# LAB 4 - C++ Classes

**Assessment:** 7% of the total course mark.

---

## 1 General Instructions

- ◇ Your programs should be written in a good programming style, including instructive comments and well-formatted, well-indented code. Use self-explanatory names for variables as much as possible. ($\approx 5\%$ of the mark)
- ◇ You have to make sure you pass all the tests. Please note that passing the tests does not grant you automatically the full mark, we run other "hidden" test cases that are not shared with you to further assess your code. You are required to make sure your work is correctly implemented to the best of your knowledge. One task that can help you with that is to add further tests to stress the corner cases of each question.
- ◇ For each method, you are required to add at least one additional test to the Test.cpp test file.

## 2 Submission Deadline

The deadline for lab4 is Nov 18th. Please note that this is a programmed deadline in the environment, so make sure you submit in time since you will not be able to submit after that dictated deadline.

## 3 Environment Setup

The good news is that you can use exactly the same Eclipse environment you have been using in the C labs. The only change we are making from our side is using a different unit testing infrastructure for C++, it is also called CUTE. If you want to learn more about how it works and how you can create your own projects with CUTE, it has excellent step-by-step tutorials here: https://cute-test.com/guides/

- ◇ **However, the only step you need to do for the labs, is to add CUTE plug-in to your Eclipse. To do so, add it from the Eclipse Marketplace as instructed here:** Install the CUTE Eclipse Plug-in from the Eclipse Marketplace: https://cute-test.com/installation/
- ◇ When creating the eclipse project, you have to follow the instructions in the CUTE guide https://cute-test.com/guides/cute-eclipse-plugin-guide/ section **Create a Project**.
- ◇ To run the tests, follow the instructions in the same link above.

### 3.1 Importing the starter code of the lab

You should follow exactly the same process you have been following in past labs to import the starter code from the following invitation link and to create the project:

https://classroom.github.com/a/8QsNPKHC

## 4 Lab Questions

Write one C++ class:

◇ `Matrix`, which represents matrices with integer elements.

The accompanying files `Matrix.cpp` and `Matrix.h` in the provided starter code contains an incomplete declaration of the functions of the class. You need to complete the declarations of incomplete methods or constructors according to the specifications given below.

Finally, a class named `Test.cpp` includes all the test cases. You should, as usual, write at least one additional test case for every method.

### 4.1 Matrix Class

- Class `Matrix` has only the following instance fields, which have to be `private`:
  - an integer to store the number of rows
  - an integer to store the number of columns.
  - a double pointer of integers to store the matrix elements.
- Class `Matrix` contains at least the following constructors:
  - `public Matrix()` - this is the empty constructor. It dynamically allocates the memory for the matrix and initializes it to a square 3x3 matrix. This is already GIVEN to you.
  - `Matrix(int row, int col)` - constructs a `row`-by-`col` matrix with all elements equal to 0; if `row` $\leq 0$, the number of rows of the matrix is set to 3; likewise, if `col` $\leq 0$ the number of columns of the matrix is set to 3.
  - `Matrix(int row, int col, int** table)` - constructs a matrix out of the double pointer by dynamically allocating enough memory for the matrix with the same number of rows, columns, and the same element in each position as in the matrix pointed to by `table`.
- Class `Matrix` contains at least the following methods:

  1) `int getElement(int i, int j)` - returns the element on row `i` and column `j` of `this` matrix; it throws an exception if any of indexes `i` and `j` is not in the required range (rows and columns indexing starts with 0); the detail message of the exception should read: "Invalid indexes.". Example of this exception is as follows: `throw std::out_of_range( "Invalid indexes." );`

2) `bool setElement(int x, int i, int j)` - if `i` and `j` are valid indexes of `this` matrix, then the element on row `i` and column `j` of `this` matrix is assigned the value `x` and `true` is returned; otherwise `false` is returned and no change in the matrix is performed.

3) `public Matrix copy()` - returns a **deep** copy of `this Matrix`. Note: A **deep** copy does not share any piece of memory with the original. Thus, any change performed on the copy will not affect the original.

4) `void addTo(Matrix m)` - adds `Matrix m` to `this Matrix` (**note**: `this Matrix` WILL BE CHANGED) ; it throws an exception if the matrix addition is not defined (i.e, if the matrices do not have the same dimensions); the detail message of the exception should read: "Invalid operation".
Example: `throw std::invalid_argument( "Invalid operation" );`

5) `Matrix subMatrix(int i, int j)` - returns a new `Matrix` object, which represents a submatrix of `the current object Matrix`, formed out of rows `0` through `i` and columns `0` through `j`. The method should first check if values `i` and `j` are within the required range, and throw an exception if any of them is not. The exception detail message should read: "Submatrix not defined". **Note**: The new object should be constructed in such a way that changes in the new matrix do not affect `the current object "this" Matrix`.

6) `bool isUpperTr()` - returns `true` if `this Matrix` is upper triangular, and `false` otherwise. A matrix is said to be upper triangular if all elements below the main diagonal are 0. Note that the main diagonal contains the elements situated at positions where the row index equals the column index. In the following examples the main diagonal contains elements `1,9,3`.
**Example** of a 3-by-3 upper triangular matrix:
1 4 1
0 9 0
0 0 3
**Example** of a 3-by-4 upper triangular matrix:
1 5 1 4
0 9 6 6
0 0 3 8
**Example** of a 4-by-3 upper triangular matrix:
1 4 2
0 9 6
0 0 3
0 0 0

7) `public String toString()` - returns a string representing the matrix, with each row on a separate line, and the elements in a row being separated by 1 blank space. For instance like this:
1 2 3
4 5 6
7 8 9

You may implement public methods in class `Matrix` to return the number of rows and the number of columns.