C PGM Manipulator

a.) Project Development:

Upon starting the code, the first challenges I faced was dealing with the C language syntax as opposed to an object oriented programming language such as C++, but after I started writing code I noticed that it wasn't too different from what I was familiar with.

Understanding the struct with type unsigned char was a challenge. Deciding whether to use functions like fscanf, fread, or other file reading methods, and facing the same dilemma for output types, posed difficulties. Additionally, I had uncertainty about pointer notation when passing parameters, whether to use a single pointer or double pointer, and whether to use ampersand or asterisk, as it wasn't always clear if I was dealing with call-by-value or call-by-reference situations. Initially, I struggled with using "r" instead of "rb" as the file mode in fopen, and transitioning from reading the header to reading the binary stream proved to be quite challenging.

Memory allocation syntax was another hurdle. Despite copying what was shown in lectures and lecture slides, I had to adapt it to the specific requirements of the program, which took a few tries. On top of that, I needed to figure out how to handle errors when using malloc.

The most significant challenge I faced was with the "rotate_again" function failing. I eventually realized that this was a symptom of a more significant issue. The root problem was that, at the end of my "writePGM()" method, I was freeing the allocated memory unnecessarily. After writing to the file, we don't always want to delete the allocated memory. This issue was directly related to "rotate_again" because I needed to use the previously allocated memory again as a parameter in the second call to "rotate."

b.) Snapshots of Images:

"rotate.pgm" and "threshold.pgm"

c.) Rotating the Image:

The reason why rotating the image twice does not give an upside-down image is because we are not rotating the image how we would conceptually picture it, like in an image editor. In actuality, we are actually performing matrix transpose on the individual pixel values in the 2D array. When I turn my head to look at the rotate_pgm image, I notice that it is not turned 90 degrees counterclockwise, in fact, it is actually vertically mirrored and turned on its side, which is another proof of matrix transpose. Lastly, the proof is in the actual code. We use a nested for loop that first counts along the row (width) values, and then we perform matrix transpose when doing "rotate[i][j].value = matrix[j][i].value;". The rotated matrix[row][col] will equal the original [col][row].

d.) Help:

I used a multitude of C language documentation sites to look at the proper syntax to perform operations I was familiar with in other languages that I didn't know how to do in C. For example, all of the file read/write and I/O operations were unique to C, which I had to learn both in lecture as well as doing my own research on C documentation sites like: "https://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm".