Programming Project 2
EE312 Fall 2017

General: The purpose of this project is to get some experience with arrays, pointers and memory management. Mastery of these concepts is critical to C programming. Unless you really know what you're doing with pointers and memory allocation, you are a danger to society (well, figuratively speaking, we hope). When I wrote this assignment, I presumed that you understood the mathematical construct of a Matrix, and the calculation necessary to multiply two matrices together.

Your Mission: Edit the file "Project2.cpp" and implement the functions multiplyMatrices and multiplyMatricesPtr.

Stage 1, Matrix Multiplication: Recall the mathematical definition of a matrix product. Given an MxN matrix A (M rows and N columns), and an NxK matrix B, calculate the MxK result matrix C as follows:

Each element $C_{ij} = A_{i0}B_{0j} + A_{i1}B_{1j} + A_{i2}B_{2j} + \ldots + A_{i(n-1)}B_{(n-1)j}$

Every element of C must be computed this way. So, we'll need two nested **while** loops, one for i (which goes from 0 to M, the number of rows in A), and one loop for j, (which goes from 0 to K the number of columns in B). Nested at the innermost level will be yet another **while** loop (I call mine the "k-loop") which goes from 0 to N and calculates the sum for each C_{ij}.

Your function should have these three loops, one nested inside the other. You must, however, explicitly code the function to use row-major ordering for the matrix storage. That means that A_{ij} is stored in the location *a*[*i * a_cols + j*] where *a_cols* is the variable holding the number of columns in A (N in the discussion above). The matrices B and C are similarly stored in the arrays *b*[] and *c*[] respectively. For your convenience, the code you are given for *multiplyMatrices* defines the variables *a_rows*, *a_cols*, *b_rows*, *b_cols*, *c_rows* and *c_cols* (well, some are parameters, others are defined as local variables, some may not be there). **You may not need to use all these variables. If you decide not to use them, please delete the variable definitions.**

Stage 2, Matrix Multiplication with Dynamic Matrices: Implement function multiplyMatricesPtr that works with dynamic matrices. Each dynamic matrix consists of an array of pointers such that each element in the array points to one row of the matrix. (See class materials for details.) Both the array of points, as well as each row, are dynamically allocated. As the result of multiplyMatricesPtr function, you should return a new **dynamic matrix** (of appropriate size) that contains the result of multiplication. We will "free" your matrix, so if you do not allocate appropriate size (or if you change the format of the matrix), the program will crash.

Stage 3, Matrix Transpose: Implement function transposeMatrixPtr. Same as in the previous stage, the resulting matrix should be dynamically allocated. The format of the matrix is described in the previous stage. We will "free" your matrix, so if you do not allocate in the format that we expect the program will crash.

Testing
You may develop your code any way you like, but the final testing should be on kamek (not luigi, or any of the other 64-bit servers), using our `Makefile`.

FAQ
Q1: What does running `'make'` do?
A: make, without any arguments, generates your executable called `proj2`. `make test` generates `proj2` and then executes it.

Q2: What does make clean do?
A: `make clean` removes all your object code and executable files (`proj2`), so that you can start your compilation and linking afresh.

Q3: My code is working on my machine, but not on kamek.
A: Remove the `--std=c++11` flag from the `Makefile` wherever it appears.
Try running `'make clean'` to remove all your generated files before running `'make'` again.

Q: I am seeing a warning about files being time-stamped in the future.
A: Some of the machines have a whacky clock. `kamek` seems to be OK, so run your code there.
Q4: Don't we get sample test cases for all the parts?
A: These might be provided later, so watch for later postings from us. You will still have time to fix your code in case your code fails with our sample tests.

Q5:In multiplyMatricesPtr, when we allocate the matrix C=A*B, should we perform error checking to see if malloc succeeded? If it fails, should we abort the program? Or can we just assume malloc will succeed?
A: Assume that malloc will succeed.

Q6: Can we get more test cases for stage 2?
A: Not from the instructors, no. You must make your own, and you are welcome to share them on Piazza.

CHECKLIST – Did you remember to:

- ☐ Re-read the requirements after you finished your program to ensure that you meet all of them?
- ☐ Make sure that your program does not need modifications of main.cpp to work?
- ☐ Make sure that your program passes all our testcases?

- ☐ Seal all memory leaks?
- ☐ Make up your own testcases?
- ☐ Upload your solution to Canvas (Project2.cpp)? If you have multiple files, you should zip them up into a file called Project2_<EID>.zip or .gzip or .gz before uploading. Please consult with us if you have multiple files.
- ☐ Download your uploaded solution into a fresh directory and re-run all testcases?