3D-Matrix design choices

For the realization of the three-dimensional matrix, it has been chosen to base the class on a vector of two-dimensional matrices, each of which represents a plane of the 3d matrix. A class has been implemented to represent the two-dimensional matrix, including all the functions required for the 3d matrix, to be able to base the implementation of the functions entirely on the call of the same functions for each 2d matrix that constitutes it. Furthermore, the class for the 2d matrix is independent of the 3d matrix and can be reused if there is a need to represent a two-dimensional matrix.

Two-dimensional matrix: class **matrix_2d**

The 2d matrix is represented by a one-dimensional array, which allows to simulate the presence of two dimensions through suitable access methods. Below the main design choices made in implementing the required functions:

- **Conversion constructor**: It is left to the compiler to decide whether to convert a data type U to a data type T using the static_cast<T> function. If this function generates an exception of any type, the matrix under construction is destroyed and a null matrix is generated.

- **Submatrix extraction (slice method)**: a submatrix independent of the initial matrix is returned. The indices used to indicate the submatrix are 0-indexed and inclusive of the last indicated row/column.

- **Equality operator**: Two matrices are considered equal if they have the same shape and the same element for each cell. When redefining operator==, the comparison between cells takes place via the == operator for any data type T: it is therefore to be used only for custom data types which in turn correctly redefine operator==. An equals method has also been implemented, similar to operator== but which compares elements of the same type using a functor passed as a parameter, to be used if the data type T does not redefine operator==.

- **Filling method (method fill):** The method has been implemented in two versions, which differ in the way of passing iterators as parameters: in the first version, the iterators are passed by copy; in the second, they are passed by reference and a third dummy parameter is used to differentiate the signature from the first version. This second version will be used by the matrix_3d class to implement its own fill method.

- **Transformation function (function transform)**: a matrix is returned, obtained by applying the functor passed as a parameter to each element of the source matrix; it is assumed that the functor is defined correctly, i.e. as a function W -> T, where W is the template data type of the source matrix and T that of the new matrix.

- **Iterators**: being the data structure on which the class is built an array, it is sufficient to map the iterators in pointers to T to obtain random access iterators.

Three-dimensional matrix: class **matrix_3d**

The 3d matrix is represented by a one-dimensional array of matrix_2d, each of which represents a plane of the matrix. All the required functions rely on those implemented in the matrix_2d class, iterating on all the planes of the matrix. Below are the main design choices made in implementing the required functions:

- **Constructors**: when allocating an array of matrix_2d it is only possible to use the default constructor for each matrix_2d: therefore, it was first chosen to allocate the array formed only by null matrices, and then to iterate on the array to create the matrix_2d consistent with the invoked constructor, relying on the constructors of the matrix_2d and also going to eliminate the previously created null matrix_2d.

- **Operator[]:** in addition to the other required access methods, it was decided to implement an access method to the z-th plane of the 3d matrix, which returns a constant matrix_2d, therefore modifiable only from the public interface of the matrix_3d class.

- **Submatrix extraction (slice method)**: there is a problem like that of the constructors in allocating the array of matrix_2d, which is solved in the same way.

- **Transformation function (transform method):** problem and solution analogous to the slice method.

- **Filling method (fill method)**: to rely on the analogous method of the matrix_2d class, the version in which the iterators are passed by reference is invoked, to propagate their scrolling at each invocation on the planes of the matrix_3d.

- **Iterators**: it was decided to implement bidirectional iterators. They are based on the iterators of the matrix_2d; it is also necessary to memorize the current matrix_2d, its position in the matrix_3d and the total number of matrix_2d, in order to be able to pass correctly from the end of a matrix_2d to the beginning of the next one or vice versa.