Pseudo BD

(falsa base de datos)

PRÁCTICA DE FUNDAMENTOS DE PROGRAMACIÓN

Se pide realizar un programa en C que emule de forma rudimentaria el funcionamiento de una aplicación de gestión de bases de datos, utilizando la línea de comandos para su funcionamiento. El programa podrá gestionar varias bases de datos, cada una de las cuales estará contenida en una carpeta (directorio) distinta en el disco duro.

La aplicación admitirá tres tipos de datos, ***NUMERO***, ***FECHA*** y ***TEXTO***. El tipo numérico incluye tanto enteros como reales, la fecha deberá estar expresada en formato DD/MM/AAAA (dos dígitos para el día, dos para el mes y cuatro para el año separados por el carácter '/'), y el tipo texto podrá contener valores de cualquier combinación de caracteres de cualquier tamaño (también vacío). En el programa a realizar en C, se definirán estos tipos de datos con la siguiente enumeración:

```
typedef enum { NUM, DATE, TEXT } TYPE;
```

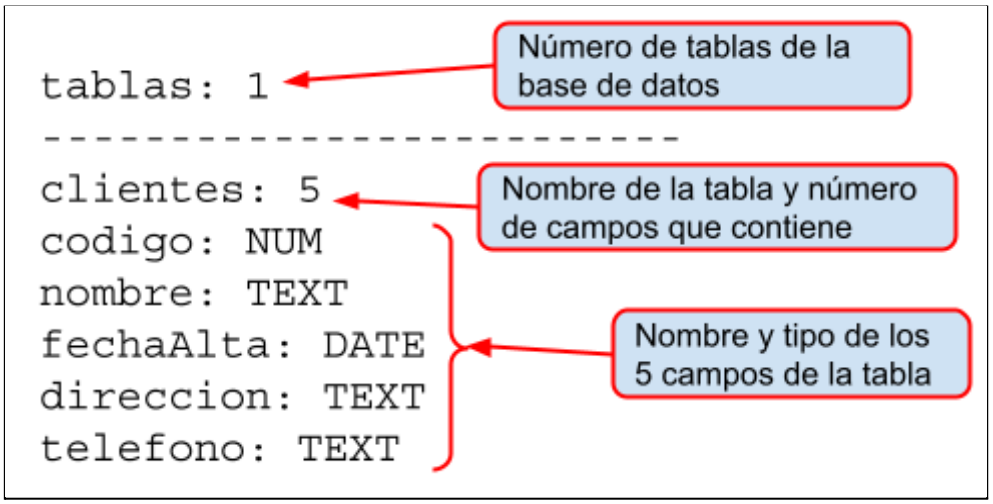
Todos los datos de cada una de las tablas de la base de datos se almacenarán en disco, en un fichero de texto que contendrá dichos datos en formato CSV (separados por comas). En dicho fichero, los valores de tipo texto (TEXT) irán siempre entre comillas dobles (los valores numéricos y de tipo fecha irán siempre sin comillas). El nombre de cada fichero de texto se llamará igual que la tabla con la extensión ".tab", por ejemplo, si en nuestra base de datos creamos una tabla "clientes", el fichero correspondiente se deberá llamar:

`clientes.tab`

Adicionalmente, la base de datos almacenará sus metadatos en un fichero que ha de llamarse:

`meta.db`

Este fichero de metadatos debe contener información sobre el número de tablas que tiene la base de datos, el nombre de cada una de esas tablas, el número de campos de cada tabla, y los nombre y tipos de datos de cada campo. Por ejemplo, una base de datos con una sola tabla "***clientes***" con los campos: ***codigo{NUM}, nombre{TEXT}, fechaAlta{DATE}, dirección{TEXT}, telefono{TEXT}***, en el fichero de metadatos se representará del siguiente modo:



Cada nueva tabla que se añada a esta base de datos se incorporará el fichero de metadatos, separando de la tabla anterior con una línea discontinua, y añadiendo debajo una línea con el nombre de la tabla y a continuación tantas líneas como campos tenga la tabla con los nombres y tipos de dichos campos.

Un posible fichero de datos “clientes.tab” con dos registros (dos clientes) podría ser el siguiente:

```
1, "Antonio Lopez", 25/10/2020, "Calle de los pinos, 1", "666777888"
2, "Luis Mas", 12/02/2021, "Avd. del Mar, 33, 4°-A", "666999777"
```

Para operar sobre una base de datos en concreto, primero habrá que seleccionarla como base de datos activa (ver comandos más adelante). De igual forma, para actuar sobre una tabla de la base de datos activada, también habrá que seleccionarla como tabla activa (ver comando más adelante). Cuando se activa una base de datos hay que guardar en memoria sus metadatos en una lista doble de nodos de tipo **TABLE**:

```
typedef struct tab
{
    char nombre[100];
    int numCampos;
    char **campos;
    TYPE *tipos;
    struct tab *sig, *ant;
}
TABLE;
```

Cuando se activa una tabla, también hay que cargar en memoria sus datos en una lista doble de registros que contendrá todas las líneas del fichero de dicha tabla. Los nodos de esta línea serán de tipo **ROW**:

```
typedef struct line
{
    char **valores;
    struct line *sig, *ant;
}
ROW;
```

Siempre que se cambie la tabla activa, se vaciará debidamente la memoria de la tabla que haya en ese momento en memoria antes de cargar los datos de la nueva tabla. Igualmente, cuando se cambie de base de datos activa, se liberará tanto la memoria de la base de datos actual, como la de la tabla que hubiera activada en ese momento.

Sobre el funcionamiento general del programa, al arrancar imprimirá en pantalla el nombre y apellidos del alumno y su email, y a continuación, para que el usuario pueda introducir comandos desde teclado, mostrará el siguiente prompt:

```
[*]:>
```

Cuando haya una base de datos activa, el prompt cambiará, indicando el nombre de dicha base de datos, por ejemplo, si se activa la base de datos llamada "***empresa***", el prompt cambiará a:

```
[empresa]:>
```

Además, cuando haya una tabla activa, el prompt volverá a cambiar para indicarlo, por ejemplo, si en la base de datos "empresa" se activa la tabla "***clientes***", el prompt quedará como sigue:

```
[empresa/clientes]:>
```

Importante: Como norma general, debe cumplirse siempre que todos los nombres de las bases de datos, las tablas y los campos dentro de las tablas tendrán que formarse usando solamente caracteres alfanuméricos estándar (a...z, A...Z, 0...9) y la barra baja (_), no se permiten eñes, vocales con acentos o diéresis, signos de puntuación, ni otros caracteres diferentes a los mencionados, tampoco se permite usar el espacio en blanco como parte del nombre de una base de datos, una tabla o un campo (*toda esta restricción no aplica a los valores de tipo texto (TEXT) que pueda haber dentro de una tabla*).

En forma de pseudocódigo, el funcionamiento a grandes rasgos del programa debe seguir el siguiente algoritmo:

1. Imprimir en pantalla datos del alumno:
   nombre, apellidos y correo electrónico
2. Mostrar el prompt
3. Leer comando introducido por teclado por el usuario
4. Analizar comando
   ⇒ Si el comando es incorrecto se indica con un mensaje de error y se vuelve al paso 2
5. Determinar si es posible ejecutar el comando
   ⇒ Si no es posible ejecutar el comando se indica con un mensaje de error y se vuelve al paso 2
6. Si el comando es '**exit**' se termina la ejecución del programa
7. Si el comando no es '**exit**' se ejecuta la acción correspondiente y se vuelve al paso 2

En el paso 7, donde dice "***se ejecuta la acción correspondiente***" habrá que realizar alguna de las acciones que se indican en el siguiente listado de comandos que indica las operaciones que deberá hacer nuestro programa.

COMANDOS:

exit

Este comando finaliza la ejecución del programa. Antes de salir y devolver el control a la consola de Windows, hay que liberar debidamente toda la memoria que hubiera reservada en ese momento (paso 6 del pseudocódigo anterior).

new db <nombre_base_de_datos**>**

Crea una nueva base de datos, por tanto crea una nueva carpeta con el nombre de dicha base de datos, pero primero comprueba que el nombre es correcto y que no existe ya una base de datos con igual nombre. Si no fuera posible crear la nueva base de datos el programa imprimirá un mensaje de error indicándolo.

use db <nombre_base_de_datos**>**

Cambia la base de datos activa, de esta manera se le indica al programa que a partir de ahora, todas las operaciones se van a realizar sobre la base de datos indicada. Esta acción cambia el prompt, en el cual se deberá indicar el nombre de la base de datos. Si la base de datos indicada no existe, se indicará con un mensaje de error y no se modificará nada.

new table <nombre_tabla> <campo> <tipo> [<campo> <tipo> ...]

Crea una nueva tabla en la base de datos actual. Se creará un nuevo fichero en blanco (vacío) para dicha tabla y se modificará la lista y el fichero de metadatos para que refleje la existencia de esta nueva tabla. Los nombres de los campos han de ser nombres válidos y no se pueden repetir, los tipos pueden ser NUM, DATE o TEXT. Si ya existe una tabla con el mismo nombre o hay nombres de campos inadecuados o repetidos, entonces el programa mostrará el mensaje de error correspondiente.

use table <nombre_tabla>

Cambia la tabla activa. Se le indica al programa que a partir de ahora, todas las operaciones que afecten a tablas, se realizarán sobre la tabla indicada (mientras no se cambie). Esta acción modifica el prompt y carga en la lista de registros los datos de la tabla activada (libera primero los datos de la tabla anterior si la hubiera). Si el nombre de tabla indicado no se corresponde con el de una tabla existente, el programa responderá con un mensaje de error y no cambia ni hace nada.

insert <valor> [<valor> ...]

Se inserta en la tabla activa el registro formado por los valores indicados. El número de valores debe coincidir con los campos de la tabla activa. Los valores deben ser del mismo tipo que los campos correspondientes de la tabla activa. En el caso de valores de tipo texto, si la cadena tiene más de una palabra, deberá ponerse entre comillas dobles. Si se dan todas las condiciones, se añadirá el registro correspondiente al final del fichero de la tabla activa y a la lista de registros o filas (ROWS). Si la operación no se pudiera realizar por incumplirse alguna de las condiciones necesarias, el programa mostrará un mensaje de error indicando el problema detectado.

select [<campo> ⊗ <valor>] [<orden>]

Este comando, cuando va sin parámetros, muestra por pantalla todos los registros de la tabla activa, primero una cabecera con los nombres de los campos y a continuación, en líneas sucesivas, todos los registros, que deberán ir precedidos de un número que indique su posición en la tabla (es decir, su posición en el fichero o la lista de filas). Si se incluyen los parámetros opcionales **[<campo> ⊗ <valor>]** se mostrarán solo los registros que cumplan la condición especificada; el carácter '⊗' representa cualquiera de los siguientes operadores relacionales: < (menor), > (mayor) o = (igual). El 'campo' y el 'valor' deben ser del mismo tipo. El parámetro opcional '**orden**' podrá ser uno de estos dos valores: '**ASC**' o '**DES**', para forzar a que el listado se muestre por pantalla de forma ordenada, ascendentemente o descendentemente según el valor de la primera columna de la tabla. Importante, si el primer campo es numérico, la ordenación deberá ser según orden numérico, si es una fecha según orden cronológico, y si es texto en orden lexicográfico. El programa deberá responder con un mensaje de error adecuado si no se cumpliera alguna de las condiciones necesarias para que el comando se pueda ejecutar.

Detalles de implementación y recomendaciones:

1. Crear un proyecto con un fichero main.c y una librería formada por los ficheros lib.h y lib.c para declarar y definir funciones adicionales.

2. La librería string.h contiene la función strcmp que sirve para comparar cadenas de caracteres según orden lexicográfico. Se deberá implementar dos funciones análogas a esta para comparar números y fechas según orden numérico y cronológico.

3. El programa manejará dos listas, una de tablas (metadatos), y otra de datos (filas/rows). Se recomienda implementar funciones que permitan recorrer, buscar, reordenar los nodos de dichas listas. También funciones para crear las listas a partir de la información contenida en un fichero y funciones para vaciar las listas y liberar la memoria.

AMPLIACIÓN 1

(EXAMEN - DICIEMBRE 2021)

Ejercicio 1 (2.5 puntos)

Hacer que el comando **`use db`** `<nombre_bd>` muestre por pantalla el listado de tablas disponibles en la nueva base de datos recién activada. Si la base de datos no tiene tablas se indicará con el mensaje "**No hay tablas**".

Ejercicio 2 (2.5 puntos)

Añadir el comando **`list`** (sin parámetros) que muestre por pantalla un listado de todas las bases de datos disponibles. Si aún no hay bases de datos se indicará con el mensaje "**No hay bases de datos**"

Ejercicio 3 (2.5 puntos)

Añadir un nuevo tipo de datos EMAIL para **`new table`**:

```
typedef enum { NUM, DATE, EMAIL, TEXT } TYPE;
```

Las operaciones **`insert`** y **`select`** deben validar, que los valores de tipo EMAIL siguen el patrón ***`[alfanumérico]@[alfanumérico].[alfanumérico]`***

Ejercicio 4 (2.5 puntos)

Añadir el comando **`delete`** para eliminar de la tabla activa todas las filas que cumplan la condición especificada. Se debe actualizar la tabla en memoria y en el fichero en disco.

```
delete <campo> ⊗ <valor>
```

'campo' y 'valor' deben ser del mismo tipo, incluido el nuevo tipo EMAIL
El símbolo '⊗' representa un operador relacional: '**>**', '**<**' o '**=**'
