CSCI 2500 — Computer Organization
Lab 03 (document version 1.0)

- This lab is due by the end of your lab session on Wednesday, September 13, 2023.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- You **must** show your code and your solutions to a TA or mentor and answer their questions to receive credit for each checkpoint..
- Labs are available on Tuesdays before your lab session. Plan to start each lab early and ask questions during office hours, on the Discussion Forum on Submitty, and during your lab session.

1. **Checkpoint 1:** Familiarize yourself with the code provided in `mm.c` (matrix multiplication). Compile this code and test it thoroughly to make sure that it is working correctly. Make sure you follow the guidelines below:

    - Every test case should be represented by its own input file. Do not overwrite or edit the same file. Create a separate file for each test case, so that you end up with an entire collection of input files.
    - Test both the "normal" cases (when the program is able to successfully compute the product of two input matrices) and "error" cases when the program is expected to print an error message. All possible error messages should be tested.
    - Test the code with "normal" cases (several matrices of different sizes) as well as all "edge" cases you can think of. For example, matrices of sizes 0 by 0, one by one, matrices with only one row or one column, huge matrices (think thousands, millions, or even billions of elements). Try to come up with additional "dimensions" that input domain can be partitioned into. For example, the size of matrices, the values of matrices (zero, non-zero, negative, non-negative, integer, double, characters or other not valid numbers), etc.
    - We provide three input test files (`test1.mat`,`test2.mat`, and `test3.mat`) to help you get started. You need to create at least seven more input files that are significantly different from the ones we provided and from each other.

    Since you should end up with at least ten different input files, note how tedious the process of repeating the same command or very similar commands to compile your code and run it with different inputs is.

2. **Checkpoint 2:**

    Now, split code from `mm.c` into several `*.c` and `.h` files as follows:

    - `mm_alloc()` should be placed in `alloc.c`.
    - `mm_matrix_mult()` should be placed in `mult.c`.
    - `mm_print()` should be placed in `print.c`.
    - `mm_read()` should be placed in `read.c`.
    - `main()` should be placed in `main.c`.
    - There should be a matching header file for each `.c` file except for `main.c`.
    - The definition of `matrix` should be in its own header file `matrix.h`.

- Each header file should contain function prototypes for all functions contained in the corresponding `.c` file.
- All header files should be in their own directory `include`.
- Don't forget to place header guards (e.g., `#ifndef MATRIX_H #define MATRIX_H ... #endif`) in all header files.
- You are not allowed to make any changes to the functionality of the original code or alter any details of implementation, other than splitting the code as described above. In particular, your code should still read file names from `stdin`. DO NOT change the code to receive input file names from command line arguments!

3. **Checkpoint 3:** Write a `Makefile` for your project according to the following requirements:

   (a) At least several targets need to be defined, including phony rules with no targets (e.g., `clean`), and rules with no commands (e.g., `all`).

   (b) At least several variables need to be defined, including variables that store the name of the compiler executable and compiler flags.

   (c) Patterns and special variables should be used to define rules instead of listing individual `.c` files.

   Test all targets to make sure everything in your `Makefile` works as expected.

4. **Checkpoint 4:** Write a bash script to perform all of the following tasks in a single script:

   (a) Call `make` command to clean your project but only if the -c flag was specified in the command line for your script.

   (b) Call `make` command to build your project, compiling only the necessary files.

   (c) Run Dr. Memory or Valgrind to check for memory leaks.

   (d) Run all the tests that you tested the original code with to confirm that your code after splitting is working correctly and preserves all the functionality of the original version. Make sure you avoid unnecessary copying and pasting the same commands even if you make slight changes to them. For example, there should be just a single command to run your code with an input file but this command should be placed inside a loop that goes over a sequence of input file names.

   You need to make sure to use all of the following features of shell scripts: command line arguments, I/O redirection, variables, functions, conditionals, and loops.

   Run your script with different command line arguments to ensure that it works as intended.

5. **Optional Checkpoint 5:**

   This checkpoint can be used to make-up up to one checkpoint from any previous lab. It cannot be used towards any future lab.

   (a) Write a Python program that performs matrix multiplication using NumPy package. Make sure that your Python implementation accepts the same format of input files and produces the same format of output as your C implementation.

   (b) Run your C and your Python implementations on all test cases you created in Checkpoint 1 and compare the outputs to make sure they match. Time the execution of each implementation for each of the inputs.

   (c) Make a single plot of execution time vs. output matrix size for your C and your Python implementations. Use `gnuplot` or any other similar library.