Programming Project #1

Christopher Koepke

CS 4080.03

April 2, 2022

For each part, my multiplication algorithm is what my algorithm analysis teacher calls the naive way. I use three loops to do my multiplication and summation. The outer loop is for the rows of the first matrix, the middle loop is for the column of the second matrix, and the innermost loop is for the rows of the second matrix. The algorithm performs the calculation in the same way as if I had done this operation on paper. In each of my programs, I had set a timer for each operation and then displayed the runtime. Using modest sized arrays (around 5x5) all algorithms of each program were very similar in their run time. The run time of a 5x5 matrix multiplication with another 5x5 matrix of random float values in part three was around 4000 nanoseconds, whereas in part four, the run time was around 8000 nanoseconds. This difference is not noticeable from an end-user's perspective but would be more pronounced with larger matrices.

The easiest part for me to write code for was in part one. I did not have any issues passing 2D arrays to methods outside the main method. There were three main categories of methods that I had created for this part: menu display, array input, and matrix operations. The second part was not much more difficult than the first. The only difference is that I passed pointer to a pointer of a 2D array instead of a reference to a 2D array. The algorithms for matrix operations were similar. The real issue I had with this programming assignment was in part three. I kept having to rewrite my class's methods to get parameters to work correctly. I had rewritten my algorithms multiple times to also get the correct calculation and output. I attempted to do an operator overload for the equals sign but just kept getting complication errors. I attempted to get that operator to call the copy constructor, but that did not work. I tried to get the operator to a copy operation inside the method, but again more problems. In my final version, I abandoned that operator and instead just had the other overloaded operators store the results in a second 2D array. This worked correctly but is not the way I had wanted to do it. My original Idea was to have the syntax looking similar to matrixC = matrixA + matrixB. My final version instead looks like this, matrixA + MatrixB, with the next line of code calling the first object's print method to display the results from the previous operation. Part four was similar in difficulty to part two, because I did not have to use pointer to a pointer in java, it was easier to setup my algorithms.

With my limited experience writing code in both java and c++, I did not have any major difficulties with this project, other than from part three when dealing with pointer to a pointer in a class. I did notice some more minor differences besides the obvious ones like using cout in c++ versus using system.print in java. The most notable difference for me was finding a way to get the runtime of my algorithms. In that aspect, java is way easier to accomplish that, through the use of System.nanoTime() at the start and end of my algorithm and then calculate the difference. For c++, I used the chrono library. This resulted in me writing three lines of code. They were

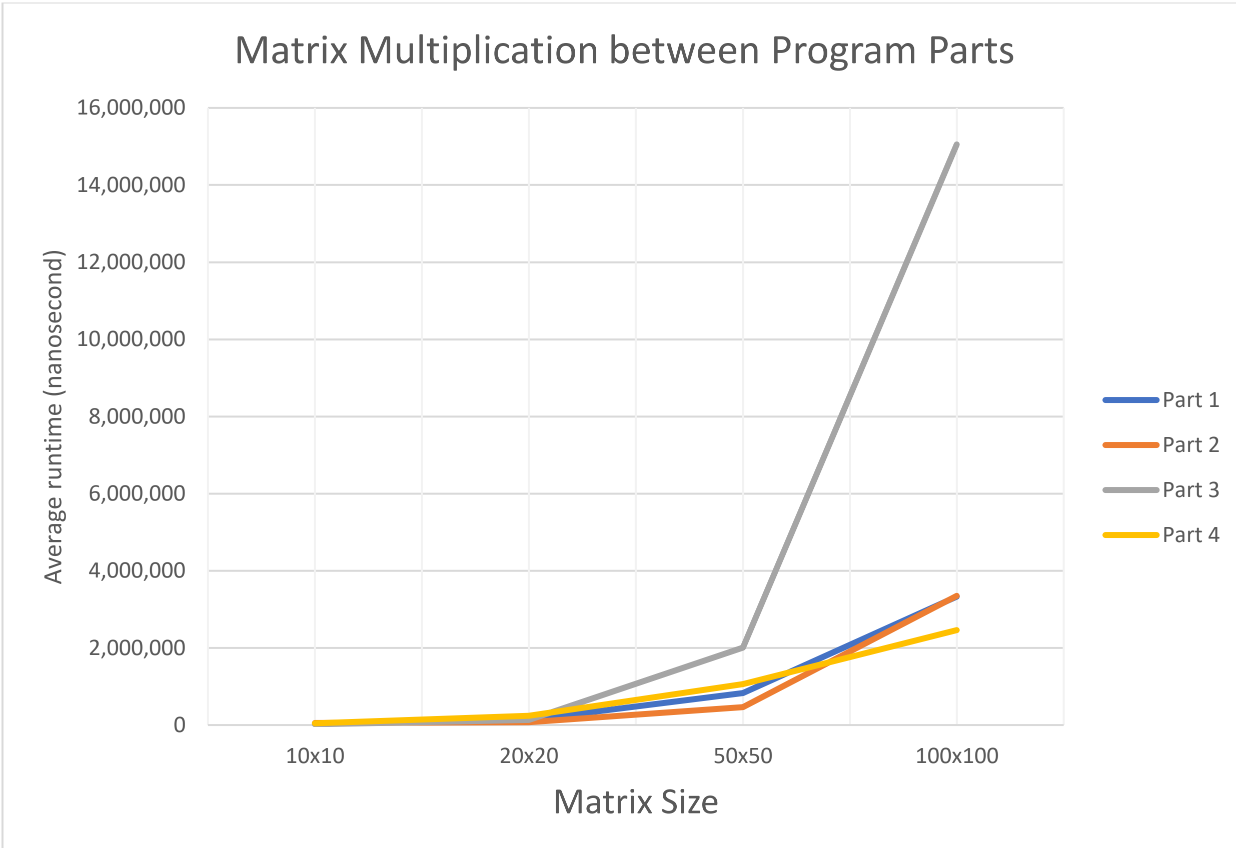
```
auto start = std::chrono::high_resolution_clock::now();
auto stop = std::chrono::high_resolution_clock::now();
```

auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start);

with start and stop at the beginning and end of my program. I would than need to do duration.count() in order to display the runtime. For the first two parts, this was not much of an issue, but for the third I kept getting compilation issues or a zero-nanosecond runtime. I did eventually get it to work correctly but that was one of the most time-consuming issues I had besides the pointer issue stated previously. I am more comfortable writing code in C++ than in Java, but my experience is similar in both. I understand the concepts of pointers and references but did not use pointers often in my previous C++ programs.

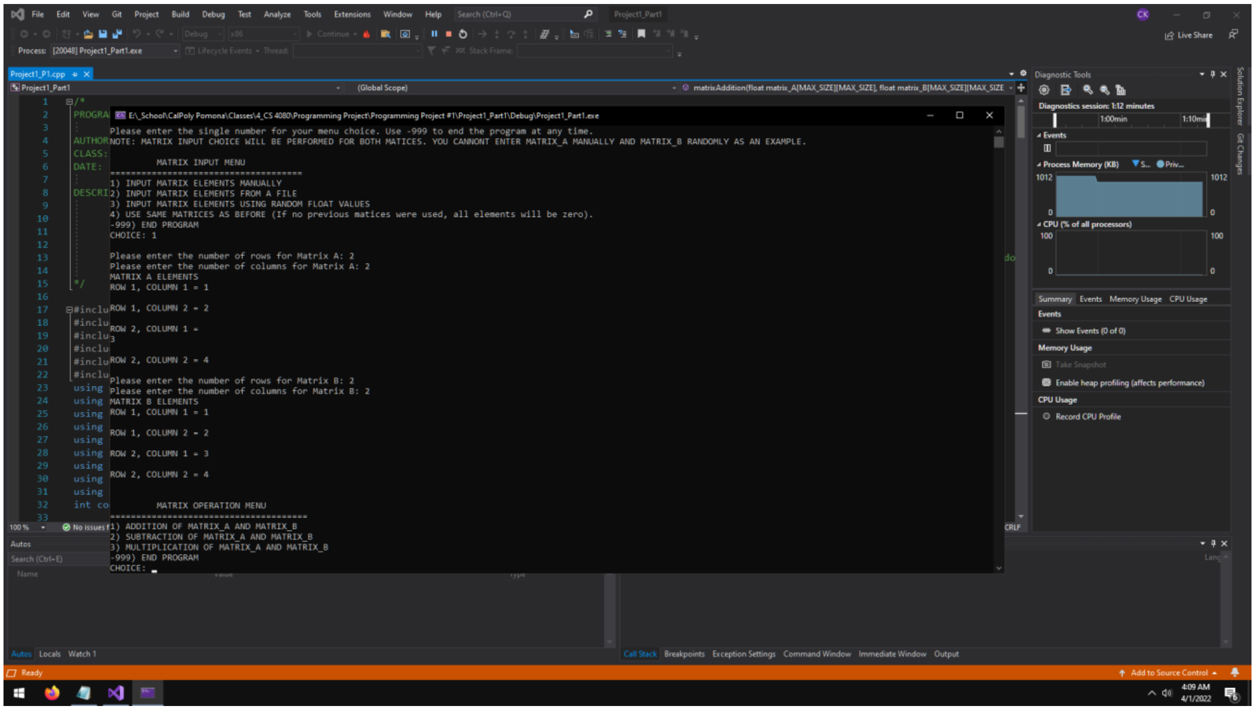
In my program, each matrix is fill with random decimal values of type float. The two matrices will have different element values, but I do not believe that will skew the results of the multiplication test. Each part is an average of 3 runs of the corresponding matrix size with new matrices being created for each run. For part one, I notice that my run time for each operation were similar, even when increasing the matrix size. Part two had a more noticeable difference between operations but still not very large. There was also a small but noticeable difference when running the same operation with different matrices of the same size. Part three is definitely much slower when the sizes increase. My timers only take account of the time of the actual operation, not the calling of the function or other assignments, except in part two where the timer is counted from before and after the assignment of the pointer to the operation. Part four had more interesting results. Each run kept getting faster. If I took a larger average, I am sure that it would end up being the fast. I am also surprised at how quickly the time increases in part three as the size of the matrices increase. I am sure there is some optimization happening behind the scenes in the java program. From this question I did a quick search on the internet and found that it does indeed do some kind of optimizations each time the program runs without it shutting down. One that I noticed is that some interpreters perform a kind of runtime analysis that will convert the most taxing portion of a program from bytecode into machine code.

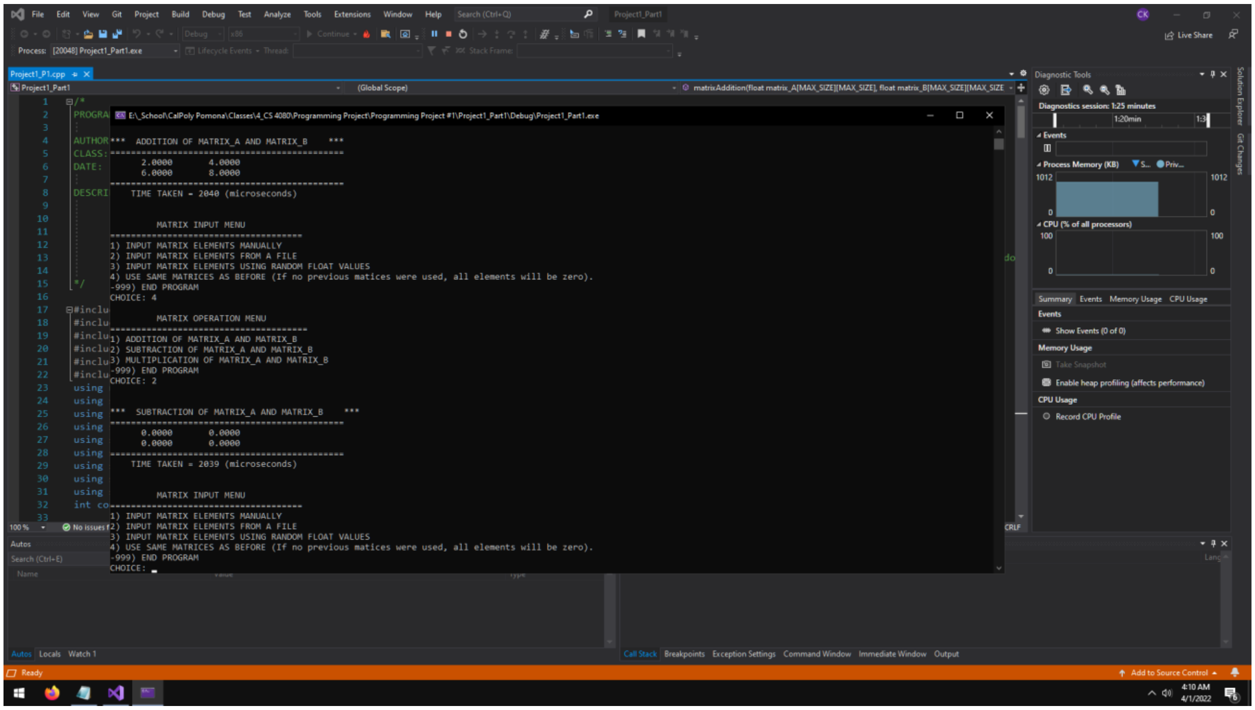
Matrix Size	Part one (nanoseconds)	Part two (nanoseconds)	Part three (nanoseconds)	Part four (nanoseconds)
10x10	33,155	54,706	20,333	45,733
20x20	132,897	76,933	134,366	240,366
50x50	826,643	465,633	2,010,133	1,059,966
100x100	3,329,521	3,350,000	15,052,666	2,461,466

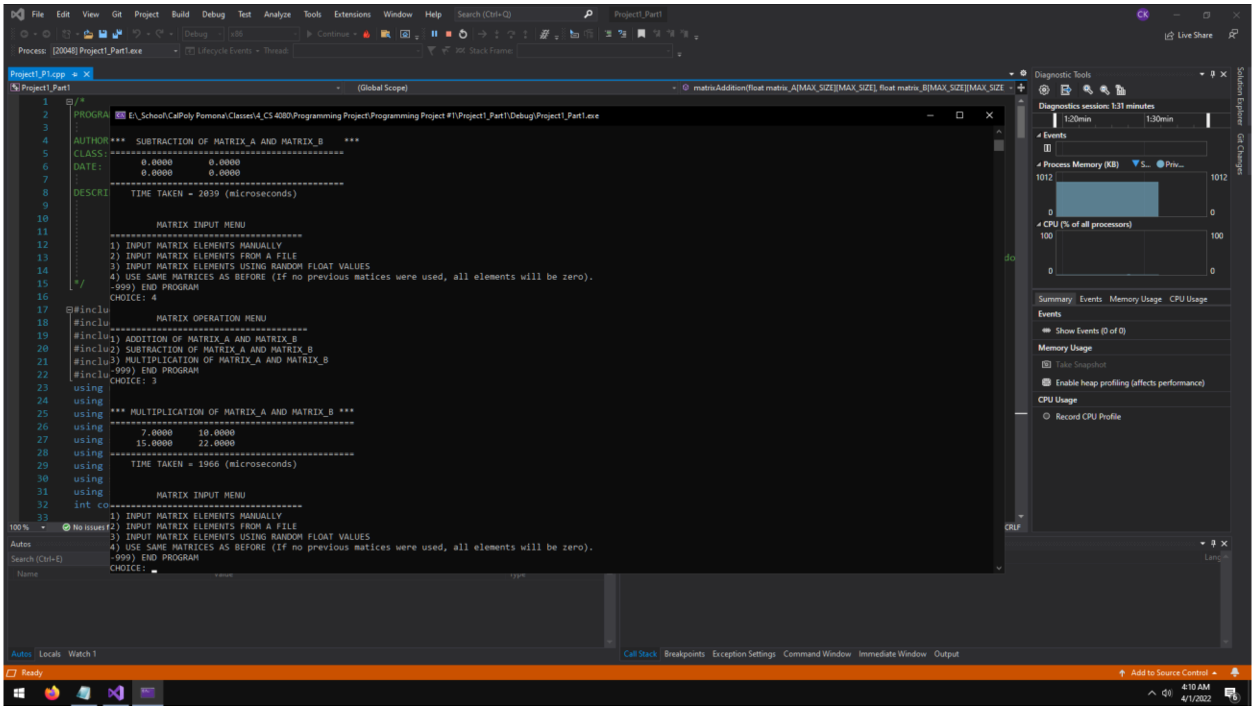


Matrix Multiplication between Program Parts
16,000,000
14,000,000
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Average runtime (nanosecond)
10x10
20x20
50x50
100x100
Matrix Size
Part 1
Part 2
Part 3
Part 4

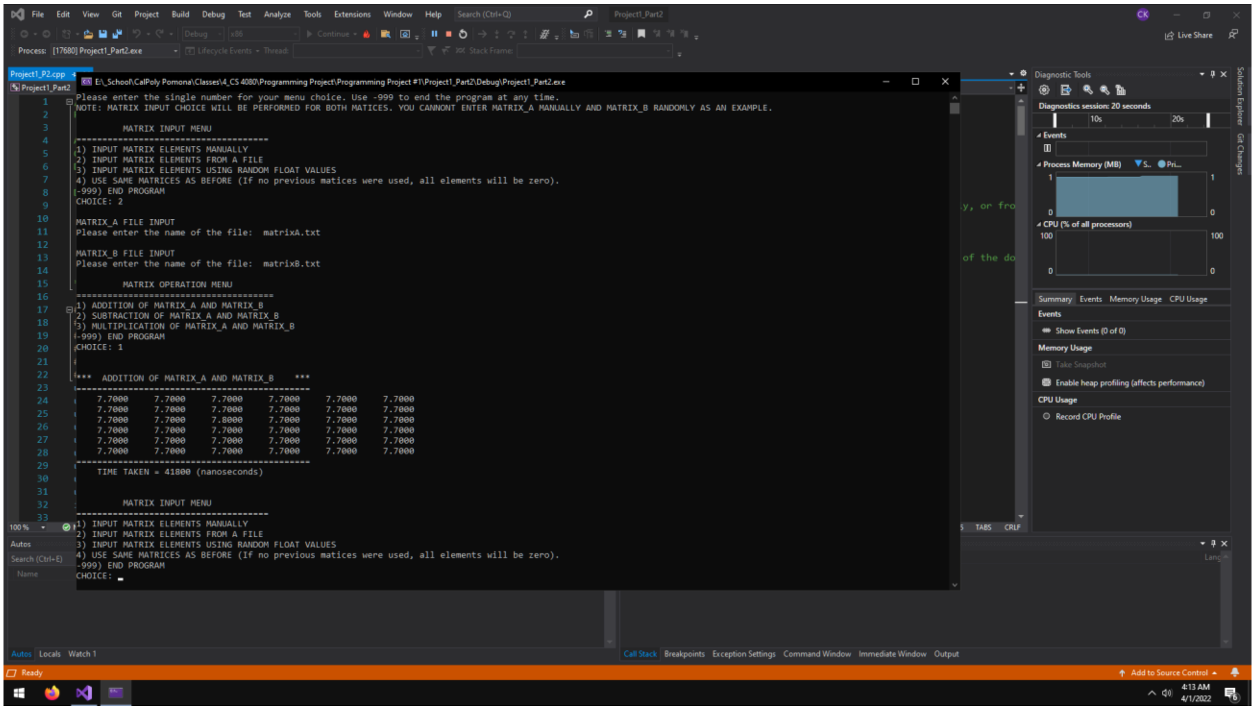
<u>PART 1 SCREENSHOT</u>

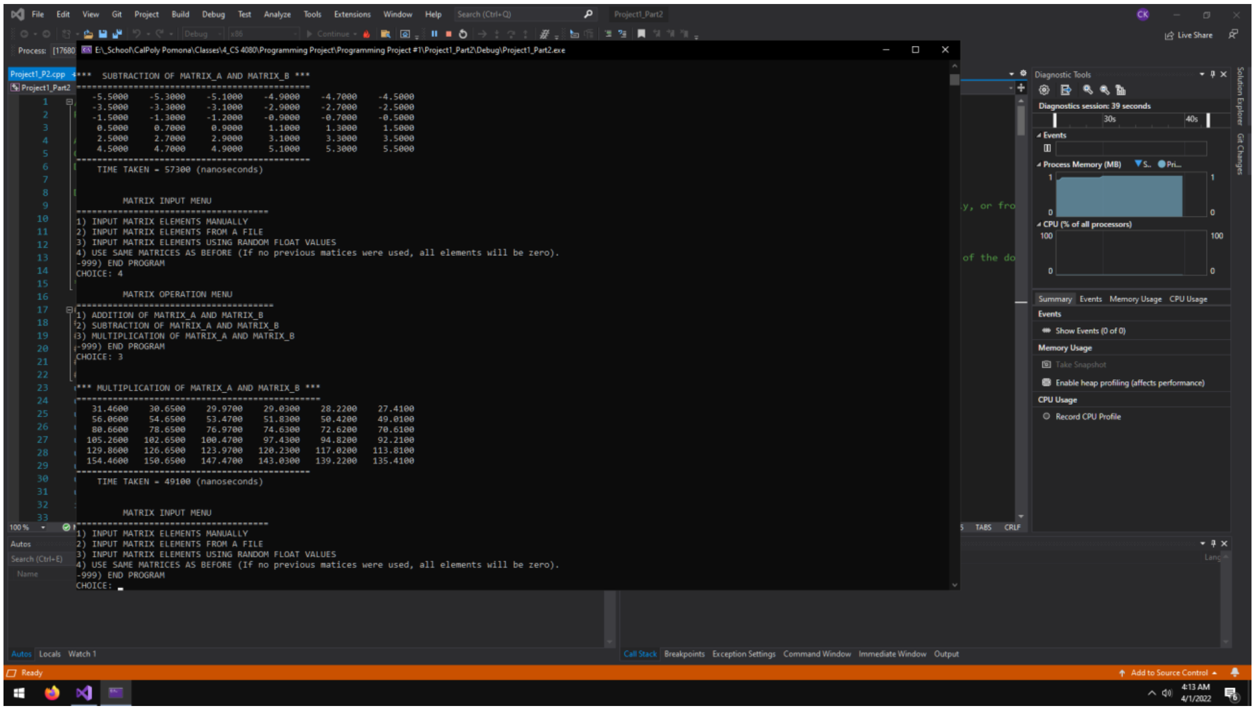




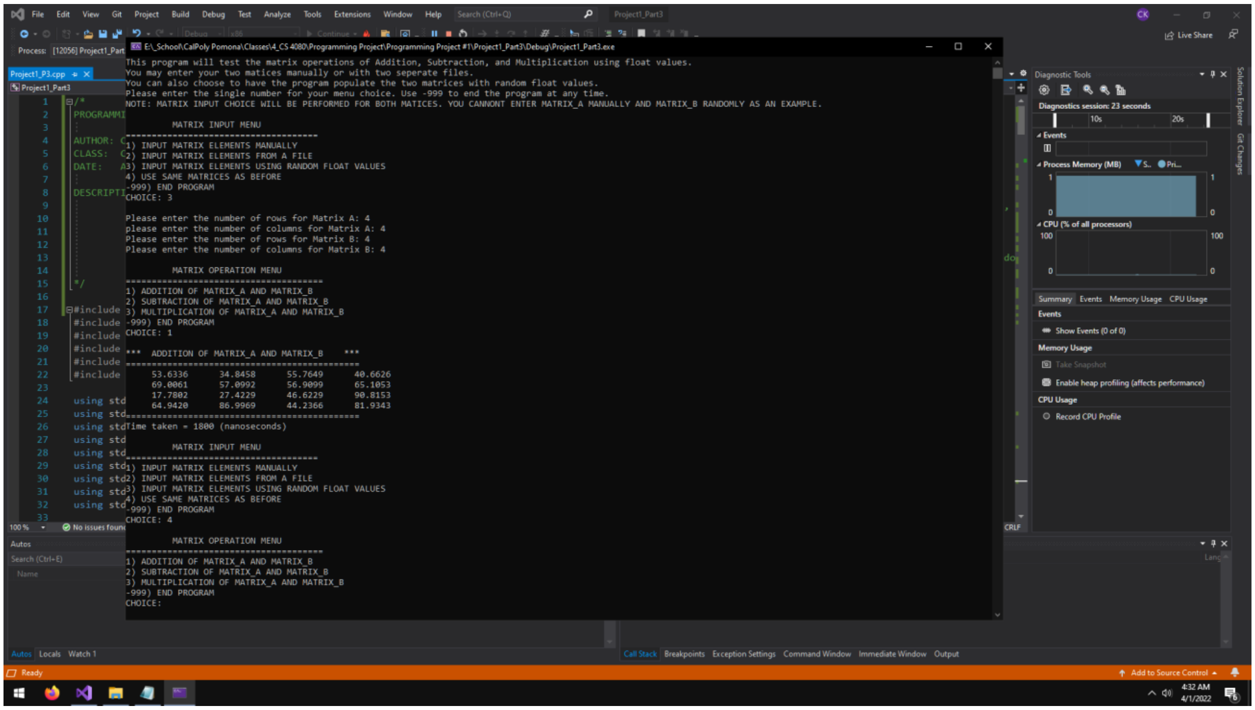

E:_School\CalPoly Pomona\Classes\4_CS 4080\Programming Project\Programming Project #1\Project1_Part1\Debug\Project1_Part1.exe
*** SUBTRACTION OF MATRIX_A AND MATRIX_B   ***
0.0000   0.0000
0.0000   0.0000
TIME TAKEN = 2039 (microseconds)
MATRIX INPUT MENU
1) INPUT MATRIX ELEMENTS MANUALLY
2) INPUT MATRIX ELEMENTS FROM A FILE
3) INPUT MATRIX ELEMENTS USING RANDOM FLOAT VALUES
4) USE SAME MATRICES AS BEFORE (If no previous matices were used, all elements will be zero).
-999) END PROGRAM
CHOICE: 4
MATRIX OPERATION MENU
1) ADDITION OF MATRIX_A AND MATRIX_B
2) SUBTRACTION OF MATRIX_A AND MATRIX_B
3) MULTIPLICATION OF MATRIX_A AND MATRIX_B
-999) END PROGRAM
CHOICE: 3
*** MULTIPLICATION OF MATRIX_A AND MATRIX_B ***
7.0000   10.0000
15.0000   22.0000
TIME TAKEN = 1966 (microseconds)
MATRIX INPUT MENU
1) INPUT MATRIX ELEMENTS MANUALLY
2) INPUT MATRIX ELEMENTS FROM A FILE
3) INPUT MATRIX ELEMENTS USING RANDOM FLOAT VALUES
4) USE SAME MATRICES AS BEFORE (If no previous matices were used, all elements will be zero).
-999) END PROGRAM
CHOICE:

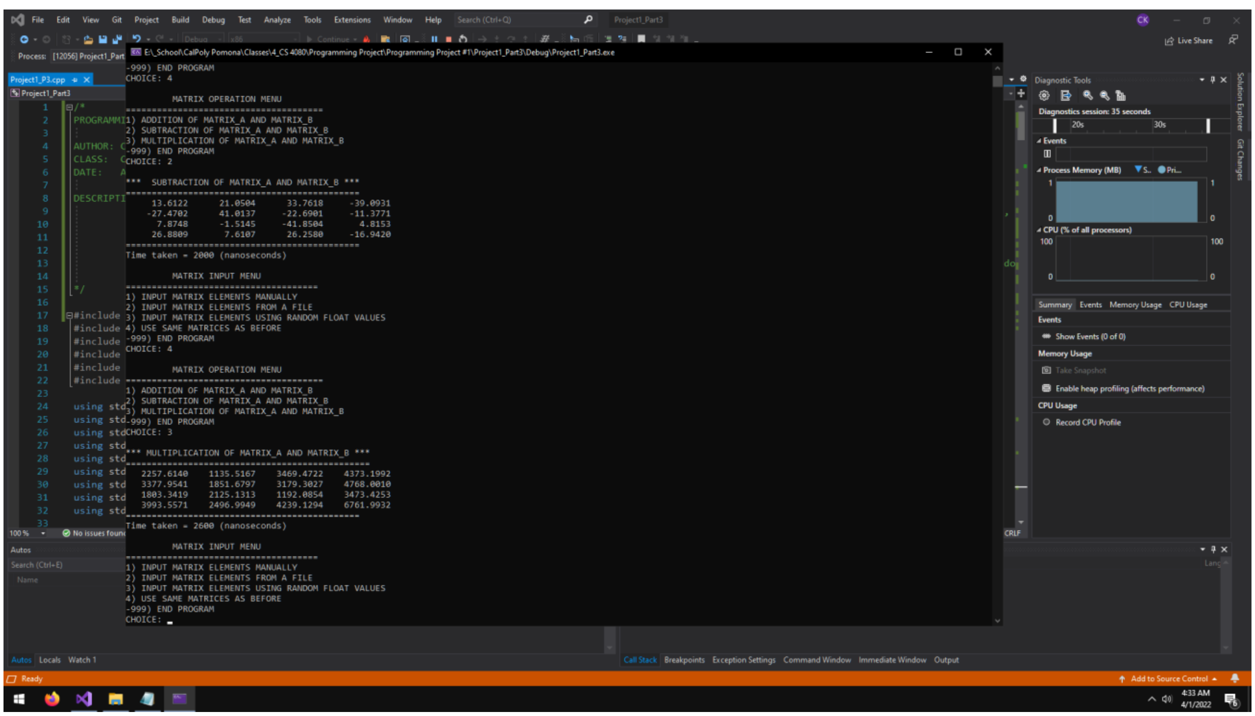
PART 2 SCREENSHOTS



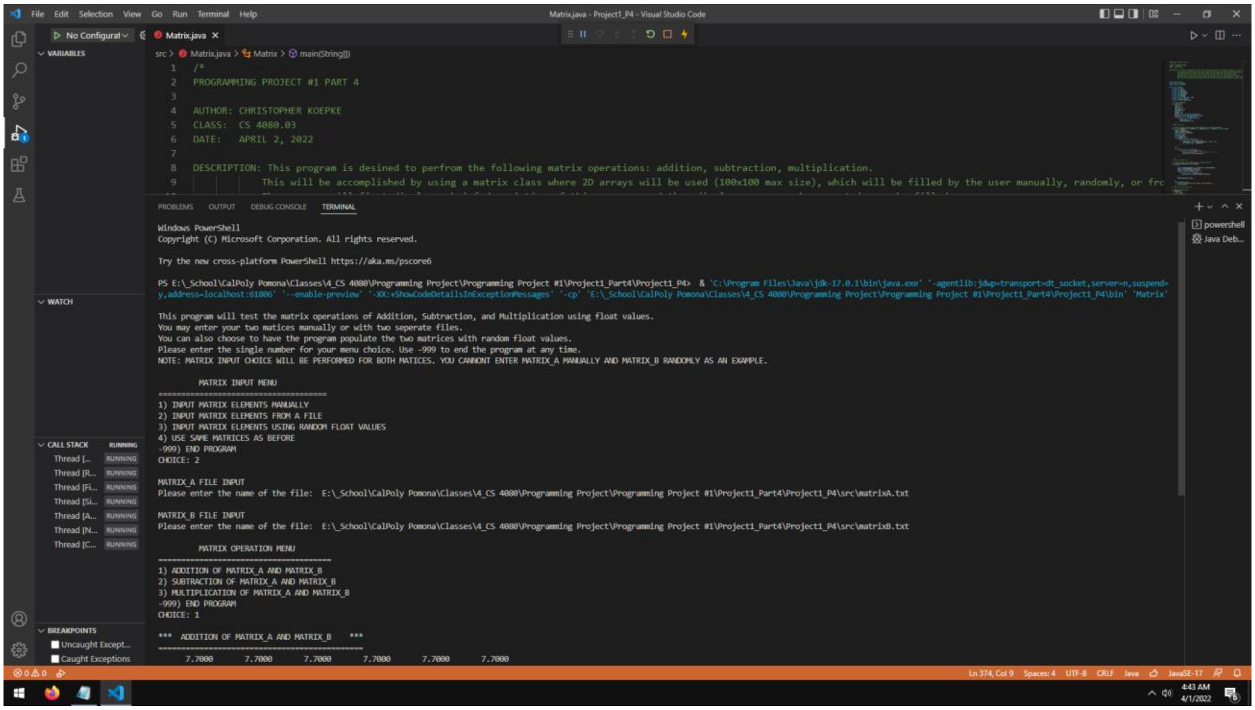


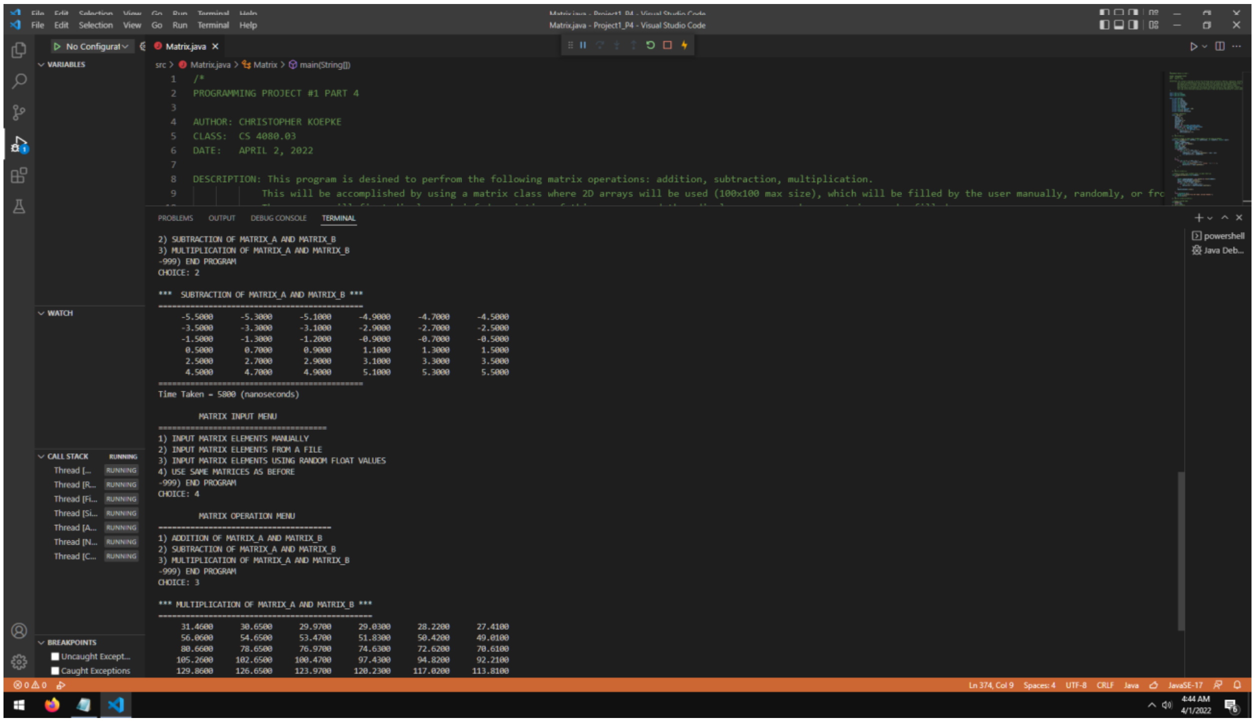
PART 3 SCREENSHOTS

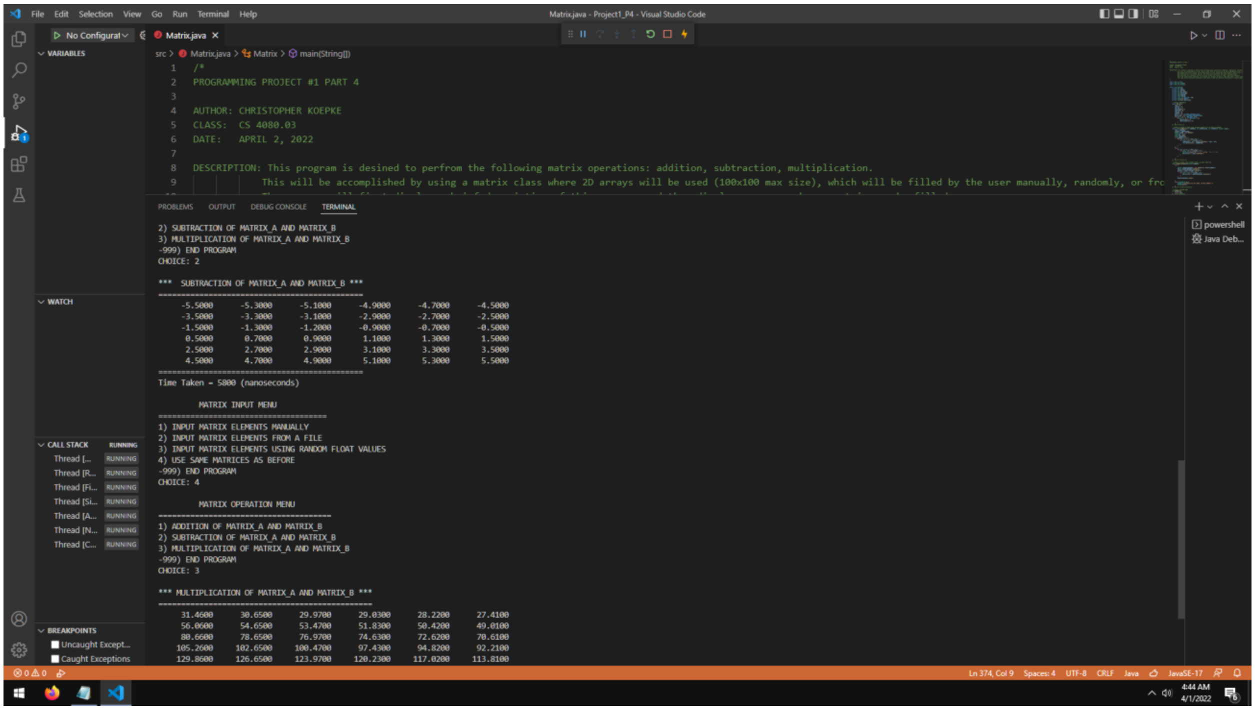




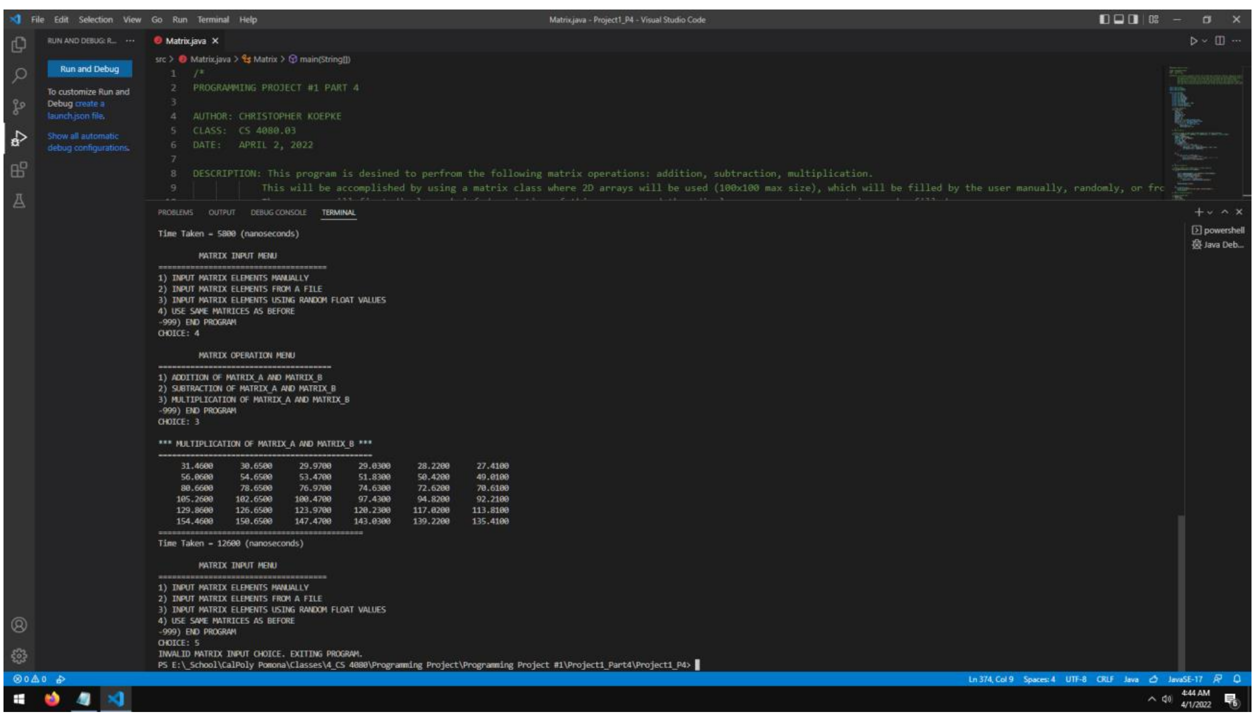
PART 4 SCREENSHOTS







Matrix.java - Project1_P4 - Visual Studio Code
/*
PROGRAMMING PROJECT #1 PART 4
AUTHOR: CHRISTOPHER KOEPKE
CLASS: CS 4080.03
DATE: APRIL 2, 2022
DESCRIPTION: This program is desined to perfrom the following matrix operations: addition, subtraction, multiplication.
This will be accomplished by using a matrix class where 2D arrays will be used (100x100 max size), which will be filled by the user manually, randomly, or
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2) SUBTRACTION OF MATRIX_A AND MATRIX_B
3) MULTIPLICATION OF MATRIX_A AND MATRIX_B
-999) END PROGRAM
CHOICE: 2
*** SUBTRACTION OF MATRIX_A AND MATRIX_B ***
-5.5000 -5.3000 -5.1000 -4.9000 -4.7000 -4.5000
-3.5000 -3.3000 -3.1000 -2.9000 -2.7000 -2.5000
-1.5000 -1.3000 -1.2000 -0.9000 -0.7000 -0.5000
0.5000 0.7000 0.9000 1.1000 1.3000 1.5000
2.5000 2.7000 2.9000 3.1000 3.3000 3.5000
4.5000 4.7000 4.9000 5.1000 5.3000 5.5000
Time Taken - 5800 (nanoseconds)
MATRIX INPUT MENU
1) INPUT MATRIX ELEMENTS MANUALLY
2) INPUT MATRIX ELEMENTS FROM A FILE
3) INPUT MATRIX ELEMENTS USING RANDOM FLOAT VALUES
4) USE SAME MATRICES AS BEFORE
-999) END PROGRAM
CHOICE: 4
MATRIX OPERATION MENU
1) ADDITION OF MATRIX_A AND MATRIX_B
2) SUBTRACTION OF MATRIX_A AND MATRIX_B
3) MULTIPLICATION OF MATRIX_A AND MATRIX_B
-999) END PROGRAM
CHOICE: 3
*** MULTIPLICATION OF MATRIX_A AND MATRIX_B ***
31.4600 30.6500 29.9700 29.0300 28.2200 27.4100
56.0600 54.6500 53.4700 51.8300 50.4200 49.0100
80.6600 78.6500 76.9700 74.6300 72.6200 70.6100
105.2600 102.6500 100.4700 97.4300 94.8200 92.2100
129.8600 126.6500 123.9700 120.2300 117.0200 113.8100



Matrix.java - Project1_P4 - Visual Studio Code
Run and Debug
/*
PROGRAMMING PROJECT #1 PART 4
AUTHOR: CHRISTOPHER KOEPKE
CLASS: CS 4080.03
DATE: APRIL 2, 2022
DESCRIPTION: This program is desined to perfrom the following matrix operations: addition, subtraction, multiplication.
This will be accomplished by using a matrix class where 2D arrays will be used (100x100 max size), which will be filled by the user manually, randomly, or
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Time Taken - 5800 (nanoseconds)
MATRIX INPUT MENU
1) INPUT MATRIX ELEMENTS MANUALLY
2) INPUT MATRIX ELEMENTS FROM A FILE
3) INPUT MATRIX ELEMENTS USING RANDOM FLOAT VALUES
4) USE SAME MATRICES AS BEFORE
-999) END PROGRAM
CHOICE: 4
MATRIX OPERATION MENU
1) ADDITION OF MATRIX_A AND MATRIX_B
2) SUBTRACTION OF MATRIX_A AND MATRIX_B
3) MULTIPLICATION OF MATRIX_A AND MATRIX_B
-999) END PROGRAM
CHOICE: 3
*** MULTIPLICATION OF MATRIX_A AND MATRIX_B ***
31.4600 30.6500 29.9700 29.0300 28.2200 27.4100
56.0600 54.6500 53.4700 51.8300 50.4200 49.0100
80.6600 78.6500 76.9700 74.6300 72.6200 70.6100
105.2600 102.6500 100.4700 97.4300 94.8200 92.2100
129.8600 126.6500 123.9700 120.2300 117.0200 113.8100
154.4600 150.6500 147.4700 143.0300 139.2200 135.4100
Time Taken - 12600 (nanoseconds)
MATRIX INPUT MENU
1) INPUT MATRIX ELEMENTS MANUALLY
2) INPUT MATRIX ELEMENTS FROM A FILE
3) INPUT MATRIX ELEMENTS USING RANDOM FLOAT VALUES
4) USE SAME MATRICES AS BEFORE
-999) END PROGRAM
CHOICE: 5
INVALID MATRIX INPUT CHOICE. EXITING PROGRAM.
PS E:_School\CalPoly Pomona\Classes\4_CS 4080\Programming Project\Programming Project #1\Project1_Part4\Project1_P4>