Matrix Calculator

Goals

- Write, compile and test a matrix calculator program
- Review the concept of pointers, and multi-dimensional dynamic arrays
- Create a simple makefile
- Write a robust input validation.

The purpose of this lab is to get familiar with the software environment we will be using this term, and review some C++ concepts to get prepared for future projects and labs. This lab might be very time-consuming since there are a lot of concepts you need to pick up.

For this course, we will be compiling code on the flip server. It is advised to use vim editor on the flip server to program, but you are allowed to use other IDEs or text editors. **Please test the program on flip server before turning it in.**

Requirements

Files:

For simplicity, there are only 2 sizes of matrix for this program: either a 2x2 or a 3x3 matrix.

Create a matrix calculator program that consists of **5 files**.

The program should have two functions:

- **void readMatrix()**
- **int determinant()**

Each function should have its own header files (.hpp) and source code (.cpp). Including the main file that contains main function to run the program, there should be **5 files** for the program to be compiled and run properly.

main function:

The main function should contain following steps:

1. Ask the users to choose the size of the matrix (2x2 or 3x3).
2. Dynamically allocates the memory space for the matrix using readMatrix() to prompt the user to enter 4 or 9 integers to fill the matrix
3. Calculate the determinant using determinant().

4. Display both the matrix and the determinant to the user.

Note: please display the matrix in a square format, do not display it in a line.

5. Free the dynamically allocated memory

void readMatrix():

The readMatrix() function has two parameters:

- A pointer to a 2D array
- An integer as the size of the matrix

The function should prompt the user for all the numbers within the matrix, that means for 2x2 matrix, it should ask the user for 4 numbers, and 9 numbers for a 3x3 matrix. Because the function takes a pointer to the 2D array, it should not return anything.

int determinant():

The determinant() function has two parameters:

- A pointer to a 2D array
- An integer as the size of the matrix

The function takes in the 2D array, which contains the value inside the matrix, and calculate the determinant. Afterwards, the function should return the determinant.

Note on how to calculate determinant of 2x2 and 3x3 matrices:

https://www.mathsisfun.com/algebra/matrix-determinant.html (Links to an external site.)Links to an external site.

Dynamic Memory

The program needs to **dynamically** allocate the 2D array in main function, and free the dynamically allocated memory when it is no longer in use. The following link can provide some hint:

http://stackoverflow.com/questions/936687/how-do-i-declare-a-2d-array-in-c-using-new (Links to an external site.)Links to an external site.

The program must use dynamic memory for this lab. This will help you understand how standard containers work.

If memories are allocated but never freed during program runtime, it will cause **memory leak**. We will enforce the memory leak check from this lab to future projects/labs. If you have memory leaks for lab 1, lab 2, and project 1, you will not be penalized for it. From lab 3 and project 2, if you still have memory leaks, TAs will take points off up to 10% of the grade.

Input validation

In this lab, you can assume users will only input integers when asked, and TAs won't test the program using other types of input, but what if they do?

Input validation is very important, it prevents the program from crashing when user input the wrong type of input. For example, inputting a character "c" when prompted an integer.

Try to write an input validation function that checks the user inputs' data type. It should return an error message if the input type does not match what is prompted, and recovers from it by re-prompting for input, until the user enters the correct data type.

Input validation is not required for this lab, but you need to start learning it since it will be required in the future.

Makefile

Once your program is working, create a makefile to build the program. You do not need a complicated makefile; you only need a makefile that have following two target (functionalities):

- Compiling, by typing "make" in the command line, the makefile compiles the program using g++.
- Removing executables, by typing "make clean" in the command line, the makefile removes all the files created during compiling.

To get help on makefile, please watch the lecture videos, check out the make help sessions in the resource module on Canvas, and join the group discussions of makefile on Piazza.

What you need to submit:

- 5 files for the program
- makefile

Please put the 6 files into a zip archive. The name of the zip file should follow this format:

Lab1_LastName_FirstName.zip

Please submit **ONLY** the zip file on Canvas by the due date.

The following command is for zipping all the files: (entire command in one line)

zip –D Lab1_LastName_FirstName.zip file1.cpp file2.cpp file1.hpp file2.hpp main.cpp makefile

There should be no internal directories in the zip file, meaning the zip file should not create a new folder when unzipped (Please test the unzipping on flip server, since TAs will be grading on flip server)

Remember to back up your files in your local computer frequently if you are developing on flip. If you delete your files by accident on flip, you might not be able to get it back.