Image Manipulation

Programming Assignment 1
CPSC 1020, Spring 2017
Due: Sunday, February 26, 2017

Program Overview:
This program will read in a ppm file and perform some transformation on the file and print, to a new file, the transformed ppm image.

Learning Objectives:
This assignment is designed to provide practice using several of the "C" concepts we have reviewed over the last couple weeks, as well as, concepts you learned in CPSC 1010 or CPSC 1110. **THIS WILL BE A "C" PROGRAMMING ASSIGNMENT NOT A "C++" ASSIGNMENT.**

Concepts covered in this assignment:
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……

Academic Integrity:
This is an individual assignment. You may not receive help from anyone other than myself or a lab TA. Please review the academic integrity policy provided in the syllabus.

Specifications:
This assignment will have two ".c"(driver.c and functions.c) files and one ".h"(functions.h). The function prototypes for the ".h" file are provided below.

Provide the implementation for the prototypes in the **functions.c** file. **Do not change the signature of any of the functions listed in the functions.h file.** (The signature of a function consist of the return value, the name of the function and the number and type of parameters passed to the function)

You will also write the **driver.c** file.

functions.h
The #include's will go in this file. Then include functions.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 me. Points will be deducted for not using the preprocessor #ifndef -- #endif.

Example:
```
#ifndef FILE_NAME_H
#define FILE_NAME_H

/* code */

#endif // #ifndef FILE_NAME_H
```

The functions.h file will include the declaration of the two structs and 10 function prototypes.

Following are the prototypes for the 10 functions needed for this program:

```
void readHeader(header_t*, FILE*);
void readImage(header_t*, pixel_t **, FILE*);
void chooseTransform(header_t *, pixel_t **, FILE*);
void printP6Image(header_t*, pixel_t **, FILE*);
void grayScaleImage(header_t *, pixel_t **, FILE*);
void flipImage(header_t *, pixel_t **, FILE*);
void rotateLeft(header_t * , pixel_t **, FILE*);
void rotateRight(header_t *, pixel_t **, FILE*);
void color2Negative(header_t*, pixel_t **, FILE*);
void reprint(header_t*, pixel_t**, FILE*);
```

You **must** create two structs:

1. One for the header information, called **header_t**. You must use typedef.
2. One for the three unsigned char values in each pixel. Call it **pixel_t** and use typedef.

functions.c

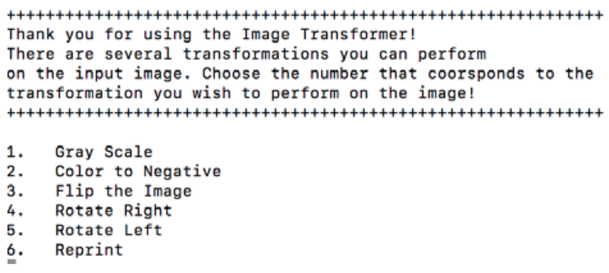
Functions.c provides the implementation for the 10 functions listed in functions.h.

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

1. **readHeader** – This function reads the header information using fscanf.

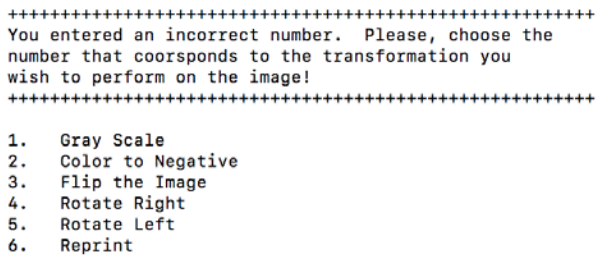
2. **readImage** – This function reads the pixels from the image. This function should be called in the driver. Notice Pixel_t is a double pointer. You are **REQUIRED** to use a 2D array.

3. **chooseTransform** – This function should present a menu of transformations that the user can choose from. Below is an example of what prints when the program is executed.



```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Thank you for using the Image Transformer!
There are several transformations you can perform
on the input image. Choose the number that coorsponds to the
transformation you wish to perform on the image!
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1.   Gray Scale
2.   Color to Negative
3.   Flip the Image
4.   Rotate Right
5.   Rotate Left
6.   Reprint
```

Ensure the user enters a correct number. If an incorrect number is entered print something similar to the following.



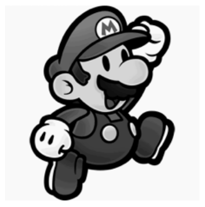
```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
You entered an incorrect number.  Please, choose the
number that coorsponds to the transformation you
wish to perform on the image!
++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1.   Gray Scale
2.   Color to Negative
3.   Flip the Image
4.   Rotate Right
5.   Rotate Left
6.   Reprint
```

The program should continue to give this message until the user enters a number between 1 and 6.

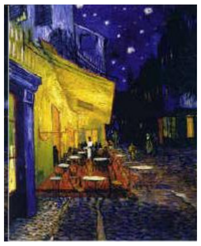
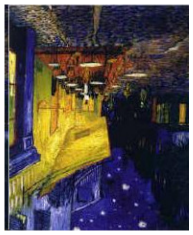
Call the function that corresponds to the transformation chosen by the user.

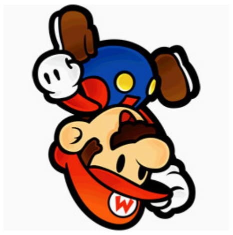
4. **printP6Image** – This function prints the image after the transformation has been completed. Use fprintf to print the header then loop through each pixel and print the RGB values, use fprintf.

5. **grayScaleImage** – This function prints out a gray scaled image of the original image. A gray scaled image is a **P5** image 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.

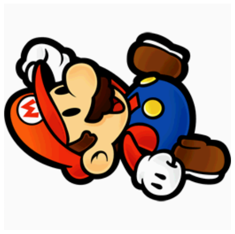
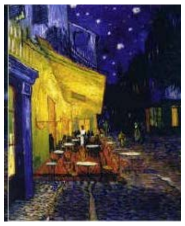
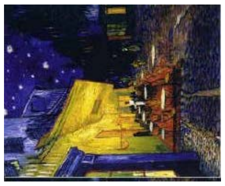


6. **flipImage** – This function flips the image (top to bottom). The original image will be passed to this function (pixel_t**). Declare a local variable of type pixel_t that is a 2D array and dynamically allocate the memory for the output 2D array. Loop through the image passed to the function flipping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

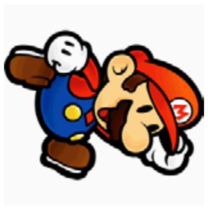
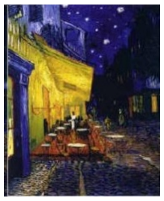
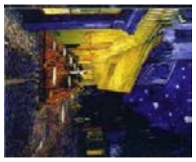





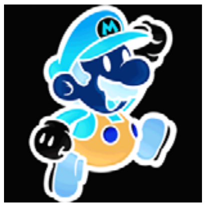
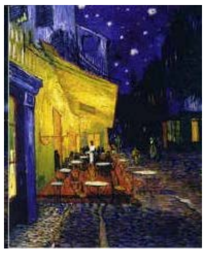
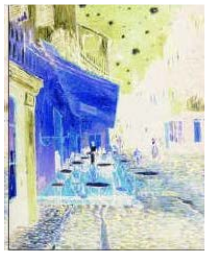
7. **rotateLeft** – This function rotates the image 90 degrees to the left. The original image will be passed to the function. Create a 2D array, dynamically allocating the memory for the rotated image. Once the memory is allocated, loop through the rows and columns manipulating each pixel, flipping the image to the left. **Remember, your program should be able to handle an image that is either square or rectangular.**

8. **rotateRight** – This is the same as rotate left except rotating the image to the right.

9. **color2Negative** – This function creates a negative of an image. Let's think about what that means. We are working with RGB and each of these color channels are represented by a number between 0 – 255. If red is 0 we want it to be the opposite which is 255, or if red is 255 then we want it to be 0. But what about the numbers in between. Suppose red is a little red, 25, then the opposite is almost completely red 255 – 25 = 230. The same goes for green and blue. So for each pixel change each of the RGB channels to equal 255 – the color value.

10. **reprint** – This function simply prints the original image. Hint: This function can be implemented using one line of code.

driver.c

1. Create two file pointers – one for reading and one for writing – and open the two files. 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.

   After opening the files check the following:
   1. Did the input file open successfully?
   2. Did the output file open successfully?

   If the files opened successfully then call the function **readHeader** to read the header of the input ppm file. Use pixel_t to create a 2D array to hold the pixel values for the input image—**using a 2D array is required not an option**. Dynamically allocate the memory for the 2D pixel_t array then call the function **readImage** which will store the pixels of the input image in the 2D array. Next, determine what type of manipulation the user wishes to perform on the input image by calling the function **chooseTransform**. After **chooseTransform** has been called, free the dynamically allocated memory and close the files opened.

Other Instructions:

I will provide a makefile. Type **make** and your program should compile.
In addition to driver.c, functions.c, and functions.h you should submit a README file. This file name should be in all caps. Your README file should provide the following:

1. A short description of any problems you encountered when writing this program.
2. How you solved the the problems you encountered.
3. Your thoughts on the assignment. This is your opportunity to tell me if you like the assignment or not. What you did or did not like about the assignment. Anything you want to tell me.

I will also provide two ppm files 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 1020 your section, Sp17
 *Your user name
 *************************/
```

Your program should compile with no warnings and no errors. There will be a deduction up to 30 points if your program does not compile. If your program compiles but has warnings, there will be a deduction up to 10 points.

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

There will be a 5 point deduction for failing to comply with the above formatting rules.

Handin:
Use handin.cs.clemson.edu to submit your files. I have created a bucket named PA1.
Things to do prior to handing in your files:

1. Test your program on the SoC servers. I will not accept the excuse "It compiled on my computer." I test programming assignments on the SoC servers.
2. Run "make clean" to remove the executable and all .o files
3. Tar zip your files naming the tarred file PA1.tar.gz if you are doing the Extra Credit name your file PA1EC.tar.gz

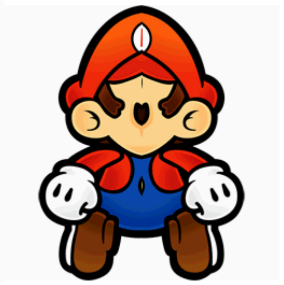
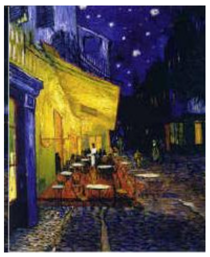
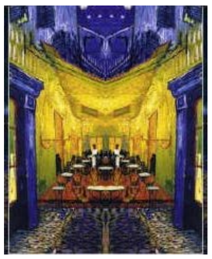
GRADING:
The grading rubric is attached:

EXTRA CREDIT:

5 POINTS EACH

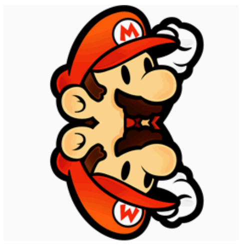
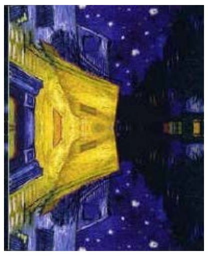
There are two opportunities for extra credit:

1. Create a transformation that will mirror an image vertically.

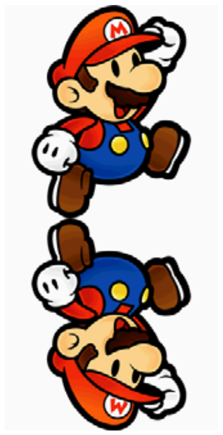
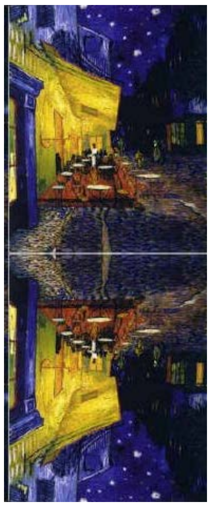




The above is what I call a half mirror. The image is the same size you mirror only half of the image. You can write the full or half mirrored version.

2. Create a transformation that will mirror an image horizontally.

The above is what I consider a half mirror. Basically it only mirrors half of the image. Below is a full horizontal mirror. You can write either the full or half version.

If you choose to do the extra credit you may do either the vertical or horizontal option for up to 5 points. If you choose to do both vertical and horizontal you could receive up to 10 extra credit points.

If you do one or both of the extra credit options add the functions to the regular assignment and only submit your files to the PA1EC folder in handin. **DO NOT SUBMIT TO BOTH HANDIN FOLDERS.**

You must follow all the above formatting specifications for the EC.

Rubric:

Did the program compile with no errors?	________	-30 if no
Did the program compile with no warnings?	________	-10 if no
Did the program use file pointer?	________	-2 if no
Did the program close the file pointer?	________	-2 if no
Did the program use command line arguments?	________	-2 if no
Did the program use a 2D array and dynamically allocate the memory?	________	-2 if no
Did the program free dynamically allocated memory?	________	-2 if no
Did the student provide a README file?	________	-5 if no
Was the file named correctly?	________	-2 if no
Did readHeader work correctly?	________	-2 if no
Did readImage work correctly?	________	-2 if no
chooseTransform:		
Provide a menu?	________	-2 if no
Provide error checking on menu option?	________	-2 if no
Did printP6Image work correctly?	________	-2 if no
grayScaleImage	________	-2 if no
flipImage:	________	-2 if no
Square	________	-2 if no
Rectangle	________	-2 if no
rotateLeft:		
Square	________	-2 if no
Rectangle	________	-2 if no
rotateRight:		
Square	________	-2 if no
Rectangle	________	-2 if no
color2Negative	________	-2 if no
reprint	________	-2 if no
Was ifndef in functions.h	________	-2 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: