Jesse Hazard 10/15/2019

For **dimensions**, I originally designed my program to just read the header from the tga file. It worked well, and was extremely efficient. Later testing revealed some strange errors, so I altered the program to instead call the readTGA function and use the data from there. I opted to create an array of integers to represent the optional input arguments. Doing so allowed me to easily verify with quick “if” statements whether the input rules were followed or not. It also allowed me to keep my output string construction to a few brief lines rather than having a large block of code dedicated to various outputs.

In **crop**, I was again searching for the fastest way to run the code. This was when I created my 2d framework and altered both the readTGA and write TGA functions to use it. In doing so, I opted to alter the readTGA function to receive only the file name input and return a reference to a full RasterImage struct. I knew I wanted to combine the file read and raster creation, but originally I wanted to have two functions to create a raster. One that received a file and another that was a blank initializer. However, I realized that the include statements wouldn’t stack properly, so I instead have only the imageIO_TGA.c file create a raster. I took the bits for the rectangle starting at the given x and y coordinates and copied them to the bottom left corner of the original image, then continued that process for every bit inside the dimensions. Once complete, I altered the raster’s column and row numbers to reflect the cropped image. The writeTGA function only accesses the copied bits due to this alteration. Thus no extra memory is allocated.

In **split**, I realized there was a much easier way to accomplish what I needed. I simply altered the writeTGA function. I added a variable called “color” to the input path. The number can range from 0 to 3. I added an “if”statement before the function prints to the file and a blank char variable. If color is 3, the raster prints normally. The other three numbers correspond to color values. If color is 0, or red, then the file prints the blank char at every raster index besides 0.

**Rotate** gave me the most trouble. The input was actually very easy. Once the argument counts were verified, I created a signed int variable. I then ran a loop on the argv[1] chars that incremented if the char was ‘r’ or ‘R’, decremented if the char was ‘l’ or ‘L’, and exitted the program if it was anything else. For the actual program, I originally tried to do the same thing I did in crop. Essentially, I was just trying to swap the values of the bits using a holder variable. However, days of testing resulted in no progress. I finally gave up and decided to allocate memory for a new raster and associated 2d raster. After that, it was a relatively simple process to find the errors. After finding the errors, I never went back to try my original method again, but I think it would use half the memory and finish slightly faster. I still try to keep memory usage low by reusing the original raster variables where possible.

The **script** wasn’t too bad. I finally figured out that if I change the IFS I can make a list from a string containing newline characters, this allowed me to just use the “find” command to search for the image files. Originally I only called “dimensions” once for each file and trimmed the result because I didn’t want to add an extra file read, but I found it was easier to just call it twice.