EEEE1042 - **Practical 6**
Bubble sort.

In this practical session we are going to implement and test a simple sorting algorithm of a 1D array called the bubble sort algorithm.

- The bubble sort is a well-known very simple, sorting algorihtm.
- Bubble-sort is usually the first sorting algorithm implemented by people new to programming and/or sorting.
- Bubble-sort is O(N^2) and is not very efficient, implementing it is more for pedagogical purposes.
- In bubble sort, the algorithm proceeds as follows:
    1. Start with an unsorted array
    2. Iterate `i` over every element of the array
    3. If the `i`th element is bigger than the `(i+1)`th element, swap the two elements in positions `i` and `(i+1)`
    4. Complete the iteration over `i` until entire array has been covered.
    5. If any swaps occurred during the iteration over `i`, go back to step 2 for another new iteration.

  In this way, successive elements of the array are swapped, or bubble upwards/downwards to their correct position in the array.
- In this practical, the **random number generator** from the last practical is used to populate the array with random numbers to be sorted by the bubble-sort algorithm.

Write a program in C to implement the bubble-sort algorithm according to these instructions

1. `#define N` as the number of elements in the array in the program header
2. `#define M` as the maximum number to be used in the array in the program header

3. We need to seed the random number generator with a unique seed so that we get a different random number sequence each time. To do this we use the `srand()` command which seeds the RNG, and pass it the output from the `clock()` (which gives it a random seed) command as follows:

   ```
   srand(clock());
   ```

   This command can be put at the top of your `main()` function to seed the RNG.

4. In order to use the previous `srand()` command, you need to `#include<stdlib.h>` for the `rand()` function and `#include<time.h>` for the `clock()` function in the header.

5. Now declare your array of integers `x[N]` and you can start working on it in the `main()` function.

6. Following the POP paradigm, declare, define and call functions: `populateArray(x);` `printArray(x);` and `sortArray(x);` to

   (a) Initialize the `N` elements of the array with random integers between 1 and `M`

   (b) Print out the array contents

   (c) Sort the array using the bubble-sort algorithm described above.

   You may use whichever method (LSB or MSB) that you prefer to calculate the random numbers between 1 and `M` in the `populateArray(x);` function..

7. When you are done sorting your array, call the `printArray(x);` function a second time to confirm that your array is properly sorted. One possible output of your program would look like this when N is set to 10 and M to 100:

   ```
   ===============before sort
   0 65
   1 89
   2 45
   3 59
   4 94
   5 47
   6 82
   7 49
   8 65
   9 44
   ===============after sort
   0 44
   1 45
   2 47
   3 49
   4 59
   5 65
   6 65
   7 82
   8 89
   9 94
   ```

Save your program to the file: `yourName_practical6_EEEE1042.1.cpp` and submit it to Moodle. Be sure you have properly commented your code. When you are done with the practical, you may use the remainder of the session to work on your coursework.