Test #1

This is a multi-level grade test. NOTE: It's better to submit a working program than one that does not work at all. Version 1 should be accessible by everyone.

Version 1 – Matrix Multiplication (highest grade C-)

In the first version, write a program to multiply two square matrices. Your code must be modular and adhere to the following. The matrices are stored as 2-D integer arrays. I would expect to see a main function, a function to load a matrix, a function to multiply two matrices, and a function to output the results. I suggest void functions except the main function, it should return an int. NOTE: a void function has parameters but does not return a value.

The 2-D arrays are dynamically defined based on the size given in the file.

The two matrices to multiply are found in file as specified below. The user of your program will specify the file names on the CLI. Example: If you named your executable program, mult, and each matrix was stored in files named A and B, the command ./mult A B C would multiply A and B and store the result in C.

Add a timer to determine the amount of time taken to perform the multiplication. ONLY time the multiplication function. Please use gettimeofday() to report the runtime.

Errors to report?

1) In the main, the program should confirm the correct number of arguments have been entered. If it fails, then output an error message to stderr and shut down with exit(1).

2) While loading a matrix or outputting a matrix, if there is a problem with the file, the program should output an appropriate error message to stderr and shut down with exit(2).

3) The program should confirm the two matrices can be multiplied and shut down the program using the exit(3) function and send an error message to stderr if the matrices are not compatible.

The format of the input file (the output file should be the same setup)

line 1: Size of the first matrix (one integer since the matrix will be square).
Refer to the int as N.
The next **N** lines: each line contains **N** integers, separated by one space.

NOTE: In this version you may assume that the maximum size of each matrix on input will be 1000x1000. For a penalty, you may use a static array instead of a dynamic array.

Version 2 – Thread Matric Multiplication (highest grade - C+/B-)

All of version 1 plus thread the multiplication using openmp.

You must clearly document exactly what you are trying to accomplish with your approach to threading the multiplication function.

Version 3 – Matrix Multiplication. (highest grade - B level)

Using Version 1, add a step in the multiplication process that utilizes openmpi. Implement matrix multiply where the problem is broken down into 8 smaller multiplications based on the top level of the recursive multiply function presented during class. A short synopsis of what is expected, the boss uses the recursive mult program to carve up the data at the top level into 8 multiplications then uses openmpi to distribute to 8 workers. Each worker performs multiplication as found in version 1 on the data it receives from the boss and send back a solution. The boss does the final add of T to C as defined in the algorithm then the main function calls the function to output the solution (or save to a file).

Version 4 – Matrix Multiplication. (highest grade - A level)

All of Version 3 with a new worker. The worker performs a threaded (using openmp) mult procedure.

Documentation and style

- You must exercise good programming style i.e. no global variables, modular code, meaningful function and variable names, etc.
- Provide an opening comment (top of source code) which includes basic information with **labels** as given. Remove the **<...>** in the following when your provide the stated information, keep the labels.
    1. Program name: <program name >
    2. Author: <Your name>
    3. Class: CSC-4310
    4. Date: <the date you started writing a solution>
    5. Location: <path to your file> NOTE: all programs should be stored on your department account. All programs will be tested based on a Ubuntu workstation unless noted otherwise.
    6. A general comment about the nature of the program.
    7. Other comments as needed for clarity, ie. the layout of a data file.
    8. Provide instructions on how to compile and execute your program.

```
/* Program Name: mult.c
 * Author: R. Shore
 * Class: CSC-4310
 * Date: 8-23-2023
 * Location: ~/HPU/csc4310/assignments. (note: the ~ represents your
             home directory.
 * This program multiplies two matrices and outputs the product matrix.
   ...
 */
```

- Use function prototypes and make the **first** function in the file the main function.
- Each function should focus on one task and be a reasonable length, approximately 20 lines of code. Add header comments to each function describing the function name, purpose and the input/output for the function, i.e. name, purpose and pre/post conditions.