

## Examen final – 8/9 de junio de 2016

Tiempo disponible: 3 horas

Se pide construir un programa modular que gestione los estudiantes matriculados en un curso universitario, separados por grupos. El programa constará de cuatro módulos: *Estudiante*, *ListaEstudiantes*, *ListaGrupos* y módulo principal (*main.cpp*).

### **Módulo *Estudiante*** (0.75 puntos)

Declara un tipo de estructura `tEstudiante` con cuatro campos: nombre completo, NIF, fecha de matrícula y nota (todos de tipo cadena de caracteres salvo la nota, de tipo real). Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga la información de un estudiante. En el archivo los datos de un estudiante están organizados en dos líneas (ver el ejemplo de archivo al final del enunciado):

```
Nombre y apellidos // Primera línea: el nombre completo (con espacios)
12345678X 14/06/01 6.5 // NIF, fecha (en formato AA/MM/DD) y nota
```

- ✓ `mostrar()`: Dado un estudiante muestra su información en la pantalla con el siguiente formato (NIF - fecha - nota - nombre completo):

```
12345678X - 14/06/01 - 6.5 - nombre y apellidos
```

### **Módulo *ListaEstudiantes*** (4 puntos)

Declara un tipo `tListaEstudiantes` para listas de estudiantes **implementada con array dinámico y ordenada por NIF**.

El módulo implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga la información de una lista de estudiantes. En el archivo primero aparece en una línea el número de estudiantes, y luego dos líneas por estudiante (ver el ejemplo de archivo al final del enunciado). En el archivo los estudiantes no están ordenados por el NIF (hay que insertarlos ordenadamente en la lista).
- ✓ `buscar()`: Dada una lista y un NIF, si el estudiante se encuentra en la lista, devuelve cierto y la posición del estudiante en la lista. Si no, devuelve falso. Implementa el algoritmo de **búsqueda binaria**.
- ✓ `insertar()`: Dada una lista de estudiantes y un estudiante añade ordenadamente el estudiante a la lista (no es necesario redimensionar la lista).

- ✓ `mostrar()`: Dada una lista muestra en la pantalla cada uno de los estudiantes de la lista (ver el ejemplo de ejecución al final del enunciado).
- ✓ `destruir()`: Dada una lista de estudiantes libera la memoria dinámica que usa.

### Módulo *ListaGrupos* (4,25 puntos)

Máx. 10 grupos

Declara un tipo de estructura `tGrupo` con dos campos: identificador (cadena de caracteres sin espacios) y lista de estudiantes matriculados. Declara también un tipo `tListaGrupos` para listas de grupos (hasta 10) **implementadas con array estático de punteros a datos dinámicos** (es decir, un array de punteros a `tGrupo`). La lista no está ordenada.

El módulo implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga la lista de grupos del archivo `notas.txt`. Para cada grupo, en el archivo primero aparece en una línea el identificador del grupo, y a continuación su lista de estudiantes. El archivo termina con el centinela "XXX".
- ✓ `mostrar()`: Dada una lista de grupos muestra por pantalla cada uno de los grupos, separados por guiones, como aparece en el ejemplo de ejecución al final del enunciado.
- ✓ `buscar()`: Dada una lista, un NIF y una posición, devuelve cierto si el NIF aparece en uno de los grupo de la lista, a partir de la posición dada. Si es así devuelve el grupo (posición o identificador) en el que se encuentra el estudiante. Si no se hace de forma **recursiva**, se perderán 0,5 puntos.
- ✓ `destruir()`: Dada una lista de grupos, libera la memoria dinámica que usa.

### Módulo principal (1 punto)

Carga la información del archivo `notas.txt` en una lista de grupos, muestra la información cargada, pide al usuario un NIF y muestra el identificador del grupo de dicho alumno (si existe), y antes de terminar libera la memoria dinámica utilizada.

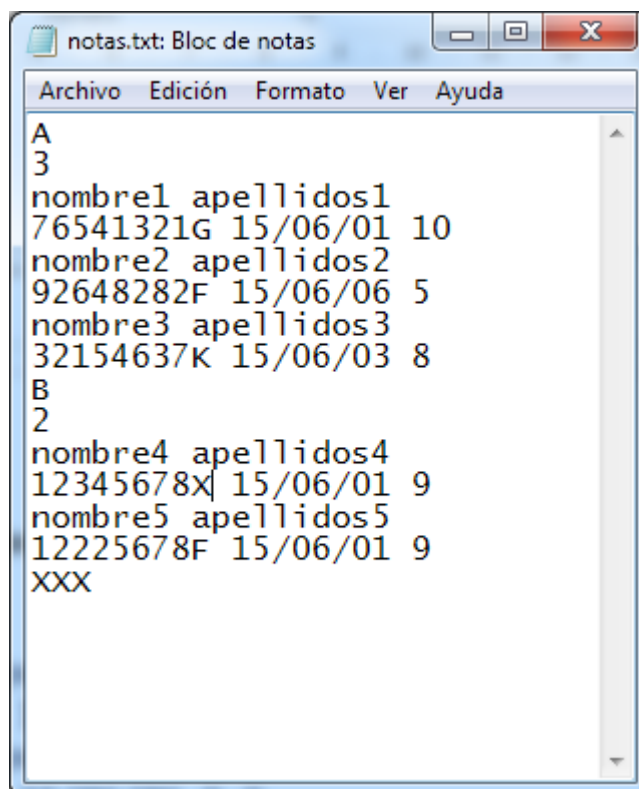
Se valorará la legibilidad, así como el uso adecuado de los esquemas de recorrido y búsqueda, de la comunicación entre subprogramas y de la memoria.

**Recuerda:** El comando para que se muestre la memoria no liberada es

```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

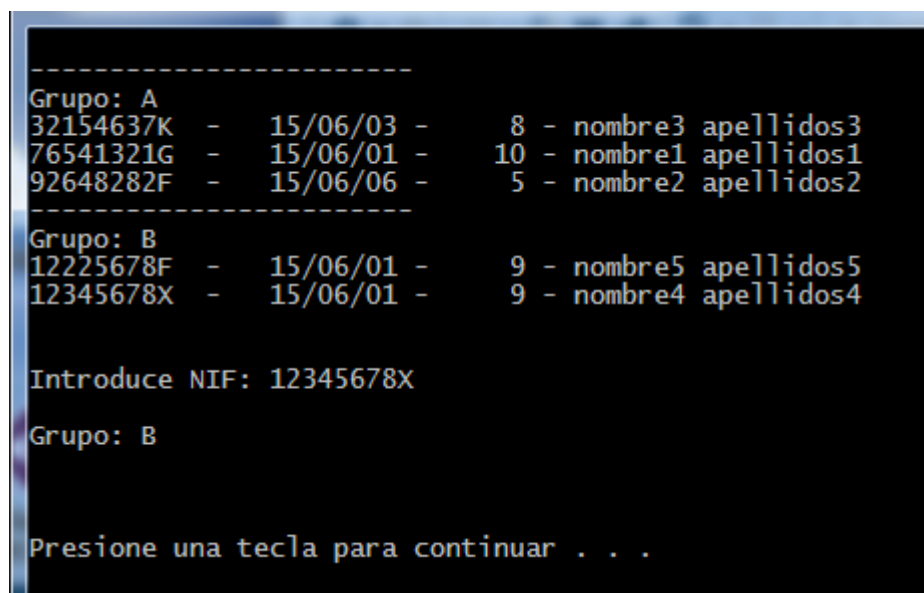
**Entrega** el código del programa **a través del Campus Virtual** (sólo `.cpp` y `.h`, comprimidos en un ZIP). ¡Asegúrate de entregar una versión sin errores de compilación!

## Ejemplo de archivo notas.txt:



```
notas.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
A
3
nombre1 apellidos1
76541321G 15/06/01 10
nombre2 apellidos2
92648282F 15/06/06 5
nombre3 apellidos3
32154637K 15/06/03 8
B
2
nombre4 apellidos4
12345678X 15/06/01 9
nombre5 apellidos5
12225678F 15/06/01 9
XXX
```

## Ejemplo de ejecución:



```
-----
Grupo: A
32154637K - 15/06/03 - 8 - nombre3 apellidos3
76541321G - 15/06/01 - 10 - nombre1 apellidos1
92648282F - 15/06/06 - 5 - nombre2 apellidos2
-----
Grupo: B
12225678F - 15/06/01 - 9 - nombre5 apellidos5
12345678X - 15/06/01 - 9 - nombre4 apellidos4

Introduce NIF: 12345678X
Grupo: B

Presione una tecla para continuar . . .
```

## Archivo checkML.h

```
#ifdef _DEBUG
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif
```

