ‘numCol’, que debe ser numérica (ENTERO o REAL), se añade una nueva columna de tipo TEXTO a la matriz de datos cuyos valores serán “BAJO” o “ALTO” según los valores de la columna indicada sean menores-iguales o mayores que el parámetro ‘umbral’. Al final devuelve ‘1’ (verdadero). Si ‘numCol’ no es una columna válida devuelve ‘0’ (falso).

Modificar el comando ELIMINAR:
(2,0 puntos)

app ELIMINAR file1 sep1 n file2 sep2 m

Tiene un comportamiento similar al de la práctica, solo que el fichero de salida, además de eliminar la columna indicada (‘n’), solo dispondrá de las ‘m’ primeras filas del fichero original (obsérvese que el parámetro ‘m’ es nuevo, no estaba en el comando ELIMINAR original que se pedía en la práctica)

ANEXO 3: ENUNCIADO DEL EXAMEN DE SEPTIEMBRE DE 2013
TIEMPO 2h. 30min.

EXAMEN
FUNDAMENTOS DE PROGRAMACIÓN

Cálculos sobre columnas numéricas:
(2,0 puntos)

int Calculos(int numCol, TValores &v);

Siendo 'numCol' una columna de tipo numérico (REAL o ENTERO), devuelve '1' (verdadero) y pone en el parámetro 'v' la suma, el promedio, el mínimo y el máximo de dichos valores. Si 'numCol' no es una columna numérica devuelve '0' (falso).
'TValores' debe ser una estructura definida del siguiente modo:

```
typedef struct
{
     double sum; // sumatorio de la columna
     double med; // promedio de los valores de la columna
     double min; // valor mínimo de la columna
     double max; // valor máximo de la columna
} TValores;
```

Ordenar los datos, también, de mayor a menor:
(2,0 puntos)

int Ordenar(int col, const char *modo);

Modificar el método 'Ordenar' para que trabaje en función del parámetro 'modo'. Ordenará los datos de menor a mayor cuando 'modo' tenga el valor "ASC" (ascendente) y de mayor a menor cuando 'modo' valga "DESC" (descendente). Si 'modo' toma un valor distinto a estos dos no se ordena nada y se devuelve '0' (falso).

Truncar decimales:
(2,0 puntos)

int Truncar(int numCol);

Siendo 'numCol' una columna de tipo 'REAL', este método deberá truncar los decimales y convertir la columna en tipo 'ENTERO'. Ojo, hay que cambiar tanto los valores de la columna como el tipo de la misma. Al terminar devuelve '1' (verdadero). Si la columna indicada en 'numCol' no es de tipo REAL o se sale de la matriz el método no hace nada y devuelve '0' (falso).

Método para generar columna mes-año:

(2,0 puntos)

int AddMesAnyo(int colFecha);

Siendo 'colFecha' una columna de tipo 'FECHA', se crea una nueva columna de nombre "MesAnyo" (de tipo 'TEXTO') que, para cada fila, contenga el valor del año y del mes correspondiente, es decir, si la fecha es 23/11/2013, el valor que debe tener el nuevo campo es "2013-11" (de tipo texto), al terminar devuelve 1 (verdadero). Si la columna indicada no es de tipo fecha o se sale de la matriz o sucediera algún otro problema devuelve 0 (falso).

Añadir el tipo "FECHA2":

(2,0 puntos)

Se debe añadir un nuevo tipo de datos a los ya existentes, el tipo "FECHA2" que consiste en una fecha expresada en formato "AAAA/MM/DD", el método encargado de leer el fichero de datos tiene que reconocer este nuevo tipo validarlo igual que la fecha normal, es decir, comprobar si los meses son de 28, 30 o 31 días, si el año es o no bisiesto, etc...