CPSC 1110 Fall 2020, Programming Assignment 3: Image File Transformation

Due 11:59:59 pm on due date

1. Overview

The purpose of this assignment is to give you some experience using command-line arguments and file I/O as well as dynamically allocated memory. Your program will read in a .ppm image file and apply a transformation to the image, saving the result as a separate .ppm image. A .ppm image file will be provided on the Assignment 3 page on canvas.

2. More Assignment Specifics

This program will read in a .ppm file specified by the command-line argument following the name of the executable/binary file. Following the command-line argument for the file name will be one of the following: ‘g’ / ‘G’ to convert to grayscale, ‘n’ / ‘N’ to invert the colors (negative), ‘r’ / ‘R’ to rotate the image 90 degrees clockwise (to the right) or ‘s’ / ‘S’ to shrink the image to half the width and height. With the ‘g’ and ‘n’ options, the program will perform a simple transformation where the color of each pixel will be converted to grayscale (‘g’) or inverted to negative (‘n’) and write the resulting image to a .ppm file with “_grayscale” or “_negative” concatenated to the base filename. With the ‘r’ option, the program will perform a simple transformation where the image is rotated 90 degrees clockwise and write the resulting image to a .ppm file with “_rotate” concatenated to the base filename. If the ‘s’ options is used, the program will average each 2x2 block of pixels to a single pixel and write a .ppm file of half the width and half the height with “_halfsize” concatenated to the base filename.

For example the following command:
`./main oneTiger.ppm g` (or `./main oneTiger.ppm G`)

will read in from a file named oneTiger.ppm located in the same directory as the program (main) and write the grayscale image to a file named oneTiger_grayscale.ppm in the same directory.

The following command:
`./main oneTiger.ppm n` (or `./main oneTiger.ppm N`)

will read in from a file named oneTiger.ppm located in the same directory as the program (main) and write the inverted image to a file named oneTiger_negative.ppm in the same directory.

The following command:
`./main oneTiger.ppm r` (or `./main oneTiger.ppm R`)

will read in from a file named oneTiger.ppm located in the same directory as the program (main) and write the image rotated 90 degrees clockwise to a file named oneTiger_rotate.ppm in the same directory.

The following command:
`./main oneTiger.ppm s` (or `./main oneTiger.ppm S`)

will read in from a file named oneTiger.ppm located in the same directory as the program (main) and write the half-size image to a file named oneTiger_halfsize.ppm in the same directory. File pointers must be used, **stdin / stdout redirection may not be used**.

It may help to divide the assignment into several parts. First you will need to use the command-line argument specifying the name of a file. Use this file name to open an input file. For your output file, add "_grayscale" / "_negative" / "_rotate" / "_halfsize" to the base file name (before the .ppm extension) based on the option (third command-line argument); you may assume a .ppm extension for the input file and the output file must have a .ppm extension.

The next step is parsing the header information in the input file (which will be also written to the output file). The beginning of the file is where the header information is stored. The header begins with a "magic number" which is the character 'P' followed by the character '6' followed by a newline character ('\n'). After this is a number (as a decimal ASCII numeric string) representing the width of the image in pixels, followed by a space (' '), followed by another number representing the height of the image in pixels, followed by a newline character ('\n'). After this is another numeric string (most often "255") representing the maximum color value. After this, the header ends with a single whitespace character, typically '\n'. Your program can assume a maximum color value of 255, but "255" followed by '\n' will need to be at the end of the header of the output file as well. Additionally, on any line before the line with the maximum color value in the header of the input file, comments may appear: comments begin on a new line with the '#' character and end with the newline character ('\n'). Comments should be ignored and do not need to be written to the output file's header. For more information about the ppm image format you can refer to: http://netpbm.sourceforge.net/doc/ppm.html

Example .ppm header information:

`P6`	"magic number": 'P', '6', followed by '\n'
`# CREATOR: GIMP PNM Filter Version 1.1`	comment
`561 375`	<width> ' ' <height> followed by '\n'
`255`	<maxval> followed by '\n' followed by image data
????????????????????????????????	image data (byte values ranging between 0 and 255)

After the header information has been parsed (and written to the output file), the next step is to read in the image data. The image data is a sequence of bytes (which can have any value – binary data as opposed to ASCII data) containing the data for each pixel in the image. Each pixel consists of a set of 3 bytes, the first being the red value for a pixel, the second being the green value for the same pixel and the third being the blue value for the same pixel. It is important to allocate enough memory using calloc() / malloc() to store a reasonable portion of the image to work on (the easiest will be to store the entire image as a 1-dimensional array of pixels, refer to **3. More Details** for implementing a `struct` to store image data). Additionally your program should use fread() and fwrite() to read in pixels byte-by-byte, i.e., one at a time. You may want to verify that the file I/O part of the program is working correctly by copying the untransformed image data to the output file before proceeding with the implementation of the 4 transformations. Be sure to close the input and output files when finished with using them in your program as well as using free() to return allocated memory when finished using a dynamically allocated block of memory.

Finally implement the 4 image transformations:
Grayscale ('g' or 'G' as third command-line argument):
Should average the 3 values in each pixel in the image. For example a normalized red color pixel (1, 0, 0) will become a gray pixel (.33, .33, .33) and gray pixels (R=G=B) remain unchanged. This should work with any pixel value: Given the following pixel: R: .39 G: .78 B: 1, the grayscale pixel will be R: .72 G: .72 B: .72

Negative ('n' or 'N' as third command-line argument):
Should invert the values in each pixel in the image. For example a white pixel (1, 1, 1) will become a black pixel (0, 0, 0) and vice-versa. This should work with any pixel value:
Given the following pixel: R: .39 G: .78 B: 1, the negative pixel will be R: .6 G: .21 B: 0

Rotate ('r' or 'R' as third command-line argument):
Should rotate the image 90 degrees clockwise (to the right). For example the leftmost column reading upwards in the original image will become the top row reading rightwards in the transformed image. After processing through

the entire image the rightmost column reading upwards in the original image becomes the bottom row reading rightwards in the transformed image.

Half-size (‘s’ or ‘S’ as third command-line argument):
Should shrink the image to half-size (both width and height). This should be done by taking the average R/G/B value of each 2x2 group of pixels in the original image to form a single pixel in the transformed image. For an original image of odd width / height, the leftmost column (with odd width) and the bottom row (with odd height) will be ignored. For example, if the original image is 401 x 251 the transformed image will be 200 x 125.

3. More Details

Each pixel from the image data of the .ppm file is a sequence of 3 bytes in the order of the red value, followed by the green value then the blue value. Each of these values can have a minimum of 0 and a maximum of 255. This max color value is stored in the image header. As a result the correct data type to use to read in, store and write out these values would be `unsigned char`. However, for image processing, pixels should be represented as *normalized* 1D `float` arrays meaning that each [0:255] pixel value is divided by 255.0 to put it in the range [0:1]. For example:

```
typedef struct {
      int cols;          // width
      int rows;          // height
      int maxc;          // max color value (usually 255)
      char *magic;       // magic number (usually ‘P6’)
      float *rpix;       // red pixels, stored as 1D float array
      float *gpix;       // green pixels, stored as 1D float array
      float *bpix;       // blue pixels, stored as 1D float array
} PPM;
```

The program should work with any size image by reading the width and height information from the header of the .ppm file. In addition to correct functionality, the code in your program will be evaluated on the formatting and programming style. Make sure your program has consistent formatting and that you use meaningful variable names. The program should be divided into functions with at most 30 – 50 statements each and any section of code that will run frequently should be made a function. Make use of function prototypes and have the main function be the first function definition with the implementation of other functions appearing below main(). **Use of global variables is not allowed**, instances of global variables within the program will result in a reduction of points.

It is also a good idea to write and test the program in various stages, making sure that everything works correctly in the current stage before moving on. This makes writing and testing the program much more manageable than trying to tackle the entire program in one step. You can follow the general sequence described in the section above (more assignment specifics) or divide up the steps in any other logical way. The sub-problems will include using command-line parameters, performing file input, performing file output, processing the header of the .ppm file, allocation of memory and the 4 image transformations.

4. Hand-in Instructions

Your file should be named asg03.tar.gz and include files Makefile, main.c and any other required .c files to be submitted via handin.cs.clemson.edu by 11:59:59 PM of the due date.

In the header comment of your main.c file, you need to also include the following things:

1. a brief description of the program
2. a guess as to how long you thought it would take to code this assignment before you got started
3. the actual time it took to code this assignment

The source code will be evaluated not only on its correctness, but also on its adherence to the coding style standards “Code Formatting Requirements” provided on Canvas. Don’t forget to use meaningful variable names, and don’t forget about indentation, comments, and lines not exceeding 80 characters (including spaces).

5. Notes on Collaboration

You are required to work individually on this assignment. **Please do not consult *anyone* other than your instructors or the lab assistants on *any* aspect of this assignment**. Other tutors that are available may be able to help with more general C concepts that you may be struggling with, but not with questions specific to the assignment.

6. Grading Rubric

80%	functionality (program works correctly and generates the correct transformed image saved as a properly formatted .ppm image given an input image of any size, uses dynamic memory allocation)
10%	formatting of code (header comment + other comments, indentation, meaningful variable names, no use of global variables, lines not longer than 80 characters)
5%	handles input correctly, should read the header information (length+width), ignore comments in .ppm file header and correctly read in the image efficiently (do not use fgetc() / fputc() for image data)
2.5%	contents of header comment (name; date; description; estimation of time to code; actual time it took to code) & file named correctly
2.5%	no warnings when compiled

The highest possible score for a program that does not compile is 20%

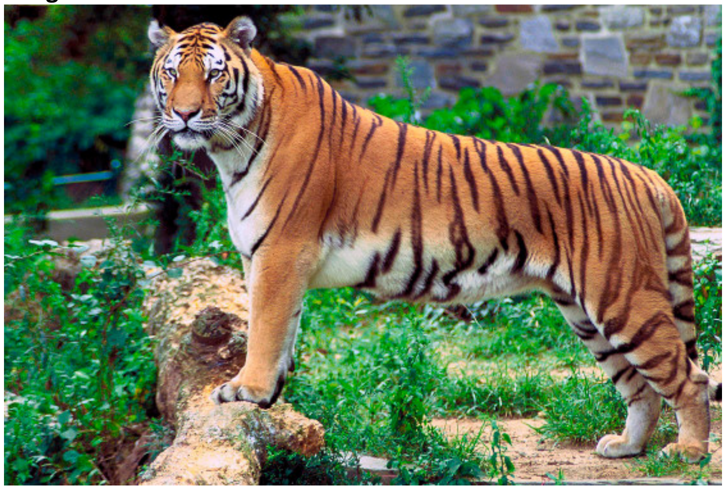
7. Sample Output

Given the input file "oneTiger.ppm":

Header:

```
P6
# CREATOR: GIMP PNM Filter Version 1.1
561 375
255
```

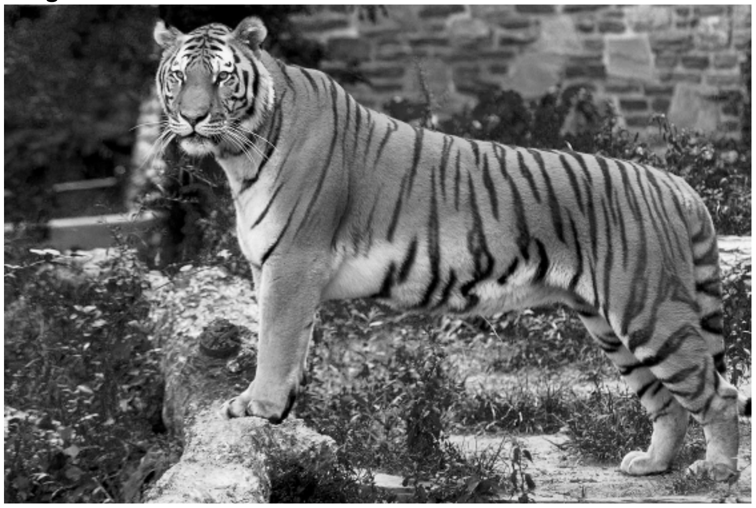
Image:



The correct output image for `./main oneTiger.ppm g` (or `./main oneTiger.ppm G`):
"oneTiger_grayscale.ppm" should appear as the following:
Header: (does not need to include comments from the header of the input file)

```
P6
561 375
255
```

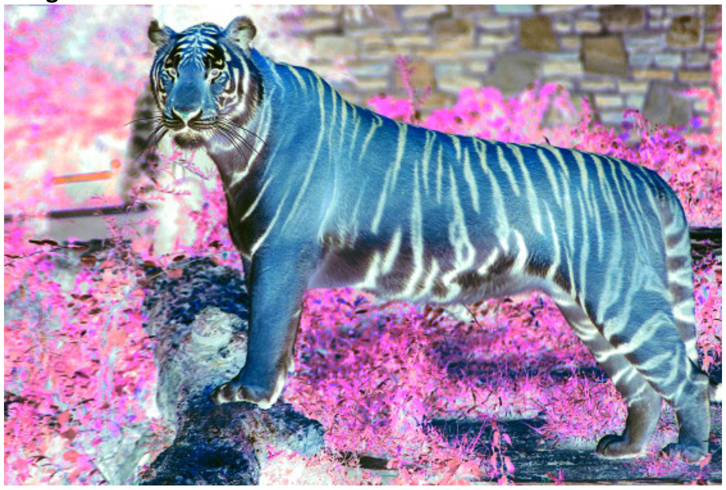
Image:



The correct output image for `./main oneTiger.ppm n` (or `./main oneTiger.ppm N`):
"oneTiger_negative.ppm" should appear as the following:
Header: (does not need to include comments from the header of the input file)

```
P6
561 375
255
```

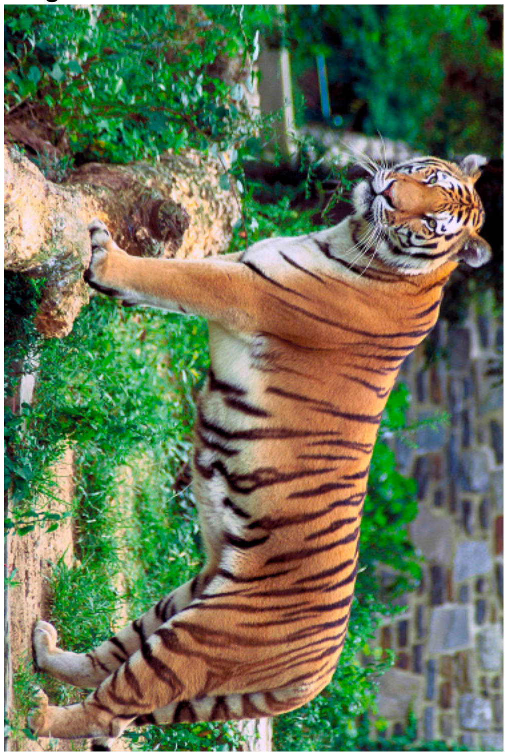
Image:



The correct output image for `./main oneTiger.ppm r` (or `./main oneTiger.ppm R`):
"oneTiger_rotate.ppm" should appear as the following:
Header: (does not need to include comments from the header of the input file)

```
P6
375 561
255
```

Image:



The correct output image for `./main oneTiger.ppm s` (or `./main oneTiger.ppm S`):
"oneTiger_halfsize.ppm" should appear as the following:
Header: (does not need to include comments from the header of the input file)

```
P6
280 187
255
```

Image: