# Homework 0 — due Wed April 11, 11:00pm

Please do not modify the file names or the Makefile.
`$ make`
will make all the files
`$ make main_q1`
will only make the first problem, etc.

Enjoy!

## Problem 1

5 points. We have two arrays of type `double` given by pointer `double *a` and `double *b`. Memory for both arrays should be allocated using:

```
a = (double *)malloc(1000000*sizeof(double));
b = (double *)malloc(1000000*sizeof(double));
```

Make sure you free the memory at the end of your code.

Initialize the data stored at memory addresses `a` and `b`.

Write a `C++` function that copies the data stored at address `a` to `b`, and vice versa (swap the data). Your function must perform the copy using `std::memcpy()`. Test that the swap copy was successful.

Time your code using `std::chrono::system_clock` and `std::chrono::duration`. Turn in your code and report the measured time.

Repeat the swap operation but this time simply swap the value of the pointers `a` and `b`, without copying the data. By checking the memory addresses, test that the swap copy of the pointers was successful. Turn in your code and the measured time.

Compare and discuss the two execution times you measured.

## Problem 2

Assume that we want to create a `C++` library for matrices. Since matrices often have structure, it doesn't make sense to create one class for all matrices. For example a diagonal and a dense matrix have very different storage requirements and using a dense matrix class to store both would be very inefficient. A better approach is to define an interface, to which all matrix classes must adhere and then create different implementations for different matrix structures.

We want to explore the use of inheritance by implementing a `C++` class for square lower triangular matrices. This class should have at least the following properties:

- Inherit from a *pure abstract* base class for general matrices.
- Use a template argument for the type of the matrix entries. Assume that the type supports all arithmetic operations: `+`, `-`, `*`, `/`, such as `float` or `double`.
- It should accept a constructor with input argument `n`, the size of the matrix.
- The storage should be $n(n+1)/2 + O(1)$ where $n$ is the matrix size.
- You should define an operator `()` to access and modify entries in the matrix. The operator should take as input a row `i` and a column `j`.
- You should define a method to calculate the $\ell_0$ "norm" (the number of non-zero elements).

(a) 25 points. Write the `C++` class. Turn in your code.

*Bonus* 5 points. The CUDA Thrust library, which we will use later this quarter, has many similarities to the C++ STL. It is therefore beneficial if you have some prior knowledge of the STL. To receive bonus credit, we ask that you replace any loops in your $\ell_0$ norm calculation with the `std::count_if` function from the `algorithm` library. You may additionally find the `functional` library useful.

(b) 15 points. We want you to demonstrate that you know how to write correct code, which includes knowing how to test your code. The instructions are a bit vague because we want to see how much you can figure things out by yourself.

Describe the operations that you want your class to correctly support and implement. Explain which tests you want to write to check whether your class correctly implements the features you want.

(c) 15 points. Write and turn in code that implements these tests. Run your code. Did your class pass all your tests?

### Problem 3

20 points. Assume you completed the matrix library in Problem 2 and are writing a function that needs to use a sequence of matrices as input. The input argument should be a `std::vector` of matrices. Since all matrices implement the same `Matrix` interface, and therefore support the same basic operations such as addition and multiplication, we would like to use that `Matrix` interface rather than the matrix classes for specific cases. This will allow appending different kinds of matrices to the same `std::vector`. How would you implement this? Please complete the code given in `main_q3.cpp` and turn it in.

### Problem 4

20 points. You are running a Monte Carlo simulation and would like the ability to quickly query the number of samples in a range given by [`lb`, `ub`]. There are many ways to achieve this, but one possibility is to store all samples in an `std::set`. Although `std::set`s have many similarities to the mathematical notion of a set, they have the additional property that they are sorted.[1] For simplicity, we assume that each data point is unique.[2] Write a function that takes the following parameters:

- A set containing the data, `std::set data`,
- A range given by `double lb` and `double ub`.

And returns the number of data points within the range. You will need to use the `std::set::lower_bound` and `std::set::upper_bound` functions.

Use the following code to generate some test data

```
#include <random>
#include <set>
  ...
std::set<double> data;
std::default_random_engine generator;
std::normal_distribution<double> distribution(0.0, 1.0);
for (unsigned int i = 0; i < 1000; ++i) data.insert(distribution(generator));
```

and report the number of points in the range `[lb, ub] = [2, 10]`. Turn in your code.

Total number of points: 100 (+ 5 bonus)

## 5 Submission instructions

To submit:

1. For all questions that require explanations and answers besides source code, put those explanations and answers in a separate PDF file. The name of the file should be: `hw0.pdf`
2. The homework should be submitted using a submission script on `rice`. The submission script must be run on `rice.stanford.edu`.

[1] This is a consequence of the fact that they are implemented as binary search trees

[2] Otherwise we would need to use the `std::multiset`, which lifts the requirement that each entry be unique

3. Copy the directory containing all your submission files to `rice.stanford.edu`. You can use the following command in your terminal:

   ```
   scp -r <directory to be submitted> <your SUNetID>@rice.stanford.edu:<your directory on rice>
   ```

   Here is the list of files we are expecting:

   ```
   Makefile
   main_q1.cpp
   main_q2.cpp
   main_q3.cpp
   main_q4.cpp
   matrix_lt.hpp
   matrix.hpp
   hw0.pdf
   ```

4. Make sure your code compiles on `rice` and runs. To check your code, we will run:

   ```
   $ make
   ```

   This should produce 4 executables: `main_q1`, `main_q2`, `main_q3`, and `main_q4`.

5. Type:

   ```
   $ /usr/bin/python /usr/class/cme213/WWW/script/submit.py hw0 <directory with your submission files>
   ```

   The `submit.py` script will copy the files listed above to a directory accessible to the CME 213 staff. Only files in the list above will be copied. Make sure these files exist and that no other files are required to compile and run your code. In particular, do not use external libraries, additional header files, etc, that would prevent the teaching staff from compiling the code successfully. The script will fail if one of these files does not exist.

6. You can submit at most 10 times before the deadline; only the last submission will be graded.

7. There will be a 10% penalty per 24 hours for late submission. We will not accept submissions that are submitted past two days.

You may review your submissions by typing in the following command in your terminal while you are on `rice`:

```
ls /usr/class/cme213/WWW/submissions/hw0/<your SUNetID>/<number of submissions>
```

In this directory, because of the ACL permissions,[3] you are only authorized to list and create new files. You cannot read, move or change the content of the files inside those directories. It is a violation of the honor code to submit your homework files without using the script provided by the CME 213 staff, unless by special permission.

[3] `https://uit.stanford.edu/service/afs/sysadmin/userguide/filepermissions`