FP::II();
//G.I.S.I.T.

# Práctica 2
*Aspectos avanzados de la POO*

## Objetivos de la práctica

En esta práctica vamos a diseñar diversas clases en C++ para ejercitar determinados aspectos avanzados de la Programación Orientada a Objetos, en concreto los relacionados con la herencia y las plantillas.

## Ejercicio 1 (herencia)

En este ejercicio vamos a implementar una clase para manipular matrices dinámicas compuestas por números enteros denominada *Matriz*. Dicha clase contendrá un array bidimensional de enteros (*int **M*) y su tamaño, que vendrá determinado por su número de filas (*NF*) y su número de columnas (*NC*). A continuación describimos algunos de los métodos y operadores de *Matriz*:

- Constructor por defecto: creará una matriz nula (de 0x0 elementos)
- Constructor de copia.
- Constructor con parámetros: recibirá el número de filas y columnas deseado y creará una matriz de dicho tamaño cuyos valores serán números aleatorios comprendidos entre 0 y 99.
- Destructor
- Operadores: asignación, flujo de salida(<<), suma, resta y producto de matrices (incluyendo auto-suma += auto-resta -= y auto-producto *=) así como producto (por la izquierda y por la derecha) y auto-producto por un escalar (entero). Nótese que vamos a prescindir del operador de flujo de entrada >> porque todas las matrices se forman con números aleatorios a través del constructor con parámetros.
- Añadir todos los métodos adicionales que se estimen oportunos.

En el caso de los operadores aritméticos es necesario comprobar que los operandos son compatibles. Recuerda que para sumar o restar dos matrices deben ser del mismo tamaño y la matriz resultante será de dicho tamaño. Para multiplicar dos matrices, sin embargo, el número de columnas de la primera

debe coincidir con el número de filas de la segunda y la matriz resultante tomará el número de filas de la primera matriz y el de columnas de la segunda. Cuando los operandos no sean compatibles el resultado de la operación será una matriz nula (de 0x0 elementos). Cuando se intente imprimir una matriz nula aparecerá por pantalla "MATRIZ NULA" (en las operaciones no mostraremos ningún mensaje cuando se produzca una matriz nula, solamente al sacarla por pantalla). Cualquier operación en la que intervenga una matriz nula resultará a su vez una matriz nula.

Una vez creada la clase *Matriz*, construiremos cuatro clases a partir de ella, utilizando el mecanismo de la herencia:

- *MatrizG*. Representa una matriz **gruyere**, esto es, una matriz en la que los ceros y los elementos distintos de cero se alternan (si fila+columna es par la celda se rellena con un 0 y con un número aleatorio entre 0 y 99 en otro caso). Se formará por herencia directa de *Matriz*.
- *MatrizC*. Representa una **matriz cuadrada**, esto es, una matriz cuyo número de filas y columnas coincide. El constructor por parámetros de esta clase recibirá un solo entero que representará tanto el número de filas como el de columnas. Al igual que en *Matriz*, dicho constructor será el encargado de rellenar los elementos de la matriz con números aleatorios entre 0 y 99. Se formará por herencia directa de *Matriz*.
- *MatrizD*. Representa una **matriz diagonal** y como tal, debe ser cuadrada, por lo que se formará como herencia de *MatrizC*. La matriz diagonal se caracteriza por tener todos sus elementos a cero excepto los de la diagonal principal. Al igual que en *MatrizC*, el constructor de *MatrizD* recibirá un solo número entero que simbolizará tanto filas como columnas y, en este caso, rellenará las celdas de la diagonal de números aleatorios entre 0 y 99 y el resto a cero.

Al formar estas clases como herencia de *Matriz*, podremos crear objetos de los cuatro tipos y realizar operaciones mixtas entre ellos. Utilizaremos los recursos de la herencia para **reutilizar al máximo** el código heredado y así minimizar la implementación de las clases hijas.

# Ejercicio 2 (plantillas)

En este ejercicio vamos a reescribir el código del ejercicio anterior con plantillas, de forma que podamos construir las distintas matrices con distintos tipos base (*int*, *double*, *float*…). Las nuevas clases se llamarán *MatrizT*, *MatrizGT*, *MatrizCT*, y *MatrizDT*.

Con las plantillas podremos operar con matrices de cualquier tipo incluyendo tipos definidos por nosotros, siempre que tengan implementados todos los operadores que utilizan las plantillas. En este sentido, vamos a crear una clase llamada *Reducido* que representa un número de una sola cifra (de -9 a 9). Para calcular el reducido de un número cualquiera, debemos sumar todas sus cifras repetidamente hasta que se el resultado se quede en una sola cifra. Por ejemplo, para calcular el reducido de 1734 hacemos 1+7+3+4 = 15. Como el resultado tiene 2 cifras, volvemos a descomponerlo 1+5 = 6. Por tanto, el reducido de 1735 sería 6. De igual forma, el reducido de -1734 sería -6.

Operar con números reducidos es muy sencillo. Por ejemplo si tenemos los números reducidos 3 y 8, la suma sería 3+8 = 11 => 2, el producto sería 3*8 = 24 => 6, la resta 3-8 = -5, y así sucesivemente. Prepararemos *Reducido* para que podamos utilizarlo como tipo base desde cualquier matriz. Para ello *Reducido* tendrá un constructor a partir de un número entero para que pueda ser inicializado con un número de 0 a 99 desde *MatrizT*. Además, implementaremos todos los operadores que utilicemos en las matrices para que puedan operar con este tipo base.

# Entrega

La entrega de la práctica se realizará a través del Campus Virtual (dentro de Evaluación->Controles), como máximo el **jueves 30 de abril** (hasta el 30/04/2020 a las 23:59h). Constará de un solo archivo comprimido (*zip*) con todos los archivos fuente de los ejercicios y un archivo de documentación (en *pdf*) que constará de los mismos puntos que la documentación de la práctica 1.