Ejercicios propuestos

En este caso se debe partir de los tres ejercicios resueltos, cuyos proyectos están disponibles en el **Tema 3. Memoria dinámica** de la página de la asignatura, en Studium. El enlace que se debe utilizar para descargarlo lleva por título "Más ejercicios resueltos sobre memoria dinámica". Los tres ejercicios resueltos se encuentran tres carpetas diferentes, que se describen a continuación, y que contienen código que implementa las tres estrategias explicadas en la **Lección 3** para la creación y procesamiento de matrices bidimensionales utilizando memoria dinámica:

- `dinamicaMatrices`: código fuente de un proyecto donde las matrices bidimensionales se implementan usando un bloque de *bytes* que está referenciado por un puntero del tipo base de los elementos a almacenar (`tipoBase *pt;` donde tipoBase es `float` o `int`).
- `dinamicaMatricesBidimensionales`: código fuente de un proyecto análogo al anterior (el `main` es idéntico y las funciones tienen el mismo prototipo), donde las matrices bidimensionales se implementan como un único bloque de bytes referenciado por un puntero de tipo fila o vector de tipoBase (`tipoBase (*pt)[COLS];`, donde la constante `COLS` es el tamaño de la fila y tipoBase vuelve a ser `float` o `int`).
- `dinamicaMatricesPunteros`: código fuente de un proyecto análogo a los anteriores, donde las matrices bidimensionales se implementan como un único bloque de bytes referenciado por un puntero a puntero de tipoBase que contiene los punteros a los bloques de memoria (reservados dinámicamente también) que representan las filas de la matriz bidimensional (`tipoBase **pt;`, con tipoBase `float` o `int`)

En estos proyectos la única función de procesamiento de matrices bidimensionales implementada es la de sumar dos matrices. El código se explica en la **Lección 3** disponible en el **Tema 6** de la página de la asignatura de Studium.

Ahora se propone al alumno que añada las siguientes funciones. Además de implementar las funciones se añadirá el código necesario, en la función `main()` de cada proyecto, para poder probar el funcionamiento de las mismas.

Ejercicio 1

Multiplicación de matrices bidimensionales

```
matFloatRef multiplicarMatFloat(matFloatRef a, matFloatRef b, intRef errNum);
```

Para cada uno de los tres proyectos se debe añadir esta función que realizará la misma tarea en los tres casos. Recibe dos matrices (en cada caso del tipo correspondiente al proyecto en cuestión) y devuelve el resultado del producto de ambas matrices. La matriz resultado debe ser creada dinámicamente y de las dimensiones adecuadas. Se puede seguir el esquema que se presenta en la función `sumarMatFloat()` entregada. El tercer parámetro, `errNum`, es un valor entero que se pasa por referencia y que sirve para indicar si se ha tenido éxito o no en la tarea, según los siguientes criterios:

- Valores negativos de -1 a -4. Error controlado por la invocación a las funciones `fallaMatrizFloat()` (se usará para comprobar la validez de `a` y `b`), y que indica fallo en los punteros o en las dimensiones almacenadas en la estructura, y `crearMatFloat()` que indica fallo en las reservas de memoria al crear la matriz resultado.
- -5 Error. El número de columnas de la matriz `a` no coincide con el número de filas de la matriz `b`, por lo que no se puede realizar el producto. La función devuelve `NULL`.
- 0 Éxito. Todos los parámetros son correctos, se ha reservado memoria con éxito para la matriz resultado y se ha realizado el producto de ambas matrices correctamente. Se devuelve la dirección de memoria al primer *byte* del bloque de memoria que almacena la estructura que representa la matriz resultado de la operación.

Nota importante: obsérvese que no se distingue entre los tres casos porque los prototipos de las funciones son idénticos en los tres proyectos, lo que cambia es la definición de los tipos `matFloat` y `matFloatRef` que representan la estructura que soporta la matriz y el puntero a dicha estructura respectivamente.

Al ser tres proyectos, uno por cada tipo de estrategia, y al ser dos funciones en cada proyecto, una para matrices de valores de tipo `float` y otra para matrices de valores de tipo int, son seis funciones en total a implementar.

Ejercicio 2

Obtención de la columna que contiene el valor máximo de la matriz bidimensional

```
float *obtenerColumnaMaxMatFloat(matFloatRef mat, intRef errNum);
```

Esta función recibe una matriz bidimensional, referenciada por el parámetro `mat`, ya cargada con valores en sus elementos y buscará, recorriendo dicha matriz bidimensional por columnas, el valor máximo de los elementos de la misma, y devolverá en un vector de valores `float` (en este caso, o de `int` para matrices del tipo `matIntRef`) que se debe crear dinámicamente, la columna que contiene ese valor máximo, es decir, debe copiar en dicho vector todos los valores almacenados la columna localizada. En el caso de que existan varios valores iguales al máximo, deberá devolver la columna correspondiente al último encontrado. El segundo parámetro es un valor entero que se pasa por referencia para devolver un código de error según los siguientes criterios:

- Valores negativos de -1 a -4. Error controlado por la invocación a las funciones `fallaMatrizFloat()` (se usará para comprobar la validez de `mat`), y que indica fallo en los punteros o en las dimensiones almacenadas en la estructura. El valor de retorno de la función será `NULL`.
- -5 Error en la reserva de memoria dinámica del vector que almacena los elementos de la columna localizada. El valor de retorno de la función será `NULL`.
- 0 Éxito. Tarea completada con éxito, la función devuelve la dirección de memoria al primer *byte* del bloque de memoria que representa el vector solución.

Nota importante: obsérvese que no se distingue entre los tres casos porque los prototipos de las funciones son idénticos en los tres proyectos, lo que cambia es la definición de los tipos `matFloat` y `matFloatRef` que representan la estructura que soporta la matriz y el puntero a dicha estructura respectivamente.

Al ser tres proyectos, uno por cada tipo de estrategia, y al ser dos funciones en cada proyecto, una para matrices de valores de tipo `float` y otra para matrices de valores de tipo int, son seis funciones en total a implementar.