

PRÁCTICA 1ª: Creación y utilización de matrices dinámicas.

OBJETIVOS: Repaso de conceptos de programación vistos en cursos anteriores, especialmente punteros y asignación dinámica de memoria.

TEMPORIZACIÓN:

Publicación del enunciado: Semana del 13 de septiembre.

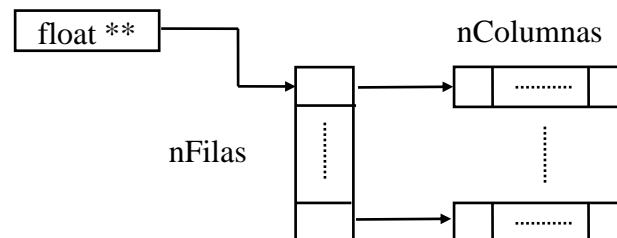
Entrega: Semana del 4 de octubre junto con la práctica 2.

Límite de entrega (con penalización): Semana del 18 de octubre.

Se deberán realizar cuatro funciones con los siguientes prototipos:

```
float **ConstruirMatriz(int nFilas, int nColumnas);
```

Esta función creará una matriz dinámica, iniciada a cero, cuyas dimensiones vendrán dadas por los parámetros de tipo entero que se le pasan. Para ello hará uso de la asignación dinámica de memoria. Verificará que la asignación de memoria fue correcta e iniciará la matriz a cero. Si durante el proceso de asignación de memoria ocurre un error porque no hay memoria suficiente, se liberará la memoria asignada hasta entonces. Devolverá un puntero a puntero a float o NULL si ocurre un error. Mediante este puntero se accederá a una estructura de datos que representará a la matriz.



```
void IntroducirDatos(MatFloat *pDestino);
```

Esta función pedirá datos por pantalla y los introducirá en la matriz recibida (previamente creada con la función anterior). Validará la entrada de datos para garantizar que son del tipo pedido. Como único parámetro se le pasará por referencia una variable del tipo MatFloat. Este tipo tiene la siguiente definición:

```
struct MatFloat
{
    int nFilas;           // Número de filas
    int nColumnas;       // Número de columnas
    float **ppMatrizF;    // Datos de la matriz
};
```

Una variable de este tipo almacena en su campo nFilas el número de filas, en su campo nColumnas el número de columnas y en su campo ppMatrizF la dirección de comienzo de la matriz dinámica; un valor NULL para ppMatrizF significa que no existe la matriz.

Es decir, una variable de este tipo almacena toda la información necesaria para la creación y utilización de una matriz dinámica.

```
void Mostrar(MatFloat mOrigen);
```

Esta función volcará los datos de una matriz dinámica en la pantalla. La matriz se mostrará por filas (una debajo de otra). Como único parámetro se le pasará por valor una estructura del tipo `MatFloat` asociada a la matriz.

```
void Destruir(MatFloat *pMatFloat);
```

Esta función destruirá la matriz dinámica liberando la memoria asignada dinámicamente. También pondrá `ppMatrizF` a `NULL`, indicativo de que la matriz no existe.

```
int main();
```

Esta función mostrará el siguiente menú:

1. Construir matriz
2. Introducir matriz
3. Volcar matriz
4. Destruir matriz
5. Terminar

y permitirá ejecutar cada una de las opciones utilizando las funciones anteriores y, al terminar el programa, verificará si hay o no lagunas de memoria indicándolo con un mensaje.

La aplicación estará compuesta, al menos, por los archivos `funciones.h` y `funciones.cpp` que contendrán las declaraciones y definiciones, respectivamente, de las funciones especificadas (excepto `main`), por el archivo `práctical.cpp` que contendrá la definición de la función `main` y por los archivos `MemoryManager` indicados un poco más adelante.

Todas las opciones del menú tienen que poder ser ejecutadas en cualquier orden y repetidas veces. Cada opción del menú deberá invocar a una función que realice el proceso requerido si ésta existe. Un valor incorrecto será rechazado volviendo a requerir otro valor. Esta forma de trabajo será la que apliquemos a cada una de las prácticas restantes.

Deberá probarse el programa con matrices cuadradas, matrices con más filas que columnas, matrices con más columnas que filas, valores para filas/columnas cero o negativos, de un tipo diferente al solicitado; si ya existe una matriz, no se podrá crear otra mientras no se destruya la actual; no se podrá introducir datos si no hay una matriz creada; al terminar, si hay una matriz creada se destruirá, etc.

Observar que, en algunas de las funciones utilizadas, la estructura representativa de la matriz se pasa por referencia. ¿Podría ser pasada por valor en todos los casos?

REALIZAR otra versión del programa utilizando el tipo `vector<T>`. Elimine las funciones y estructuras que no sean necesarias al utilizar dicho tipo. Por ejemplo, el método `size()` de `vector<T>` invita a prescindir de la estructura `MatFloat`. Esta otra versión contendrá, al menos, las funciones:

```
vector<vector<float>>> ConstruirMatriz(int nFilas, int nColumnas);  
void IntroducirDatos(vector<vector<float>>& v);  
void Mostrar(vector<vector<float>>& v);  
int main();
```

NOTA: En esta práctica se permitirá el uso de los mecanismos de C para entrada/salida y asignación/liberación dinámica de memoria: `printf`, `scanf`, `malloc`, `free`..., pero, en las siguientes prácticas se deberá usar los mecanismos propios de C++: `cin`, `cout`, `new`, `delete`...

NOTA: Para que el programa detecte las lagunas de memoria (memoria dinámica reservada y no liberada), se debe incluir el fichero `MemoryManager.h` al comienzo de cada archivo con extensión `.cpp` que haga reserva dinámica de memoria (`malloc`, `new`, etc.) y añadir el fichero `MemoryManager-vc201x-d.obj` al construir el ejecutable si está utilizando el EDI Microsoft Visual C++, o bien el fichero `MemoryManager_GNU_GCC.o` si está utilizando un compilador GCC (*GNU Compiler Collection*), por ejemplo con el EDI Code::Blocks. Estos ficheros pueden obtenerse en la página web de la asignatura y deben copiarse, los que se vayan a utilizar, en la carpeta del proyecto. A continuación, proceda de la forma siguiente: añadir el fichero `.obj` al proyecto que ha generado.

Ejemplo de uso (obsérvese que en la función `main` hay dos bloques anidados):

```
#include "MemoryManager.h"  
// otras directrices del preprocesador  
// declaraciones de funciones  
  
int main()  
{  
    {  
        // declaraciones de variables  
  
        // cuerpo de la función main  
    }  
    MemoryManager::dumpMemoryLeaks();  
}
```

`MemoryManager` está fuera del bloque interno, donde se escribe todo el código de `main`.

La detección de lagunas de memoria será aplicable a cada una de las prácticas restantes cuando se haga uso de memoria dinámica. Esto es, la función `main` tendrá en todas las prácticas el formato expuesto en el ejemplo anterior. Así mismo se deberá presentar el código perfectamente tabulado y documentado.

PARA PROBAR EL FUNCIONAMIENTO DE ESTA PRÁCTICA Y DE LAS SIGUIENTES TIENE A SU DISPOSICIÓN JUEGOS DE PRUEBAS EN EL SITIO DONDE DESCARGÓ LA PRÁCTICA.

LOS DATOS AHÍ EXPUESTOS SERÁN INTRODUCIDOS EN EL ORDEN QUE USTED HAYA PROGRAMADO LA ENTRADA.

EL DÍA DE LA EVALUACIÓN DE LA PRÁCTICA, SI ÉSTA NO PASA ESTOS JUEGOS DE PRUEBA SERÁ CONSIDERADA NO VÁLIDA.