Programming Assignment #2

The purpose of this project is to get some experience with arrays, pointer 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. If you do not understand how to multiply matrices, please ask for help!

Setup

Begin by downloading the starter files provided on Canvas and opening them in VSCode. You will be completing the functions described below in `Project2.cpp`.

Part a)

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

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

Every element of C must be computed this way. So, we'll need two nested 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 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.**

Part b)

Matrix Multiplication with Dynamic Matrices: Implement the 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. **Do not free anything in your code. For this assignment only, we will be doing the freeing for you.**

Part c)

Write the function `createSubMatrix` - Given an M x N matrix, return a matrix of size (M-1) x (N-1) that results from removing row `row_x` and col `col_y` from the given matrix. Same as in the previous part, the resulting matrix should be dynamically allocated. The format of the matrix is the same as described in Part B. 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. **Do not free anything in your code. For this assignment only, we will be doing the freeing for you.**

In all cases, your functions should **NOT** alter the input matrices.

Submission

You should submit `Project2.cpp` to the corresponding Gradescope assignment. Do **NOT** change the name of `Project2.cpp` and do **NOT** submit other files. Remember to read the requirements to make sure you meet them, that you did not modify main.cpp, that your program passes our test cases, and that you sealed all memory leaks.

FAQ

Q1: In multiplyMatricesPtr, when we allocate the matrix C=A*B, should we perform error checking to see if malloc succeeded?
A: Assume malloc will always succeed.

Q2: Can we get more test cases for part b?
A: No, but you are welcome to make your own.

Q3: May we assume the given matrices can be multiplied?
A: Yes

Q4: Can we assume that the matrices passed into the multiply functions are at least 1x1, and that the matrices passed into `createSubMatrix` are at least 2x2?
A: Yes, this is the correct assumption.