This is a multi-level grade assignment. 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 (C+/B- level)

In the first version, write a program to multiply two square matrices. Keep it simple, this will get you coding skills flowing again. 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 multiple two matrices, and a function to output the results. Make void functions except the main function, it should return an int. NOTE: a void function has parameters but does not return a value.

The two matrices are found in one file as specified below. You are to multiple the first matrix in the file by the second matrix in the file. The program should confirm the two matrices can be multiplied and shut down the program using the exit() function and send an error message to stderr if the matrices are not compatible.

The format of the input file will be

line 1: Size of the first matrix (two integers). Refer to them as **N1** and **M1**. **N1** is the number of rows in the matrix and **M1** is the number of columns.

The next **N1** lines: each line contains **M1** integers, separated by a space.

The next line after the first matrix: two integers, the number of rows and columns of the second matrix. Refer to the two integers as **N2** and **M2.**

The next **N2** lines: each line contains **M2** integers, separated by a space.

NOTE: Use I/O redirection. Also, in this version you may assume that the maximum size of each matrix on input will be 100x100 and thus you can use a static array definition for the input and output matrix.

Version 2 – Matric Multiplication – the user needs more control (B+/A- level)

We will need some control over what files are processed without recompiling the code. Incorporate argv/argc to handle input arguments to the program on the command line. Let's start with the user indicating the files that contains the data. If you have not experienced argv/argc, here is a basic guide. From version 1, split the data between two files. Each file contains the size and the matrix data. Suppose the name of the compiled program is **mult** and the name of the input files are **matrix1 and matrix2**. When the user executes the program, they would enter

mult matrix1 matrix2 product

then hit return. In the main, if the program is started correctly and argv/argc are correctly established in the program, **argc** should be **4** and **argv[0]** = "mult", **argv[1]** = "matrix1", and **argv[2]** = "matrix2", and **argv[3]** = "product". At this point, you are loading the matrices from the file "matrix1" and file "matrix2" and then placing the results of the multiplication in a file named **product** in instead of using I/O redirection.

NOTE: In this version you may assume that the maximum size of the matrix on input will be 100x100 and thus can use a static array definition for the input matrix. All error message should be sent to stderr. I will be using fscanf and fprintf to complete the assignment.

Version 3 – Matrix Multiplication. (A level)

Version 2 but dynamically allocate the 2-D arrays based on the input.

Requirements:

- VERSION 1 should be completed by the next class.
- DO NOT include any data files or executable code with your submission. I will compile and use my own data files.
- Only tar, zip, or gzip files will be accepted.
- You will not receive full credit if your program does not follow the above specifications.
- You will not receive full credit if your code does not follow the run-time specifications.

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.
   ...
 */
```

- 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.