CSCI 2500 — Computer Organization
Lab 04 (document version 1.0)

- This lab is due by the end of your lab session on Wednesday, September 28, 2022.
- 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. **If you just submit your lab on zyBooks but do not show it to the lab TA/mentor to get a checkpoint, you will not receive any credit**.
- 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:** On your *nix system, make sure you have access to the following commands: `find`, `grep`, and `sed`.

   (a) Clone or download Submitty source code from the GitHub repository (https://github.com/Submitty/Submitty) and unpack it to a local directory.

   (b) Use a single `find` command with a regular expression to find all files (but not directories!) in the Submitty repository written in C++(`.c`, `.cc`, `.cpp`, `cxx`, `c++`, `h`, and `hxx` files), PHP (`.php` files), and Python (`.py` files) whose names are between 4 and 7 alphabet characters (i.e., a to z, case insensitive) optionally followed by one or two digits. Output the names of all matching files with path.

   (c) Use a single `grep` command with a regular expression to find all Python files (`.py`) in the Submitty repository that use list comprehension with `if` (e.g., `[ line_ for line_ in line_list if len(line_) > 0 ]`). Output the names of all matching files with path as well as each matching line, four lines preceding the matching line and four lines following it.

   (d) Use `find` and `sed` as a single command (i.e., use `-exec` flag of `find` to run `sed`) with a regular expression to change `str.format()` method of formatting strings in all Python files (`.py`) in the Submitty repository with newer f-Strings. For example,
   `"Missing the yaml file {}".format(file_path)`
   should be replaced with
   `f"Missing the yaml file {file_path}"`
   Remember that Python may use either single quotes or double quotes to delimit string literals.

   For simplicity, you may make the following assumptions:

   - Each Python statement is written on a single line
   - There is only one argument in `str.format()`
   - The expression inside `{}` in the original code is always empty
   - There will be no varargs in the argument of `str.format()` (e.g., `"PGPASSWORD='{}' host={}".format(*variables)`

   Remember that your replacements should only be made on applicable lines of code. Other code should not be affected. After performing replacement all code should remain syntactically valid and correct.

2. **Checkpoint 2:** Take your solution for Homework 1 (matrix multiplication) and split it 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.

   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.

3. **Checkpoint 3:** Write a bash script to perform all of the following tasks in a single script:

   (a) Call `make` command to clean your project, 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 several tests to ensure that your implementation of matrix multiplication is working correctly.

   The following requirements are optional:

   (e) Write 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 files as your C implementation.

   (f) Run your C and your Python implementation on the same input files and compare the output files to make sure they match. Time the execution of each implementation for each of the inputs. Make sure to use matrices of different sizes (try some big matrices too!)

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

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

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