• Scientific Computing with C++
Course Project - I
Extending Class Functionalities

1. **Complex Number Class**
   Extend the Complex class taught in the course project to include the following features.

   - Methods called `GetRealPart` and `GetImaginaryPart` that allow us to access the corresponding private members. In the class of complex numbers, the members representing the real and imaginary parts of the complex number - called `mreal` and `mimag` - are `private` members. These members may be set through using a `constructor`, but there is no way to access them.
   - `Friend` functions `RealPart` and `ImaginaryPart` so one may either write `z.GetImaginaryPart()` or `ImaginaryPart(z)`.
   - An overridden `copy constructor`.
   - A `constructor` that allows us to specify a real number in complex form through a constructor that accepts one double precision floating point variable as input, sets the real part of the complex number to the input variable, and the imaginary part to zero.
   - A `const` method `CalculateConjugate` which returns the complex conjugate $x - iy$ of a complex number $x + iy$.
   - A method `SetToConjugate` which has a `void` return type and sets the complex number $x + iy$ to its complex conjugate $x - iy$.
   - **Optional** - Write code to `dynamically` allocate memory for a 3×3 matrix of complex numbers. Extend this code to calculate the exponential of the matrix, where the exponential of a matrix A is given by
     $$\exp(A) = \sum_{n=0}^{\infty} \frac{A^n}{n!}$$
     where, in practice, the infinite sum above is truncated at a suitably large value of n. Having allocated the memory for this array dynamically what should you now do?
   - **Optional** - Test the class to ensure that special cases give sensible results. For example $(0 + 0i)^n$ should equal zero for most values of n, but any number raised by $n = 0$ should return 1.

2. **A Simple 2D Matrix class**
   Develop a class of 2×2 matrices of double precision floating point variables that has the features listed below.

   - An overridden `default constructor` that initialises all entries of the matrix to zero.

- An overridden `copy constructor`.
- A `constructor` that specifies the four entries of the matrix and allocates these entries appropriately.
- A `method (function)` that returns the determinant of the matrix.
- A `method` that returns the inverse of the matrix, if it exists.
- `Overloading` of the assignment `operator`, allowing us to write code such as $A = B$; for instances of the class `A` and `B`.
- `Overloading` of the unary subtraction `operator`, allowing us to write code such as $A = -B$; for instances of the class `A` and `B`.
- `Overloading` of the binary addition and subtraction `operators`, allowing us to write code such as $A = B + C$; or $A = B - C$; for instances of the class `A`, `B` and `C`.
- A `method` that multiplies a matrix by a specified double precision floating point variable.