**Image Manipulation**

Lab5

CPSC 2310

Due: Monday, February 21, 2022

**Program Overview:**

This program will read in a ppm file and perform two transformations on the file. Each transformation will be printed to a new ppm image.

**Learning Objectives:**

This assignment will give you practice in the following concepts:

file I/O,

structs,

pointers,

dynamic allocation of memory, more specifically for a 2D array,

passing arrays and other pointers to functions,

working with multiple files,

fprintf and fscanf functions,

command line arguments,

problem solving,

and more……

**Specifications:**

This assignment will have two “.c”(driver.c and ppm.c) files and one “.h”(ppm.h). The function prototypes for the “.h” file are as follows:

header\_t\* readHeader(FILE\*);

pixel\_t\*\* readPPM(FILE\*,header\_t\*);

void writeP6Image(header\_t\*, pixel\_t\*\*, FILE\*);

void grayScaleImage(header\_t\*, pixel\_t\*\*, FILE\*);

void flipImage(header\_t\*, pixel\_t\*\*, FILE\*);

pixel\_t\*\* allocateMemory(header\_t\*);

void freeMemory(pixel\_t\*\*, header\_t\*);

Provide the implementation for the prototypes in the **functions.c** file.

You will also write the **driver.c** file. The driver should have minimal code. You are only allowed to create variables, and call functions in the driver.

**ppm.h**

The #include’s will go in this file. Then include ppm.h in the remaining “.c” files. Use the preprocessor #ifndef -- #endif to prevent duplicate declaration compile errors. If you are not sure how to do this ask a TA or review the header guard notes. Points will be deducted for not using the preprocessor #ifndef -- #endif.

You **must** create two structs:

1. One for the header information, called **header\_t**. You must use typedef. Header has char magicNum[3], int width, int height, int maxVal.
2. One for the three unsigned char values in each pixel; unsigned char red, unsigned char green, unsigned char blue. Call it **pixel\_t** and use typedef.

**ppm.c**

ppm.c provides the implementation for the functions listed above.

Below is a brief description of each function listed in ppm.h

1. **readHeader** – This function reads the input ppm image header information using fscanf. %s should be use for the magicNum, %d for width, height, and maxVal
2. **readPPM** – This function calls allocateMemory to allocate the memory for a 2D array of pixel\_t type. As you can image, the 2D array will be used to store the red, green, and blue values of the ppm image. You are **REQUIRED** to use a 2D array.
3. **writeP6Image** – This function prints the image. Use fprintf to print the header then loop through each pixel and print the RGB values, use fprintf.
4. **grayScaleImage** – This function prints out a gray scaled image of the original image. In a gray scaled image the magicNum is a **P5** rather than a P6. It also only takes in one value – a combination of the red, green, and blue values of a particular pixel in the image. In other words, for each pixel multiply the red value by .299, the green value by .587, and the blue value by .114. Add the multiplied values together and print that value only. Use %c when printing. Remember no need to print three values only one. Use fprintf to print the image. You should pay close attention to the type of data you are working with. You may need to cast. This function does not change the original image values it only uses them to calculate the value of the grayscaled image. Below is an example output.

 

1. **flipImage** – This function flips the image (top to bottom). The original image will be passed to this function (pixel\_t\*\*). Declare a local variable of type pixel\_t that is a 2D array and dynamically allocate the memory for the output 2D array. Loop through the image passed to the function flipping the pixels top to bottom. **Your program should be able to handle an image that is either square or rectangular. Below is an example of an output.**

 

1. **allocateMemory** – This function will allocate the 2D array and return the address of the allocated memory. You are not allowed to statically allocate the memory you must use malloc or calloc. Pixel\_t pix[height][width] is not allowed. When [] is used to allocate memory this is stored on the stack. You are not allowed to do this.
2. **freeMemory** - This function gives the memory back to the operating system use the ‘C’ function free.

**main.c**

Create three file pointers – one for reading, one to write the flipped image, and one to write the grayscale image. Open all three. The names of the files will be supplied using command line arguments. Using command line arguments allows me to provide various images to test your program. Be sure to check that the user entered the correct number of arguments on the command line. Also check that the files opened successful. If the user did not do either, print to stderr a message then exit the program. You may use assert in main or add a function to ppm.h and ppm.c and call the function in main.

If the files opened successfully then call the function **readHeader** to read the header of the input ppm file. Next call **allocateMemory** to dynamically allocate the memory for the 2D pixel\_t array. Call the function **readImage** which will store the pixels of the input image in the 2D array. Call **grayScaleImage** and **flipImage** passing in the header, the input image you read in and the appropriate file pointer.

**Other Instructions:**

I will provide the Mario ppm file for you to use to test your program.

**Formatting:**

You will need to add a header to each of your files similar to the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Your name

\*CPSC 2311 your lab section

\*Spring 22

\*Your email

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Your program should compile with no warnings and no errors. There will be a deduction up to 50 points if your program does not compile. If your program compiles but has warnings, there will be a deduction up to 20 points.

* Your code should be well documented. (comments)
* There should be no lines of code longer than 80 characters.
* You must use proper and consistent indention.
* Variable names should be meaningful.

There will be a minimal of 5 point deduction per infraction for failing to comply with the above 4 formatting rules.

**Handin:**

Use handin.cs.clemson.edu to submit your files to the Lab5 bucket

Things to do prior to handing in your files:

1. Test your program on **cerf15** on the SoC servers. I will not accept the excuse “It compiled on my computer.” I test programming assignments on cerf15 of the SoC servers.
2. Tar zip your files naming the tarred file Lab4\_<username>.tar.gz Do not place your files in a folder and then tar the folder. This will break the grading script I plan to give the TA’s. When your files are untarred all your files must be visible to the TA. The TA should not see a folder with your files. This will require a step that will not be in the grading script.

**GRADING:**

The grading rubric is attached:

Note to grader: If the files did not compile, look at the code; if a substantial effort was made to complete the assignment give a grade of 30 and move on. If a substantial effort was not made give a 0 and move on.

**Rubric:**

**Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_ Grade:\_\_\_\_\_\_\_**

Did the program compile with no errors? \_\_\_\_\_\_\_ -50 if no

Did the program compile with no warnings? \_\_\_\_\_\_\_ -20 if no

Did the program use file pointer? \_\_\_\_\_\_\_ -5 if no

Did the program close the file pointer? \_\_\_\_\_\_\_ -5 if no

Did the program use command line arguments? \_\_\_\_\_\_\_ -5 if no

Did the program use a 2D array and dynamically allocate the memory?\_\_\_\_\_\_\_ -10 if no

Did the program free dynamically allocated memory? \_\_\_\_\_\_\_ -5 if no

Were the appropriate typedef structs used. (2) \_\_\_\_\_\_\_ -5 for each no

Did the student implement the functions required? \_\_\_\_\_\_\_ -5 if no

Did the following work appropriately.

grayScaleImage \_\_\_\_\_\_\_ -10 if no

flipImage:

Square \_\_\_\_\_\_\_ -5 if no

Rectangle \_\_\_\_\_\_\_ -5 if no

Were header guards used in ppm.h \_\_\_\_\_\_\_ -10 if no

Formatting:

Was code well documented (comments) \_\_\_\_\_\_\_ -5 if no

Were headers in the files? \_\_\_\_\_\_\_ -5 if no

Were there any lines over 80 characters? \_\_\_\_\_\_\_ -5 if yes

Proper and consistent indention \_\_\_\_\_\_\_ -5 if no

Meaning variable names. \_\_\_\_\_\_\_ -5 if no

Notes: