Programmation 1 - HAX816X - 2021/2022 - Evaluation 3- Inheritance and Polymorphism

Les fichiers à rendre à l'issue de cette évaluation sont indiqués en rouge. merci de respecter la dénomination demandée. Les fichiers à rendre à l'issue de cette évaluation seront placés dans un unique fichier zip qui sera déposé sur l'ENT.

We aim at improving our toolbox for linear algebra: we want to add a new matrix class dedicated to the storage of dense symmetric matrices. Indeed, recognizing the special character of symmetric matrices can save time and storage during the solution of linear systems. Given a square symmetric matrix $A = (a_{ij})_{1\le i,j\le n}$, **only the upper (or lower) triangular portion of A has to be explicitly stored**. The implicit portions of A can be retrieved using the symmetry relation. An efficient data structure for storing dense, symmetric matrices is a simple one-dimensional array. If the lower triangular portion of A is retained, the array is organized in the following manner:

$$\text{values} = [a_{11}, a_{21}, a_{22}, a_{31}, a_{32}, \dots, a_{n1}, \dots, a_{nn}].$$

If array and matrix subscripts are one based, the entry a_{ij} is retrieved from the 1d array by the following indexing rule:

$$a_{ij} = \text{values}[\frac{i(i-1)}{2} + j].$$

If array and matrix subscripts are zero based, the formula is

$$a_{ij} = \text{values}[\frac{i(i+1)}{2} + j].$$

Now, what do you have to do in this evaluation ?

(1) declare and define (in new files Sym_Mat.hpp and Sym_Mat.cpp) a new class Sym_Mat that only stores the lower triangular part of symmetric square matrices in a one dimensional array.
As for the class Full_Mat, this new class should inherit from the Abstract_Mat class and provides a constructor that allocates and initializes to 0 all $\frac{n(n+1)}{2}$ entries,

(2) overload the **operator**() to return the (i, j) entry (note that here, such an entry may be any of the $n \times n$ entries),

(3) add a copy constructor, the overload of assigment operator (=) and a destructor,

(4) define a matrix-vector product especially designed for this symmetric storage,

(5) test your class in a main.cpp program. Your tests should include the constructor, matrix-vector product, and conjugate gradient relying on an invertible symmetric matrix of your choice.

(6) * what would be the definition of the overloaded **operator**() to return the (i, j) entry when **only the upper part** of the matrix is retained (instead of the lower part) ? (you may write such overloaded operation in another additional source file named upper.cpp).

Fin du sujet.