```
#########################
#                       #
# Mathias Eriksen       #
# mterikse@ucsc.edu     #
# Lab 03                #
#                       #
#########################
```

In this lab, the only person I worked with was Rikuu Mikami, and we mostly discussed how to set up the test harnesses, specifically how to check each function's result using a hard coded expected matrix. I also helped him out in understanding how functions work in general, especially the idea of a void function versus one that has return value.

In summary, the main part of this lab that had to be understood was how to iterate through rows and columns within the matrix using a nested for loop. Once I understood this concept, and how to access elements in each array using the counters in both the for loops, it was simple to implement the functionality for each function. Most of the functions just involved looping through matrices and manipulating each element individually to fit the desired behavior. Next, implementing the test harness was simple, but also somewhat repetitive. I eventually created the test harness by running each function individually and then checking it against an expected value I calculated by hand.

My approach to the lab was writing each function individually and checking the results visually by extensively using the MatrixPrint() function in my test harness. I would try adding matrices together and visually check that the value was correct through inspection. Unfortunately, I then had to rewrite the test harness in order to run the tests in the way the lab doc outlined.

The biggest problems I encountered in this lab were when I had some issues with the naming convention of my counters when calling a function within a function. This happened during my first try at the determinant function, but I eventually switched my strategy in approaching this function. Instead, I was just using the determinant formula with elements in the array directly declared as variables. My first approach was an iterative one, and it had some issues with segmentation faults, so I dropped it. Another issue I encountered was checking for delta precision to values, which I initially struggled with due to the greater than/less than operators in the if statements. So, I switched to using fabs() for floating absolute value in order to find the difference between two values that were being checked, and tested that value against the FP_DELTA value. This function required me to include the math.h function, which I hope is ok with the test scripts.

The most difficult function to implement was the inverse function. It involves multiple steps, and uses some of the previously written functions. In order to implement it, I followed a wiki how article illustrating the steps, and by translating each of these steps into c code, I was able to achieve the correct behavior. For this function, I had to map out what I was going to do in comments before writing, and this really helped with my design. When testing, I learned that in order to properly represent a fraction in C, you must add a .0 to the end of each number in order to specify that the value must be divided as a float instead of an int. I enjoyed this part as it brought together all the concepts within the lab quite well.

I probably ended up spending about 4-5 hours on this lab, but it was quite easy for me to implement the functions since I've done work before on arrays and iterating through their elements. Therefore, the theory behind the implementation was quite simple for me to understand, and the hardest part was making sure that the syntax was correctly for C.