## Task 2 – Matrix (Task-2-Matrix)

Short explanation: You are provided with a Matrix.h file and a main.cpp file which uses it.

* Create a Matrix.cpp file which implements the class Matrix (declared in Matrix.h)
* The files should successfully compile together (e.g. in a Code:Blocks project with MinGW)
* The resulting program should output “tests passed” (i.e. the tests in main.cpp should pass)
* Submit a .zip file containing the Matrix.cpp implementation and nothing else

Long explanation:

You are tasked with implementing the definition of a Matrix class. A matrix in this case is a data structure, which represents a 2-dimensional integer (int) array. Each row in that array has an equal number of elements (“columns”) to any other row. The matrix class has methods to initialize, change size, get/set an element at a certain row and column (i.e. a certain element of a certain row of the matrix), get the number of rows/columns in the matrix, as well as get a string representing the matrix.

Advice: see the files you are provided with and if you don’t understand something come back and read the task description – you will probably understand what to do much quicker that way, than reading words from a text file. Here are the expected behaviors of some of the methods:

* changeSize – changes the number rows and/or columns in the matrix, and makes sure any previous values are kept at their positions after the rows and columns are changes (if they fit into the new matrix). I.e. if we have a position at row, col in matrix m, then we do m.changeSize(numRows, numCols), if numRows > row and numCols > col, then the value at position row, col should remain the same as before the **changeSize** operation. If a position didn’t previously exist in the matrix (i.e. if the changeSize operation added new rows/columns), then that position should be set to the value 0
* toString – returns an std::string, which contains the elements of the matrix, where elements in a row (i.e. columns) are separated by single spaces (" ") and rows are separated by std::endl. The order is top-down, from left to right, i.e. the string should begin with the elements for row 0, then row 1, … row R-1, where R is the number of rows in the matrix, and each row’s elements should start from element 0, then element 1, … element C-1, where C is the number of columns in the matrix

You are also provided with some private fields and methods. You are **required to use the private fields in your implementation**, but can **choose whether to use the private methods** or not (you still have to give them definitions for the code to compile, but it is your decision whether to call them or not).

If the get or set methods receive parameters which are not inside the matrix (i.e. the row and column parameters are larger than or equal to, respectively, the matrix rows or matrix columns), then you should execute this exact line of code:

throw std::range\_error("Out of bounds");

This will throw an exception and crash the program and this is the expected behavior of the class in this situation. You do not need to understand what this code does exactly, you just need to do it when an access which is outside of the range of the matrix is attempted by outside code.

The code you will be provided with has a main() function. You should ONLY implement the Matrix class and nothing else. The provided main() function will use your implementation – the code of the main() function you will receive will automatically run tests on your code with some predefined test data. When you submit your solution to the Judge system, the system will update that main function to use other test data (so don’t try to cheat by writing code that just works in a certain way for the test data you see in the main function you are provided). If all of the tests in main() pass successfully, then you will get a “tests passed” line printed on the console – otherwise you will get a message indicating what test failed and some partial info about why it failed.

Keep in mind that even if the tests pass locally for you, that doesn’t guarantee your code is completely correct. The Judge tests will use different data, so make sure you test manually too.

### Solution Skeleton & Submission Instructions

You will be provided with a Matrix.h and a main.cpp file

* Matrix.h contains the declaration of the Matrix class and its members
* main.cpp contains the program entry point (the main() function) and all the testing logic and operations which will be done with the Matrix class to test it out

You must submit a Matrix.cpp file in a .zip archive. The Matrix.cpp file should contain the definitions of the members of the Matrix class, which is declared in Matrix.h. Hint: Matrix.cpp should have an #include "Matrix.h" directive.

The Judge system will place all the files from the solution skeleton and all the files from the zip you submit in a single directory, then it will build a program from them and run it.

Advice: create a project with the IDE you use, place the solution skeleton in it, and then write empty definitions for each member of the class – so that you have something that compiles correctly. Then start actually implementing the members one by one, and run the program after each complete implementation you do to see what works and what doesn’t (running the program will run the tests in main.cpp)

### Input & Output

The main.cpp file handles input and output. You should only correctly implement the Matrix class correctly for the program to work.

### Restrictions

The Judge system will run the program with various sizes and values for the matrix.

The total running time of your program should be no more than 0.5s

The total memory allowed for use by your program is 5MB

### Example I/O

|  |  |
| --- | --- |
| Example Input | Expected Output |
|  | tests passed |