Enter the matrix

This exercise will allow you to review both pointers arithmetic, dynamical memory allocation while also getting a glimpse at how multi-dimensional arrays are handled internally by our programming language.

- **thematrix.h** will contain the headers (aka declarations) of our functions.
- **thematrix.c** will contain the definitions of all the functions we will implement.
- **tests.c** will contain the main function used to test our functions.

Architecting the Matrix

In our program, a dynamically allocated matrix of ROW by COL integers will be defined as a memory zone large enough to hold ROW x COL x sizeof(int) bytes. While it is convenient to visualize a matrix as a table, internally our elements will be arranged one after the others, row after row as illustrated by the following diagrams.

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

This is the conceptual view of a 2x3 matrix, below is the representation in memory as a series of contiguous integers.

(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)

As you can see, the same information can be allocated linearly in memory. In order to access element (x,y) we will need to do a little bit of pointer arithmetic though. Since all rows are stored one after the other, the first thing to do is to compute the offset from the start of the matrix in terms of complete rows;

(matrix start address) + x * (size of an entire row)

We can then add the offset corresponding of the position of our element inside this row;

(matrix start address) + x * (size of an entire row) + y * sizeof(int)

This allows us to manipulate a dynamically allocated matrix of integers as a *int** and be able to access any of its element by applying the above formula to this pointer.

Work to do

The following functions need to be implemented and tested;

int* thematrix_allocate (int nbRows, int nbColumns);

- Returns NULL if *malloc* didn't work.
- Returns the address of a newly allocated dynamical integer array of nbRows x nbColumns integers otherwise.

void thematrix_deallocate (int * data);

- Do not do anything if *data* is NULL
- Deallocates the dynamic memory pointed to by *data* otherwise

void thematrix_deallocate_cleanly (int** data);

- Same than above, plus assign **data* to NULL thus preventing the pointer which address was used as effective parameter to be dereferenced to the old allocated address by mistake

void thematrix_display (int * data, int nbRows, int nbColumns);

- Display the contents of the integer matrix *data*

void thematrix_scan (int * data, int nbRows, int nbColumns);

- Display the contents of the integer matrix *data*

Testing

- Update *tests.c* to test your functions and make sure that all the error cases are handled properly.