CMSC 313
Grisha Goldberg
4/11/2025

HW #7 - Github

File 1: addmats.cpp

This is the file with the most operations going on. First, we see that two ints get declared, ARRAY_SIZE which initialized to 5 and DEFAULT_RETURN_VALUE which is 1.

Then, the class MatrixFunctions gets declared which only has public members.

The only member variable is array_size, which is set to the parameter from the constructor.

The last thing within this class is the addMatracies function which takes in three int pointers: 1 for each matrix and the result. Within the function, we have a for loop which iterates ARRAY_SIZE number or times, and sets the sum of each element in matrix 1 and 2 to the result index.

Now, let's look at main.

Withing main, we have the three matrices declared. The first element in matrix 1 gets initialized to 1, and for matrix 2, the first element is initialized to 2.

Then, we create an object instance named funcs for the MatrixFunctions class and send the argument of ARRAY_SIZE to the constructor. Lastly, the addMatricies function is called, and uses the three declared matrices as arguments.

File 2: addmatsSubr.c

Functionality wise, this file accomplishes the same exact thing as addmats.cpp. The only thing is, because it is a C file and not C++ file, it cannot have a class. So instead, it only has two things outside of main:

1. the two define statements, identical to addmats.cpp
2. the addMatracies function. This function is identical to the addmats.cpp version except for one difference. Because there is no class, there is also no member variables, which means the array size used in the for loop is just the in ARRAY_SIZE from the very top #define statement.

In main, the same three arrays get declared: result, matrix 1, and matrix 2. Plus, just like in addmats.cpp, matrix 1 gets its first element initialized to 1 and matrix 2 to 2. Below these declarations, no objects are made and instead, addMatricies is directly called, using the three arrays as parameters.

CMSC 313
Grisha Goldberg
4/11/2025

File 3: addmatsSimple.c

This is the least code out of all files. The only thing outside of main are the two default statements, identical to the other two files. Within main, we have the three same arrays declared. The big difference is that instead of calling a function to do the add math, the for loop is done within main. This is very minimalistic since it only has the declarations and the for loop.

Examining the differences in Assembly code:

When looking at the assembly for the three versions of the program, a few things stand out right away. All of them follow a similar structure with a section labeled _main, and they use a lot of the same kinds of instructions over and over, like movl, addl, and pushl. They also include labels like L2 and L3, which seem to be used for loops. Each one starts and ends with a kind of setup and cleanup routine that looks almost identical, using instructions like pushl %ebp at the beginning and leave and ret at the end. The differences are more obvious when you compare how long and complex each one looks. The simple C version is the shortest and most straightforward. Everything happens in one place and it's easier to follow just by looking at it. The version with a subroutine breaks the code into two parts, which adds a bit of complexity but still feels organized. The C++ version is clearly the most complicated. It has much longer and more confusing function names, and there’s extra setup that looks like it’s creating or initializing something before the actual work starts. There’s even a part that seems to be tied to a class constructor, which doesn’t appear in the other two. It’s hard to tell what every line does, but it’s pretty obvious that the simple C version is more compacted, while the C++ version is more layered and harder to follow.