Práctica 3. Ficheros binarios.

Esta práctica tiene como objetivo conocer cómo se trabaja con ficheros binarios en C++ y cómo se implementan los distintos esquemas. Para ello, se desea gestionar diversa información de los estudiantes de la Universidad de La Rioja. En concreto, se trata de diseñar un programa que se encargue del mantenimiento y actualización de datos de estudiantes, cuando dichos estudiantes cursan una asignatura. Para ello, se utilizarán dos ficheros: un fichero en el que se almacenará la información relacionada con los estudiantes de la Universidad y otro fichero con las notas obtenidas por los estudiantes en las asignaturas que han cursado.

ESTRUCTURA DE LOS FICHEROS

Para la realización de la aplicación es necesario utilizar ficheros binarios, cuya estructura y tipo pasamos a comentar a continuación. La siguiente descripción indicará, para cada fichero: su tipo; su contenido, indicando la estructura registro que almacena (sus campos, y una descripción de su utilidad); y su criterio de ordenación.

1.1 Fichero de Estudiantes:
Tipo: binario

Contenido: registros de tipo `tEstudiante`. Los campos de este registro son:

- *DNI*: cadena de caracteres que almacenará el DNI del estudiante.
- *Nombre*: cadena de caracteres de un tamaño máximo de 30 caracteres con el nombre del estudiante.
- *Apellido*: cadena de caracteres de un tamaño máximo de 50 caracteres con el primer apellido del estudiante (sin espacios en blanco para separar los apellidos compuestos).

El tipo se `tEstudiante` puede modelar con la siguiente estructura:

```
struct tEstudiante{
   char dni[10];
   char nombre[30];
   char apellido[50];
}
```

Orden: el fichero estará ordenado crecientemente por el DNI del estudiante.

1.2 Fichero de Notas:
Tipo: binario

Contenido: registros de tipo `tNotaAsig`. Los campos de este registro son:

- *CodAsignatura*: cadena con el código asociado a la asignatura (podemos suponer que el código de una asignatura está formado por 4 caracteres).
- *DNI*: cadena de caracteres que almacenará el DNI del estudiante.
- *Nota:* número real que corresponde a la nota que el estudiante tiene en la asignatura en un momento del curso.

Puedes modelar el tipo `tNotaAsig` con una estructura similar a la de `tEstudiante`.

Orden: el fichero estará ordenado crecientemente por el código de la asignatura.

APLICACIÓN

La aplicación comenzará comprobando si existe el fichero de estudiantes y el de notas; en caso contrario, los creará. A continuación, la aplicación muestra el menú principal, a partir del cual el usuario podrá elegir distintas operaciones. El menú se mostrará hasta que se introduzca la opción 6. El a*specto del menú principal* será parecido al siguiente.

Menú principal:
1.- Dar de alta
2.- Mostrar información de los estudiantes de una asignatura
3.- Modificar nota de estudiante en una asignatura
4.- Dar de baja estudiante
5.- Nota media de asignatura
6.- Terminar
Elegir opción:

NOTAS SOBRE LA IMPLEMENTACIÓN

En esta práctica tendrás que trabajar con cadenas de caracteres en varios apartados, por lo que es importante recordar que para trabajar con cadenas de caracteres no se compara con el `==` y tampoco se asigna con el `=`, sino que se utilizan respectivamente los métodos `strcmp` y `strcpy` (para lo cual debes incluir la librería `cstring`). Recuerda el comportamiento de `strcmp`:

- `strcmp(A,B)` devuelve un valor menor que 0 si la cadena de caracteres `A` es menor que la cadena de caracteres `B`.
- `strcmp(A,B)` devuelve 0 si la cadena de caracteres `A` es igual que la cadena de caracteres `B`.
- `strcmp(A,B)` devuelve un valor mayor que 0 si la cadena de caracteres `A` es mayor que la cadena de caracteres `B`.

En la práctica deberás definir un método principal y varios subprogramas. En el método principal del programa deberás declarar dos variables de tipo cadena de caracteres llamadas `nombreFEstudiantes` y `nombreFNotas` a las que debes asignarle el nombre de los ficheros que utilizarás como almacén de datos. Estas variables serán utilizadas como parámetros de los diferentes subprogramas que debes definir.

Aunque en la práctica se dan las cabeceras de las funciones, no se recomienda copiar el código directamente desde el navegador a CodeBlocks ya que se introducen caracteres extraños.

DESCRIPCIÓN DE OPERACIONES

1. Alta de una nota:

Se introduce una nueva nota de asignatura en el sistema (dar un alta). El usuario comenzará por introducir el DNI del estudiante. Pueden darse dos casos:

1. Si ese estudiante no esta dado de alta en el sistema (no está en el fichero de Estudiantes), se pedirá el resto de información del estudiante (Nombre y apellidos) y se creará una nueva entrada en el fichero de estudiantes. Posteriormente se creará un nuevo registro en el fichero de notas de asignaturas.
2. En el caso de que dicho estudiante ya este dado de alta en el sistema, sólo se creará el correspondiente registro con la nota del estudiante y el código de la asignatura, y se insertará en el fichero de notas.

Para implementar esta funcionalidad se recomienda definir los siguientes subprogramas. Para cada subprograma damos su precondición, postcondición y, en algunos casos, algunos comentarios que deben ser tenidos en cuenta a la hora de implementar cada subprograma.

```
tEstudiante completarInformacionEstudiante(char dni[]){ }
{Pre: }
{Post: devuelve un tEstudiante con el DNI recibido como parámetro y el
       resto de información leída por teclado.}

tNotaAsig leerNota(){ }
{Pre: }
{Post: devuelve un tNotaAsig con la información leída por teclado.}

bool existeEstudiante(char nombrefEstudiantes[], char dni[]){ }
{Pre: nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI.}
{Post: devuelve verdad si en el fichero de estudiantes existe un
       estudiante con el dni indicado, y falso en caso contrario.}
{Comentario: este algoritmo debería aprovechar que el fichero está
             ordenado.}

void guardaEstudiante(char nombrefEstudiantes[], tEstudiante e){ }
{Pre: nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI; además,
      el tEstudiante e no está incluido en el fichero.}
{Post: añade, de manera ordenada, el tEstudiante e al fichero
       nombrefEstudiantes.}

void guardaNota(char nombrefNotas[], tNotaAsig n){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura;
      además, la tNotaAsig n no está incluida en el fichero.}
{Post: añade, de manera ordenada, la tNotaAsig n al fichero
       nombrefNotas.}

void altaNota(char nombrefEstudiantes[],char nombrefNotas[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura; y
      nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI.}
{Post: da de alta una nota del modo explicado en el enunciado de este
       apartado.}
```

2. Mostrar información de las asignaturas:

Mostrar la información de notas de una asignatura, a partir de su código y utilizando el siguiente formato.

DNI	Nombre estudiante	Nota

Para implementar esta funcionalidad se recomienda definir los siguientes subprogramas.

```
void buscaEstudiante(char nombrefEstudiantes[],char dni[],tEstudiante& e){}
{Pre: nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI; además,
      en dicho fichero existe un tEstudiante con dni el dni pasado como
      parámetro de la función.}
{Post: almacena en e el registro del tEstudiante almacenado en el fichero
       nombrefEstudiantes con dni el dni pasado como parámetro de
       la acción.}

bool existeAsignatura(char nombrefNotas[], char codigo[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura}
{Post: devuelve verdad si en el fichero de notas existe una
       tNotaAsig con el código indicado, y falso en caso contrario.}
{Comentario: este algoritmo debería aprovechar que el fichero está
             ordenado.}

void formateaNotasAsignatura(char nombrefNotas[],
                             char nombrefEstudiantes[], char codigo[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura;
      nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI; además,
      en el fichero nombrefNotas existe un registro con el código pasado
      como parámetro de la acción.}
{Post: muestra por pantalla la información de notas de una asignatura
       usando el formato indicado.}
{Comentario: este algoritmo puede aprovechar que el fichero de notas está
             ordenado.}

void muestraNotasAsignatura(char nombrefNotas[], char nombrefEstudiantes[])
{ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura;
      nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI.}
{Post: pide un código de asignatura y comprueba si existe algún registro
       con esa nota; en caso de que existir, muestra las notas de todos
       los registros que tienen ese código usando el formato indicado; en
       caso contrario, muestra un mensaje indicando que no existe ningún
       registro con ese código.}
```

3. Modificar nota parcial en la asignatura:

Un profesor desea modificar la nota de la asignatura. En este caso, al profesor se le pedirá el DNI del estudiante, el código de la asignatura y la nota del examen. Dicha información se almacenará en el fichero de notas, modificando la nota del estudiante en esa asignatura. Esta operación sólo se realizará después de haber comprobado que en el fichero de notas de asignaturas existe una entrada con el DNI y el código de asignatura introducido.

Para implementar esta funcionalidad se recomienda definir los siguientes subprogramas.

```
bool existeNotaEstudiante(char nombrefNotas[], char codigo[], char dni[])
{ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura.}
{Post: devuelve verdad si en el fichero de notas existe una
       tNotaAsig con el código y dni indicados, y falso en caso contrario.}
{Comentario: este algoritmo debe aprovechar que el fichero está
             ordenado.}

void modificaNota(char nombrefNotas[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por codAsignatura.}
{Post: pide al usuario el código, dni y nota; comprueba si existe algún
       registro almacenado con el código y dni indicados, en cuyo caso
       modifica dicho registro en el fichero dejando el fichero ordenado;
       en caso de no existir, muestra un mensaje indicando que no existe
       el registro que se quiere modificar.}
```

4. Dar de baja un estudiante:

Un estudiante decide cambiar de Universidad y es necesario borrar sus datos. Se procederá a realizar la baja del estudiante del fichero estudiante y del fichero de notas de asignaturas a partir del DNI del estudiante.

Para implementar esta funcionalidad se recomienda definir los siguientes subprogramas.

```
void eliminaEstudiante(char nombrefEstudiantes[], char dni[]){ }
{Pre: nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI.}
{Post: elimina de dicho fichero el registro con el DNI pasado como
       parámetro. El fichero resultante sigue ordenado. En caso de no
       existir en el fichero ningún registro con el dni pasado
       como parámetro, el fichero no se modifica.}
{Comentario: en el fichero nombrefEstudiantes como mucho hay un estudiante
             con el DNI pasado como parámetro.}

void eliminaNotasEstudiante(char nombrefNotas[], char dni[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por código.}
{Post: elimina de dicho fichero todos los registros tNotaAsig con el
       dni pasado como parámetro. El fichero resultante sigue ordenado.
       En caso de no existir en el fichero ningún registro con el
       dni pasado como parámetro, el fichero no se modifica.}
```

```
{Comentario: en el fichero nombrefNotas puede haber múltiples registros
             con el dni pasado como parámetro.}

void bajaEstudiante(char nombrefNotas[], char nombrefEstudiantes[], char
dni[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por código y
      nombrefEstudiantes es el nombre de un fichero físico existente que
      contiene registros de tipo tEstudiante ordenados por DNI.}
{Post: elimina de los ficheros todos los registros cuyo DNI sea el pasado
       como parámetro. Los fichero siguen ordenados después de haber
       eliminado los registros. }
```

5. Media de asignatura:
Mostrar la nota media obtenida por los estudiantes de una asignatura, a partir del código de la asignatura.

Para implementar esta funcionalidad se recomienda definir el siguiente subprograma.

```
float notaMedia(char nombrefNotas[], char codigo[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por código; además,
      existe al menos un registro con el código pasado como parámetro.}
{Post: a partir del fichero de notas, devuelve la nota media obtenida por
       los estudiantes de la asignatura con el código pasado como
       parámetro.}

void muestraNotaMedia(char nombrefNotas[], char codigo[]){ }
{Pre: nombrefNotas es el nombre de un fichero físico existente que
      contiene registros de tipo tNotaAsig ordenados por código.}
{Post: a partir del fichero de notas, muestra la media de la asignatura
       indicada con código. Si no existe en el fichero ningún registro
       con el código pasado como parámetro, proporciona esa información
       al usuario.}
```

PROBANDO LA APLICACIÓN

Conviene que compruebes dos cosas al ir completando la práctica.

- No se produce ningún error al compilar.
- A medida que implementas las distintas funcionalidades de tu programa es conveniente que pruebes que el programa hace lo que se espera. Para ello puedes hacer las siguientes comprobaciones (puedes hacer otras, pero las indicadas a continuación prueban toda la funcionalidad del sistema):
  1. Ejecuta tu programa, comprueba que se muestra el menú correctamente.
  2. Comprueba que la primera vez que se ejecuta el programa se crean dos ficheros vacíos que servirán para guardar los estudiantes y las notas respectivamente.
  3. Añade a la base de datos las siguientes notas utilizando la opción 1 del menú, en el orden que se indica a continuación:
     a) Código Asignatura: 801G, DNI: 1234567B, Nombre: Maria, Apellido: Lopez, Nota: 4.
     b) Código Asignatura: 801G, DNI: 1111111A, Nombre: Pepe, Apellido: Perez, Nota: 7.

   c) Código Asignatura: 701G, DNI: 1111111A, Nota: 6.
   d) Código Asignatura: 701G, DNI: 1234567B, Nota: 8.
4. Muestra las notas de las asignaturas 701G y 801G utilizando la opción 2 del menú.
5. Utilizando la opción 3 del menú, comprueba que no es posible cambiar la nota del estudiante 22222222G en la asignatura 750K.
6. Utilizando la opción 3 del menú, cambia la nota del estudiante con DNI 1111111A en la asignatura 801G a un 9,5. Utiliza la opción 2 del menú para comprobar que se ha cambiado la nota.
7. Utiliza la opción 4 del menú para dar de baja al estudiante con DNI 1234567B. Utilizando la opción 2 del menú comprueba que este estudiante ya no aparece en las notas de ninguna de las asignaturas.
8. Utilizando la opción 5 del menú obtén la nota media de las asignaturas 701G, 801G.
9. Sal del programa utilizando la opción 6 del menú.

EFICIENCIA

Por último se pide estimar la eficiencia de todos los subalgoritmos definidos en la práctica.

Subalgoritmo	Eficiencia
`leerEstudiante`	
`leerNota`	
`existeEstudiante`	
`guardaEstudiante`	
`guardaNota`	
`altaNota`	
`buscaEstudiante`	
`muestraNotasAsignatura`	
`existeNotaEstudiante`	
`modificaNota`	
`eliminaEstudiante`	
`eliminaNotasEstudiante`	
`bajaEstudiante`	
`notaMedia`	