Module [201] – Practice Assignment

We want to implement two functions which are part of a program meant to operate on two dimensional integer matrix. One function will test whether a given matrix is in order or not. The other will sort it. In addition to these functions, we will also work on defining code to automate the testing of our two main functions. We're getting one tiny step nearer to how testing is actually done "in the real world".

This is the first practice assignment where you will be able to collaborate with other students by posting your tests in a format making it easy to run other students' tests on your own implementation of the above-mentioned functions. You should find that having other students evaluate your program by running their tests against your code, will help you find bugs, catch features you forgot to implement, and fix miss-interpretations of the requirements… All this without actually having other students take the keyboard away from you to fix your "bugs" without giving you an opportunity to learn from your errors.

In addition to being used to detect novice errors and offer tutorials on compiler errors, CLUE will be used this time to download and upload your PA.

The files you will have to work with

You will be provided with several files to get you started working on this assignment. Make sure you edit them but do not alter their name or add new files. Here are the files;

main.c	do not modify this file, it will by default call the two functions you have to implement and apply them to a list of tests which you will also define.
setup.h	This header file contains some constants which are used in the whole program to make things a little easier. You should not modify the number of rows and columns of the matrices but you will have to alter the number of tests since you will be the one writing them. When you modify this file, make sure to rebuild the whole project as explained in the comments.
tools.c	Part of your work will consist in implementing two functions. The prototypes are already provided along with comments describing what they should be doing.
tests.c	The other part of your work will consist in devising tests which will guarantee that your implementation of both functions is valid. You will have to comment each test, as illustrated with the first one which I give you for free, to explain what you are trying to evaluate with it.

Task #1 – Implementing and testing your isMatrixSorted

Some reading before you start

Before to start working on implementing anything, you'll have to read the entire source and make sure you understand what it is doing so that you are able to add to it without breaking everything. Pay specific attention to the way the program parts already written for you will use the data in **tests_inputs** and **tests_expected**, defined in *tests.c*, in order to invoke your function with some pre-determined parameters and verify it returns the expected result.

In addition, spend some time reading the requirements in this document and modify the arrays in *tests.c* to reflect the tests you think your program ought to pass to meet these requirements. When you do so, make sure you update accordingly the value of the **NB_TESTS** integer constant defined in the same file.

When you think your tests are sufficient, post this preliminary version in a new thread on the module [201] forum.

Implementing your solution

Next step is implementing the **isMatrixSorted** function in the *tools.c* file so that it detects whether a matrix passed as parameter is already sorted or not and returns an appropriate value. Refer to the comments in the code for the specific details.

To give you an example, a two dimensional 3x5 matrix is sorted when its elements are ordered as follows;

3	5	9	20	20
42	50	55	99	142
200	209	500	900	999

Testing your solution

Now is time to take what you learned when you actually started implementing the solution and when you had to fend off your first bugs. Revise the tests you previously used to populate the arrays **tests_inputs** and **tests_expected** in file *tests.c*. As you add or remove tests, make sure you remember to update accordingly the value of the **NB_TESTS** integer constant defined in the same file.

When you are satisfied that you have a good suite of tests, post your *tests.c* file on the module [201] forum again, revising the one you initially posted right as you were reading the requirements.

Having others do the testing for you

Now that you're pretty happy with your implementation and your testing, what about we get someone else to try to break your work for you?

Go to module [201] forum and download a few *tests.c* files posted by your classmates. Rename each file *tests-smith.c* or *tests-jake.c* before to save them to your project's folder. You should rename your own *tests.c* as *tests-myown.c* to make sure it never gets overwritten.

Now, each time you want to use another student's tests against your implementation of the function, simply copy his/her file as *tests.c* and recompile the whole project. Their testing data will be applied to your function.

This will help you find bugs, features you forgot to implement, features you miss-interpreted, … With just enough help to get you unstuck but while still letting you develop these troubleshooting skills you need to develop on your own.

Feel free to post in your own thread or in the threads of students whom tests you are using to let them know how it worked. E.g. "Sorry but your test #3 is wrong, the assignment doesn't ask for this!!!".

Bit of refactoring

Do not attempt the rest of this assignment until your function is working on a good quality suite of tests. See the last section for advices on how to design a good "test harness".

Now that things are working, what about refactoring / simplifying your program and tests? Adding comments and making your code easier to maintain.

Task #2 – Implementing and Testing matrixSort

When we are going to test your 2nd function, **matrixSort**, we are going to rely on **isMatrixSorted**. This means you should not attack this part of the practice assignment before to be fairly sure your **isMatrixSorted** function works.

Some reading before you start

Read the requirements and the comments in the program itself to make sure you understand what's expected of this function and then take a shot at adding more tests focused on ensuring the function works. This should be relatively easy and maybe you won't be able to think to many more tests to add to those you used to validate **isMatrixSorted**. Go back to your thread in module [201] forum and post your new version of the tests, if any.

Implementing your solution

The **matrixSort** function will take a two dimensional integer matrix which might or might not be already ordered and order it in a manner which **isMatrixSorted** will recognize as ordered. There are different algorithms you might implement.

Most students like to use the "bubble sort" from the textbook and adapt it to a two dimensional matrix. I personally think it's a bit too much work.

Others go with the following logic; we need to look at each element of the matrix in order, one after the other. For each such element, I use loops to go other all the elements between it and the end of the matrix. Each time, I compare the two elements and swap them if they are out of order. This result is me looking for the smallest element and putting it first in the matrix, then looking for the next smallest one and putting it second in the matrix. For a one dimensional array, the algorithm would look something like;

```
// my array name is data which has indexes between 0 and N-1
int i,j;
for(i=0 ; i < N-1; i++){
      for(j=i+1; j < N ; j++){
            if( data[i] > data[j]) SWAP THEM
      }
}
```

The hard part is to adapt this to a two dimensional matrix. Some students found interesting work around so feel free to explore.

Testing your solution

Again, you might have learned a bit more about the problem as you were trying to implement it. You might have found some bugs which you are now able to write a test to make sure you won't revert back to them.

E.g. I had a bug when only the first and last elements were not in order, I was pretty much off by one in one of my loops. I'm now adding a test to make sure I detect this specific bug from now on.

Revise your tests, post them again.

Having others do the testing for you

Others might have had different bugs, they might have got different insights about the problem at hand. Time to see if your implementation handles their tests.

Bit of refactoring

Now that things are working, what about refactoring / simplifying your program and tests? Adding comments and making your code easier to maintain.