1. Ejercicios propuestos

Los ejercicios propuestos para realizar durante esta segunda semana de docencia *online* son:

Ejercicio 1

Este ejercicio consiste en escribir una colección de funciones básicas para poder manejar una colección de registros, de tipo `Estudiante` (se entrega la definición del tipo de dato en Listado 2), de manera que permita realizar las operaciones más básicas:

- Creación de la colección, se tratará de dos tipos de colecciones: mediante un vector de registros (ver Figura 1) y mediante un vector de punteros a registro (ver Figura 2), tal y como se ha estudiado en la segunda lección.
- Carga de los registros de las colecciones (vector de registros, colección mediante vector de punteros a registros) con valores aleatorios.
- Liberación de la memoria reservada por ambas colecciones.

Las imágenes Figura 1 y Figura 2 representan las dos colecciones, con las dos variables, `regEstudiantes` y `regEstudiantesRef`, declaradas en el código de la función `main()` de Listado 1.

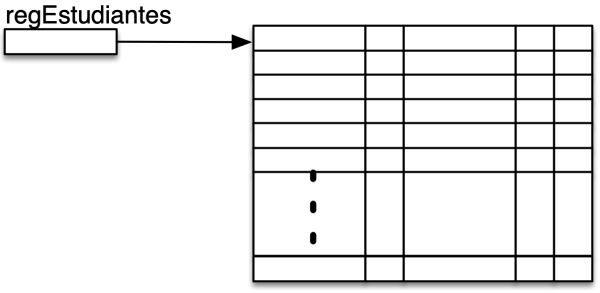

Figura 1: Colección de registros mediante vector de registros.

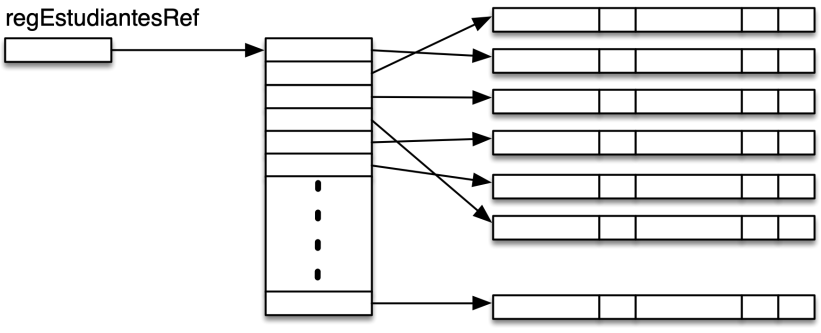

Figura 2: Colección de registros mediante vector de punteros a registros.

Se debe escribir el código correspondiente a las funciones que se describen a continuación. El proyecto constará de tres ficheros. El fichero `main.c` básico queda como sigue:

Listado 1: Fichero `main.c` con la función `main()`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <time.h>
#include "estudiante.h"

int main(void)
{
  Estudiante *regEstudiantes;
  Estudiante **regEstudiantesRef;
  int  numEstudiantes,errNum;

  srandom(time(NULL));

  printf("\nNúmero de estudiantes: ");
  scanf("%d%*c", &numEstudiantes);

  printf("\nEl tamaño de un registro es: %ld",sizeof(Estudiante));

  regEstudiantes = crearVectorRegistros(numEstudiantes, &errNum);
  if (!cargarRegistrosAleatorios(regEstudiantes, numEstudiantes)) {
    mostrarRegistros(regEstudiantes, numEstudiantes);
  }

  regEstudiantesRef = crearVectorRegistrosRef(numEstudiantes, &errNum);
  if(!cargarRegistrosAleatoriosRef(regEstudiantesRef, numEstudiantes)){
    mostrarRegistrosRef(regEstudiantesRef, numEstudiantes);
  }

  free(regEstudiantes);
  regEstudiantes = NULL;
  if (!liberarVectorRegistrosRef(regEstudiantesRef, numEstudiantes))
    regEstudiantesRef = NULL;

  return 0;
}
```

El fichero `estudiante.h` contendrá el tipo de dato y los prototipos de las funciones:

Listado 2: Fichero de cabecera `estudiante.h` con definición de tipo, constantes y prototipos de funciones.

```c
#ifndef __ESTUDIANTE_H
#define __ESTUDIANTE_H

#define NUM_ASIGNATURAS 10

typedef struct tipoEstudiante {
    char apellidos[30];
    char nombre[15];
    float calificaciones[NUM_ASIGNATURAS];
} Estudiante;

Estudiante *crearVectorRegistros(int num, int *codError);
```

```
13 int cargarRegistrosAleatorios(Estudiante * estudiantes, int num);
14
15 Estudiante **crearVectorRegistrosRef(int num, int *codError);
16 int liberarVectorRegistrosRef(Estudiante **estudiantesRef, int num);
17 int cargarRegistrosAleatoriosRef(Estudiante **estudiantesRef, int num);
18
19 int cargarUnRegistroAleatorio(Estudiante * unEstudiante);
20
21 void mostrarUnRegistro(Estudiante unEstudiante);
22 void mostrarRegistrosRef(Estudiante ** estudiantesRef, int num);
23 void mostrarRegistros(Estudiante * estudiantes, int num);
24 #endif
```

El estudiante debe escribir, en este primer ejercicio y en el fichero `estudiante.c`, todas las funciones, cuya descripción se hará a continuación, salvo un par de ellas, cuyo código se proporciona en Listado 3 y en Listado 4. Por último, también debe escribir el correspondiente fichero `Makefile` para generar correctamente el proyecto.

Listado 3: Carga de un registro tipo `Estudiante` con valores aleatorios.

```
1  int cargarUnRegistroAleatorio(Estudiante * unEstudiante)
2  {
3    static char apellidos [][15] = {"ALONSO","ALVAREZ","ARBESU",
4      "DOMINGUEZ","FERNANDEZ","FLORIANO", "GONZALEZ","GOMEZ",
5      "GUTIERREZ","MARTIN", "MORO", "PEREZ","TURRION","ZAMBRANO"};
6    static char nombres [][15] = {"ALVARO","ARSENIO","DOMINGO",
7      "FELIPE","FATIMA","FABIAN", "GONZALO","PEDRO","PATRICIA",
8      "TOMAS", "ZOILO"};
9    int numApellidos = sizeof(apellidos)/15;
10   int numNombres = sizeof(nombres)/15;
11   int i;
12   if (unEstudiante == NULL) {
13 #ifdef DEBUG
14     fprintf(stderr, "Error: registro no válido!\n");
15 #endif
16     return -1;
17   }
18   sprintf(unEstudiante->apellidos,"%s %s",
19     apellidos[random() % numApellidos],
20     apellidos[random() % numApellidos]);
21   sprintf(unEstudiante->nombre,"%s",nombres[random() % numNombres]);
22   for (i = 0; i < NUM_ASIGNATURAS; i++){
23     unEstudiante->calificaciones[i] = (random() % 100 + 1)/10.0f;
24   }
25   return 0;
26 }
```

El fichero `vectorDinamica.c` contiene la función que crea un vector de enteros utilizando memoria dinámica:

Creación del vector de registros con memoria dinámica

```
Estudiante *crearVectorRegistros(int num, int *codError);
```

Esta función crea, de forma dinámica, un vector de registros de tipo `Estudiante` y de tamaño `num` y devuelve la dirección de memoria del primer *byte* del bloque reservado si la función tiene éxito, y `NULL` en caso contrario. La estrategia seguida es la que se muestra en la Figura 1. Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- −1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- −2 Error. Se ha producido un error en la invocación a la función de reserva dinámica de memoria.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del bloque reservado.

Carga de un vector de registros con valores aleatorios

```
int cargarRegistrosAleatorios(Estudiante * estudiantes, int num);
```

Esta función carga los elementos del vector de registros referenciado por el parámetro `estudiantes` con valores aleatorios, y de tamaño `num`, devolviendo un valor entero que indica si ha tenido o no, según los siguientes criterios:

- −1 Si el valor del parámetro vector no es válido (referencia `NULL`).
- 0 Si se ejecuta correctamente.

Para poder realizar esta tarea correctamente, el alumno implementará un bucle que recorrerá el vector de registros recibido a través del parámetro `estudiantes` y utilizará, en cada iteración, la invocación de la función `cargarUnRegistroAleatorio()` que se proporciona en Listado 3.

Creación del vector de punteros a registro con memoria dinámica

```
Estudiante **crearVectorRegistrosRef(int num, int *codError);
```

Esta función crea, de forma dinámica, un vector de punteros a registro de tipo `Estudiante` y de tamaño `num`, y a continuación reservará el mismo número de bloques de memoria de tamaño registro `Estudiante`, almacenando sus referencias en las celdas del vector de punteros. La función debe devolver la dirección de memoria del primer *byte* del bloque de punteros reservado si tiene éxito, y `NULL` en caso contrario. La estrategia seguida es la que se muestra en la Figura 2, el bloque de punteros almacena las direcciones al primer *byte* de cada registro reservado. Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- −1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- −2 Error. Se ha producido un error en la invocación a la función de reserva dinámica de memoria del bloque de punteros a registro.
- −3 Error. Se ha producido un error en la invocación a la función de reserva dinámica de memoria para alguno de los bloques que almacenará un registro del tipo `Estudiante`, y cuyas referencias se almacenan en el bloque de punteros. En caso de que se produzca este error se debe liberar la memoria reservada con éxito para todos los bloques de los registros anteriores, además del propio bloque de punteros a registro, devolviendo un `NULL`.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del bloque de punteros a registro reservado.

Liberación de bloques de memoria dinámica reservada para los registros y el vector de punteros a registro

```
int liberarVectorRegistrosRef(Estudiante **estudiantesRef, int num);
```

Esta función libera toda la memora reservada, según la estrategia de la Figura 2, primero los bloques de memoria que almacenan los registros, y después el bloque de memoria que almacena los punteros a registro. La función devolverá un valor entero indicando:

- −1 Error. Si el valor del parámetro `estudiantesRef` es incorrecto (referencia `NULL`).

- 0 Éxito. Libera correctamente toda la memoria reservada

Carga los registros referenciados por punteros del vector de punteros con valores aleatorios

```
int cargarRegistrosAleatoriosRef(Estudiante **estudiantesRef, int num);
```

Esta función carga los registros referenciados por los punteros almacenados en el bloque de punteros, de tamaño `num`, y referenciado por el parámetro `estudiantesRef` con valores aleatorios devolviendo un valor entero que indica si ha tenido o no, según los siguientes criterios:

- -1 Error. Si el valor del parámetro `estudiantesRef` es incorrecto (referencia `NULL`).
- 0 Éxito. Si se ejecuta correctamente.

Para poder realizar esta tarea correctamente, el alumno implementará un bucle que recorrerá el vector de punteros a registros recibido a través del parámetro `estudiantesRef` y utilizará, en cada iteración, la invocación de la función `cargarUnRegistroAleatorio()` que se proporciona en Listado 3 para poder cargar el registro referenciado por dicho puntero con valores aleatorios.

Mostrar el contenido de los registros almacenados en un vector de registros

```
void mostrarRegistros(Estudiante * estudiantes, int num);
```

Esta función muestra por pantalla el contenido de los registros almacenados en el vector de registros referenciado por el parámetro `estudiantes`.

Para poder realizar esta tarea correctamente, el alumno implementará un bucle que recorrerá el vector de registros recibido a través del parámetro `estudiantes` y utilizará, en cada iteración, la invocación de la función `mostrarUnRegistro()`, que se proporciona en Listado 4.

Mostrar el contenido de los registros referenciados por un vector de punteros a registros

```
void mostrarRegistrosRef(Estudiante ** estudiantesRef, int num);
```

Esta función muestra por pantalla el contenido de los registros referenciados por los punteros almacenados en el vector de punteros `estudiantesRef`.

Para poder realizar esta tarea correctamente, el alumno implementará un bucle que recorrerá el vector de punteros a registros recibido a través del parámetro `estudiantesRef` y utilizará, en cada iteración, la invocación de la función `mostrarUnRegistro()`, que se proporciona en Listado 4.

Listado 4: Función que muestra por consola el contenido de un registro tipo `Estudiante` pasado por valor.

```
1  void mostrarUnRegistro(Estudiante unEstudiante)
2  {
3    int i;
4    float media, suma;
5    printf("%s, %s\n", unEstudiante.apellidos, unEstudiante.nombre);
6    suma = 0.0f;
7    for (i = 0; i < NUM_ASIGNATURAS; i++){
8      printf("%3.1f\t", unEstudiante.calificaciones[i]);
9      suma += unEstudiante.calificaciones[i];
10   }
11   media = suma/NUM_ASIGNATURAS;
12   printf("\nCalificación media: %3.1f\n\n", media);
13 }
```

Ejercicio 2

Vamos a añadir dos funciones a la biblioteca. La primera servirá para obtener una colección de registros representada mediante un vector de registros (Figura 1) a partir de un vector de punteros a registro (Figura 2). La segunda hará justo la operación inversa, proporcionará un colección de registros representada mediante un vector de punteros a registros (Figura 2) a partir de un un vector de registros (Figura 1).

Adicionalmente, el estudiante debe añadir las líneas necesarias en la función `main()` del Listado 1 que permitan comprobar el correcto funcionamiento de las funciones implementadas.

Devolver un vector de registros con el contenido del vector de punteros a registros de entrada

```
Estudiante * pasarRegistrosAVector(Estudiante ** estudiantesRef, int num, int *codError);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector de punteros a registro (Figura 2) y devolverá una copia de dicha colección representada mediante un vector de registros (Figura 1). Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- -1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- -2 Error. Se ha producido un error en la invocación a la función de reserva dinámica de memoria.
- -10 Error. Si el parámetro `estudiantesRef` es un puntero `NULL`.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del bloque reservado.

Si para realizar las reservas de memoria del vector de registros se utiliza la función `crearVectorRegistros()`, los códigos de error -1 y -2 coinciden con las definiciones de error descritas.

Devolver un vector de punteros a registros con el contenido del vector de registros de entrada

```
Estudiante ** pasarVectorARegistros(Estudiante * estudiantes, int num, int *codError);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector registros (Figura 1) y devolverá una copia de dicha colección representada mediante un vector de punteros a registros (Figura 2). Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- -1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- -2 Error. Se ha producido un error en la invocación a la función de reserva dinámica de memoria.
- -10 Error. Si el parámetro `estudiantesRef` es un puntero `NULL`.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del bloque reservado.

Si para realizar las reservas de memoria del vector de registros se utiliza la función `crearVectorRegistrosRef()`, los códigos de error -1 y -2 coinciden con las definiciones de error descritas.

Ejercicio 3

En este ejercicio se propone añadir dos nuevas funciones que utilizaremos en otras funciones. Creará un vector de tipo `float`, mediante reserva dinámica de memoria, y del mismo tamaño que la colección. Recorrerá la colección y, para cada alumno, calculará la calificación promedio y almacenará el resultado en la correspondiente celda del vector creado, según el mismo índice del alumno dentro de la colección. Esta calificación promedio se obtiene promediando las calificaciones almacenadas en el campo `calificaciones` del registro.

Adicionalmente, el estudiante debe añadir las líneas necesarias en la función `main()` del Listado 1 que permitan comprobar el correcto funcionamiento de las funciones implementadas.

Devolver un vector con las calificaciones promedio de los registros `Estudiante` del vector de registros

```
float * devolverVectorPromedios(Estudiante * estudiantes, int num);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector registros (Figura 1) y devolverá un vector, de tipo `float`, creado de forma dinámica y del mismo tamaño que la colección, las calificaciones promedio de cada alumnos. En la primera celda de dicho vector se guardará el valor de la calificación promedio del primer registro de la colección, en la segunda celda del vector de guardará el valor de la calificación promedio del segundo registro, y así sucesivamente. La función devolverá:

- Éxito. La dirección de memoria del primer *byte* del bloque de *bytes* que representa el vector creado dinámicamente con los valores promedio de las calificaciones de cada uno de los registros.
- Error. Puntero `NULL`. Si el parámetro `estudiantes` es un puntero `NULL` o si falla la reserva de memoria del vector que se debe devolver.

Devolver un vector con las calificaciones promedio de los registros `Estudiante` del vector de punteros a registros

```
float * devolverVectorPromediosRef(Estudiante ** estudiantesRef, int num);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector de punteros a registros (Figura 2) y devolverá un vector, de tipo `float`, creado de forma dinámica y del mismo tamaño que la colección, las calificaciones promedio de cada alumnos. En la primera celda de dicho vector se guardará el valor de la calificación promedio del primer registro de la colección, en la segunda celda del vector de guardará el valor de la calificación promedio del segundo registro, y así sucesivamente. La función devolverá:

- Éxito. La dirección de memoria del primer *byte* del bloque de *bytes* que representa el vector creado dinámicamente con los valores promedio de las calificaciones de cada uno de los registros.
- Error. Puntero `NULL`. Si el parámetro `estudiantesRef` es un puntero `NULL` o si falla la reserva de memoria del vector que se debe devolver.

Ejercicio 4

En este ejercicio se proponen dos nuevas funciones que, haciendo uso de las funciones del **Enunciado 3**, deben devolver un nuevo vector de registros (Figura 1) conteniendo todos los registros de la colección de entrada cuya calificación promedio esté contenida entre dos valores de calificación.

Adicionalmente, el estudiante debe añadir las líneas necesarias en la función `main()` del Listado 1 que permitan comprobar el correcto funcionamiento de las funciones implementadas.

Devolver la colección de registros cuyas calificaciones promedio están dentro de un rango. Entrada: vector de registros

```
Estudiante * devolverVectorRegistrosRango(Estudiante * estudiantes, int num,
                                          float bajo, float alto, int *codError);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector registros (Figura 1), parámetro `estudiantes`, y devolverá otro vector de registros (Figura 1) que contendrá únicamente aquellos del vector de entrada cuya calificación promedio está dentro del rango definido por los parámetros `bajo` y `alto`, ambos inclusive. Para la realización de esta tarea se utilizará la función `devolverVectorPromedios()` que permitirá, recorriendo el vector proporcionado por la misma, contar el número de registros que cumplen la condición. Una vez conocido este dato, se procederá a crear un nuevo vector de registros, mediante el uso de la función `crearVectorRegistros()`, para crear el vector que se debe devolver. A continuación, y haciendo uso del vector de calificaciones promedio ya citado, se copiarán los registros que cumplan la condición definida del vector de entrada al vector de salida. Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- -10 Error. Si el parámetro `estudiantes` es un puntero `NULL`.
- -1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- -2 Error. Se ha producido un error en las reservas dinámicas de memoria, sea del vector de promedios, sea del vector de registros.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del vector de registros creado.

Los códigos de error -1 y -2 coinciden con los de la función `crearVectorRegistros()`. Importante, se supone que se debe cumplir la condición `bajo` $\leq$ `alto`. Si no se cumpliese esta condición se debe proceder a un intercambio de los valores pasados a través de dichos parámetros para que, antes de que la función se ponga a realizar la tarea descrita, si la cumpla.

Devolver la colección de registros cuyas calificaciones promedio están dentro de un rango. Entrada: vector de punteros a registros

```
Estudiante * devolverVectorRegistrosRangoRef(Estudiante ** estudiantesRef, int num,
                                             float bajo, float alto, int *codError);
```

Esta función recibe una colección de registros de tipo `Estudiante` representada mediante un vector punteros a registros (Figura 2), parámetro `estudiantesRef`, y devolverá un vector de registros (Figura 1) conteniendo únicamente aquellos del vector de entrada cuya calificación promedio está dentro del rango definido por los parámetros `bajo` y `alto`, ambos inclusive. Para la realización de esta tarea se utilizará la función `devolverVectorPromedios()` que permitirá, recorriendo el vector proporcionado por la misma, contar el número de registros que cumplen la condición. Una vez conocido este dato, se procederá a crear un nuevo vector de registros, mediante el uso de la función `crearVectorRegistros()`, para crear el vector que se debe devolver. A continuación, y haciendo uso del vector de calificaciones promedio ya citado, se copiarán los registros que cumplan la condición definida del vector de entrada al vector de salida. Para controlar el correcto funcionamiento se incluye, además, el parámetro `codError` que se utiliza para devolver un valor entero (paso por referencia) indicando:

- -10 Error. Si el parámetro `estudiantesRef` es un puntero `NULL`.
- -1 Error. Si el valor del parámetro `num` es incorrecto (menor o igual que cero).
- -2 Error. Se ha producido un error en las reservas dinámicas de memoria, sea del vector de promedios, sea del vector de registros.
- 0 Éxito. Además la función devuelve, como ya se ha indicado anteriormente, la dirección de memoria del primer *byte* del vector de registros creado.

Los códigos de error -1 y -2 coinciden con los de la función `crearVectorRegistros()`. Importante, se supone que se debe cumplir la condición `bajo` $\leq$ `alto`. Si no se cumpliese esta condición se debe proceder a un intercambio de los valores pasados a través de dichos parámetros para que, antes de que la función se ponga a realizar la tarea descrita, si la cumpla.