Lab Project 1. The Matrix Library

Our aim is to create a software to visualise 3D scenes. In each lab project, we will create part of this software.

We need to start our work in an organised way, in order to be able to continue it in the next lab project. You are not obliged to follow to the letter, but here is an example of organisation of your files for this lab, and the following ones:

```
<software_name>/
   |
   +- bin/      (executables)
   +- build/    (objets --- .o)
   +- include/  (headers --- .h)
   +- src/      (implementation --- .cpp)
   +- test/     (test routines --- .cpp)
   +- makefile
```

As the course progresses, we will fill this structure to create the complete software.

For this first lab project, we will create a library for vectors and matrices calculations. This library will be the foundation of the software. This library must be created in C++.

You must create a header file (e.g., `libmatrix.h`) where you will define the interface of your library. The implementation must be defined in another file (e.g., `libmatrix.cpp`). All classes and functions of the library must be created in a namespace (e.g., `libmatrix`). You also have to create test routines for your library. The library must, at least, define the classes, methods, functions and constants described below.

Class Vector: Defines a fixed size sequence of n values of type T, where n and T are class parameters. The class must, at least, define the following methods:

- **Methods:**
  - `at()`: addresses the i-th element (i is give as an argument) of the vector.
    Raises an exception if i is out of range.
  - `cross()`: cross product with another vector (given as an argument). Uses only the first 3 coordinates. Raises an exception if the vector has less than 3 elements.
  - `dot()`: dot product with another vector (give as an argument).
  - `is_ortho()`: returns `true` if the vector is orthogonal to another given as an argument, `false` otherwise.
  - `is_null()`: returns `true` if the vector contains an invalid value, `false` otherwise.
    Notably, if the vector contains `nan` as values.
  - `is_unit()`: returns `true` if the vector is unit, `false` otherwise.
  - `norm()`: returns the norm of the vector.
  - `to_unit()`: returns a copy of the vector normalised.
  - operator `<<`: overloads this operator for outputs.
  - operator `[]`: addresses the i-th element (i is given as an argument) of the vector.
    It must also permit assignment.
    It does not verify if i is valid.
  - operator `+`: vector addition.
  - operator `+=`: vector addition and assignment.
  - operator `-`: defines two operations (overloading):

    - inverse.
    - vector subtraction.
  - operator `-=`: vector subtraction and assignment.
  - operator `*`: defines four operations (overloading):
    - multiplication of scalar and vector.
    - multiplication of vector and scalar.
    - multiplication of vector and matrix (see class Matrix below).
    - multiplication of matrix and vector (see class Matrix below).
  - operator `*=`: multiplication (all operators above) and assignment.

Class Matrix: Defines a fixed size $n \times m$ matrix of values of type T, where n, m and T are class parameters. The class must, at least, define the following methods:

- **Methods:**
  - `at()`: addresses element (i, j) (given as arguments) of the matrix.
    Raises an exception if i or j is out of range.
  - `inverse()`: returns the inverse of the matrix. Returns a null matrix if the current matrix is not invertible.
  - `is_null()`: returns `true` if the matrix contains invalid values, `false` otherwise.
  - `is_ortho()`: returns `true` if the matrix orthogonal, `false` otherwise.
  - `transpose()`: returns the transpose of the matrix.
  - operator `<<`: overloads this operator for outputs.
  - operator `[][]`: addresses element (i, j) (given as arguments) of the matrix.
    It must also permit assignment.
    It does not verify if indexes are out of range.
  - operator `+`: matrix addition.
  - operator `+=`: matrix addition and assignment.
  - operator `*`: defines four operations (overloading).
    - multiplication of scalar and matrix.
    - multiplication of matrix and scalar.
    - multiplication of vector and matrix.
    - multiplication of matrix and vector.
    - multiplication of matrix and matrix.
  - operator `*=`: multiplication (all operations above) and assignment.

Fonctions:

- `dot()`: alias for method `dot`. The two operands are given as arguments.
- `cross()`: alias pour method `cross`. The two operands are given as arguments.

Class Vec2i Defines a vector of 2 integers.

Class Vec3i Defines a vector of 3 integers.

Class Vec4i Defines a vector of 4 integers.

Class Vec2r Defines a vector of 2 reals.

Class Vec3r Defines a vector of 3 reals.

Class Vec4r Defines a vector of 4 reals.

Class Mat44r Defines a 4×4 matrix of reals.

Constants The Matrix Library must in addition define the following constants:

- `zerovector`: a vector of zeros.
- `zerovec2i`
- `zerovec3i`
- `zerovec4i`
- `zerovec2r`
- `zerovec3r`
- `zerovec4r`
- `IdentityMat`: the identity matrix.
- `Identidy44i`
- `Identity44r`