Programming Assignment 2, TCSS 333A, Winter 2021

OBJECTIVE

The objective of this assignment is to give you more practice with 2D arrays, representation of data, and basic strings.

ASSIGNMENT SUBMISSION

To get credit for this assignment, you must

- ✓ write a program in C that you wrote on your own (no copying or help outside of CSS mentors or TLC or instructor allowed)
- ✓ submit your files through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment by the due date

PROBLEM STATEMENT

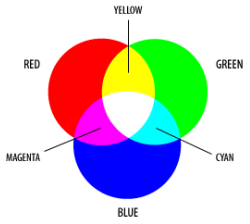
Create a program that reads 24-bit bmp files and generates new versions of the original image. The new versions to be supported are:

- original converted to sepia tones
- original converted to color pop
- original mirrored along horizontal axis
- original flipped along vertical axis
- your choice of interesting and complex image manipulation

You need to follow the algorithms explained below while processing the images.

BMP files

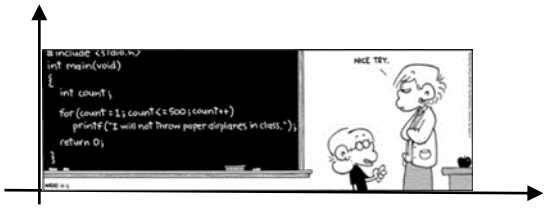
A 24-bit bmp file is capable of storing 2d images. A file consists of a 54 byte header and the rest of the file is pixel information. Each pixel is represented by 3 bytes, where each byte contains the information for blue, green, and red hues – in this order. Each hue can be of value 0..255, where 0 signifies the complete lack of that color, while 255 signifies its full saturation. New colors are generated by mixing different intensities of this basic color palette. All together 256^3 = 16,777,216 colors can be represented using this scheme.



Note that the image height and width in pixels correspond to how we store information in a 2D array. In the bmp format the pixel (0,0) is located at the bottom left (origin of the plane) at locations (0,0), (0,1), (0,2).

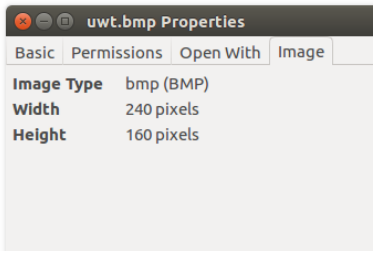
Width (x 3)

height		0	1	2	3	4	5	.....				N
	height-1	Pixel 1 B	Pixel 1 G	Pixel 1 R	Pixel 2 B	Pixel 2 G	Pixel 2 R	.....		Pixel N B	Pixel N G	Pixel N R
	...											
	2											
	1											
	0											



A sample file `bmpchange.c` is provided to show how to read and write such files, and how blue and green components of each pixel could be switched. You are to use this file as the basis of your program (for reading and writing files, just use the provided code).
A few small bmp test files are provided with this homework as well. Since your program will need to ask for the height and width of the picture in pixels, you may find this information in Ubuntu by right-clicking on the bmp image file, selecting Properties and then Image tab.


uwt.bmp



Pixel Manipulation Algorithms

Sepia Algorithm

For each pixel, recalculate its BGR values as:

newBlue = (R * 0.272 + G * 0.534 + B * 0.131)

newGreen = (R * 0.349 + G * 0.686 + B * 0.168)

newRed = (R * 0.393 + G * 0.769 + B * 0.189)

If the new value > 255, make it 255

Color Pop Algorithm

Color pop refers to keeping some pixel colors and making all other grayscale. You will perform color pop based on the color of the middle pixel of each image (at location width/2 by height/2 – remember that width-wise it really means three array cells). All pixels whose color are within a certain "distance" to the middle pixel color will remain the same as in the original image. All pixels whose color is greater than or equal to the "distance" are changed to grayscale pixels. To calculate the distance between two pixels, use the formula:

$$\sqrt{(\text{red\_value\_1} - \text{red\_value\_2})^2 + (\text{green\_value\_1} - \text{green\_value\_2})^2 + (\text{blue\_value\_1} - \text{blue\_value\_2})^2}$$

To calculate the grayscale of a pixel, use the formula

(blue_value + green_value + red_value) / 3 = average

Replace original BGR values with average, average, average

In our implementation, let's use the distance of 55 as our threshold for changing between grayscale and color

Vertical Axis Flip

For each row in old image, copy the row in reverse:

Before

a	b	c
d	e	f
g	h	i
j	k	l

After

c	b	a
f	e	d
i	h	g
l	k	j

Horizontal Axis Mirror

For each column in old image:

- Copy the first half as-is
- Copy the second half reversed

Before

a	b	c
d	e	f
g	h	i
j	k	l

After

a	b	c
d	e	f
d	e	f
a	b	c

Before

a	b	c
d	e	f
g	h	i
j	k	l
m	n	o

After

a	b	c
d	e	f
g	h	i
d	e	f
a	b	c

Your choice of interesting and complex image manipulation

If you would rather replace one of the options above with some other manipulation, feel free to do so. However, note that to receive credit, the manipulation must include the manipulation of positions and colors. For example, one interesting idea is to figure out how to "censor" a picture by pixelating it.

PROGRAM SPECIFICATIONS

Your program is to prompt for the name of the input file and its pixel dimensions. Your program has to run exactly as shown below and follow the same input format. Note that although a file will have a bmp extension, a user will only enter the part of the name that precedes it, so the program will need to append *.bmp*; then the user is to be prompted for height and width (in this order) – assume valid input.

This is a sample run of the program (bold and italics indicate user entries).:

```
---
Enter the filename: test1
Enter height and width (in pixels): 160 240
Done. Check the generated images.
---
```

Note that the generated files should be named:

```
sepia.bmp
pop.bmp
mirror.bmp
flip.bmp
surprise.bmp
```

Sample input files for processing and sample processed versions of one of them are uploaded into Canvas.

- Your program is to be contained in a single c file named ***pr2.c***
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), meaningful variable names, etc.
- Overall, your program is to use at most two 2D variable length arrays: one that contains the original image and the other that contains the copy of the image
- Open and close files per-need basis
- If you want to copy an entire matrix at once, use *memcpy* function to do so instead of copying cell by cell, as in *memcpy(matrix2, matrix1, sizeof(matrix1);* // this copies matrix 1 into matrix 2
- **Your program must compile in gcc gnu 90 – programs that do not compile will receive a grade of 0**
- If you use a math library, you will need to compile with -lm flag (note, the first letter is lowercase L as in lion or letter) at the end, gcc -std=gnu90 pr1.c -lm
- Your program should include the following comments:
    - Your name at the top
    - Whether you tested your code on the cssgate server or Ubuntu 16.04 LTS Desktop 32-bit
    - Comments explaining your logic
    - If you choose your own image manipulation, explain it

GRADING

Remember, the programming assignments are graded as pass / no pass only and 85 points (out of 100 are needed to pass). If you do not pass this assignment, you can turn it in again with assignment 3, 4, or 5.

Each image manipulation implemented according to instructions and running perfectly constitutes 15 points but the maximum earned here is 60 points (i.e. 4 out of 5 choices)

Proper coding style, meaningful identifiers, etc. is worth 15 points

Program that runs according to the specs that conserves resources is worth 25 points. Conserving resources in this instance means things such as:

- variables are never declared inside loops
- the program reads once into a 2d array and uses the in-memory image version for processing
- at most two 2d arrays are used
- input and output streams are only open when needed

PROGRAM SUBMISSION

On or before the due date for assignment 2, use the link posted in *Canvas* next to *Programming Assignment 2* to submit your C code. Make sure you know how to do that before the due date since late assignments will not be graded until the next grading period. Valid program format: a single file named *pr2.c*