Image Processing

Summary: In this homework, you will be implementing a program that loads an image file, applies filtering to it, and saves it back to disk. You will also learn how to read and write real-world binary file formats, specifically BMP.

1 Background

In this assignment, you will write a program that applies three different filters to an image. The image will be loaded from a BMP file, specified by the user, and then be transformed using: 1) a grayscale conversion, 2) a color shift also specified by the user, or 3) a scale operation. The resulting file will be saved back to a BMP file that can be viewed on the system.

This document is separated into four sections: Background, Requirements, Loading Binary Files, Include Files, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Loading Binary Files, we will discuss the BMP file format and how to interact with binary files. Lastly, Submission discusses how your source code should be submitted on Canvas.

2 Requirements [36 points]

Your program needs to implement loading a binary BMP file, applying one or more filters, and saving the modified image. The width and height may be of any size, but you can assume that you will be capable of storing the image data in memory. Assume that all BMP files are 24-bit uncompressed files[1] (i.e., compression method is BI_RGB), which have a 14-bit bmp header, a 40-bit dib header, and a variable-length pixel array. See Section 3 for further discussion of the BMP format. This is shown in Figure 1.

As a base requirement, your program must compile and run under Xubuntu (or another variant of Ubuntu) 22.04. Several sample outputs are shown in subsection 2.1. **Do not modify the header files given except to fill in any TODOs.**

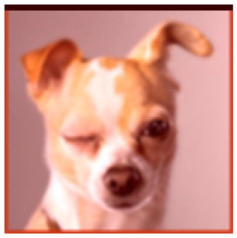
- **Specific Requirements:**
  - BMP Headers IO: Create structures for the headers of a BMP file (BMP header struct and DIB header struct) and functions that read and write them. [4 Points]
    - See struct BMP_Header, struct DIB_Header, readBMPHeader, writeBMPHeader, readDIBHeader, writeDIBHeader, makeBMPHeader, and makeDIBHeader. These functions should be implemented in a file called BMPHandler.c.
  - Pixel IO: Create a structure that represents a single pixel (24-bit pixel struct) and the functions for reading and writing all pixels in BMP files. [4 Point]
    - See struct Pixel, readPixelsBMP, and writePixelsBMP. These functions should be implemented in a file called BMPHandler.c.
  - Input and output file names: Read the input file name and output file name from the command line arguments. The user should be required to enter the input file name and this should be the first argument. The output file name is an "option," it is not required and it can be in any place in the options list. The output file option is specified by "-o" followed by the output file name. Validate that the input file exists. [4 Points]

[1] In case you think this is making the assignment unrealistic, think again: this is the default Windows 10 BMP file format.

    * The input file is the file to read and copy.
    * The output file name is the file to write to and copy to (the new file created).
  - Filter command line parsing: Accept an option for grayscale (black and white). This is specified by "-w". Accept options for red, green, or blue color shift. These are specified by“-r” “-g” or “-b” followed by an integer. Accept an option for scale. This is specified by "-s" followed by a float. Like the “-o” option described above, these can come in any order in the options list and are optional for the user to enter. [4 Points]
  - Copy images: Have the ability to copy an image from a BMP file to a new BMP file. [4 Points]
  - **Grayscale filter** (image_apply_bw): convert each RGB pixel to its grayscale equivalent. [4 Points]
    * In a grayscale image, each RGB component has the same value. Use the following formula to compute it: grayscale = 0.299R + 0.587G + 0.114B [2]
    * This filter MUST happen before the color shift filter is applied.
  - **Color Shift filter** (image_apply_colorshift): Shift the color of the new image before saving it, according to the options the user entered. [4 Points]
    * Color shift refers to increasing or decreasing the color in a pixel by the specified amount. So, if the user entered “-b -98” all of the blue values in a pixel would be decreased by 98.
    * If no color shift option was entered for a color, do not shift it.
    * After the color shift, color should be clamped to 0 ~ 255. For example, color R = 100, shift = 200. The color R after shift should be 255 (300 clamped to 255).
  - **Scaling filter** (image_apply_resize): perform nearest neighbor resize on the image. [4 Points]
    * Create a new pixel array of the appropriate size (e.g., old_width * scaling_factor).
    * For each pixel in the new pixel array, set its value to (roughly) the nearest neighbor in the original. For example, if you have an image that is 100x100 and being resized to 50x50, then the pixel at 25x25 in the smaller resized image would be the same as the one at 50x50 in the original. (Informally, the pixelhalf way through a smaller image would be the same as the image halfway through the larger image.
  - OOP: Follow the provided header file for Image to structure it as an object. The Image functions (five below and three filters above) should be implemented in a file called Image.c. [4 Points]
    * See image_create, image_destroy, image_get_pixels, image_get_width, image_get_height.

2.1 Sample Outputs

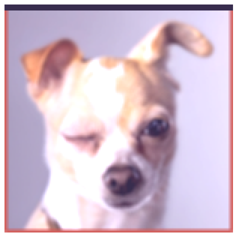
- ImageProcessor ttt.bmp -r 56

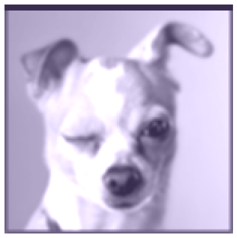
  Output file name was ttt_copy.bmp.

- ImageProcessor ttt.bmp -r 56 -b 78 -g 45 -o filtered1.bmp

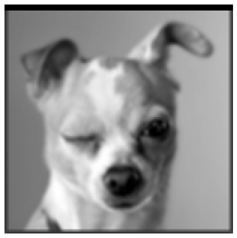
[2]https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-space-conversion/



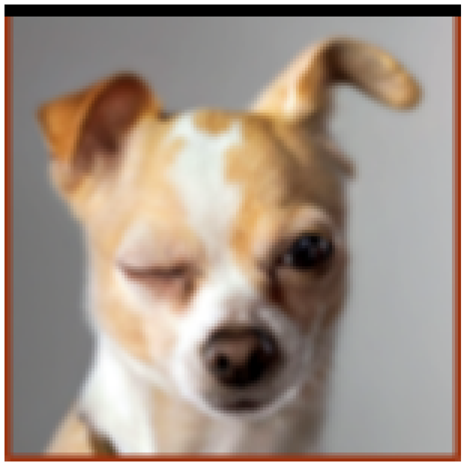
- ImageProcessor ttt.bmp -w -r 56 -b 78 -g 45 -o filtered2.bmp



- ImageProcessor ttt.bmp -w -o filtered3.bmp



- ImageProcessor ttt.bmp -s 2.0 -o filtered4.bmp



- ImageProcessor ttt.bmp -s 2.0 -w -o filtered5.bmp

- ImageProcessor ttt.bmp -w -s .5 -r 56 -o filtered6.bmp

3 Loading Binary Files

3.1 The BMP File Format

For reference, use the BMP specification on Wikipedia: https://en.wikipedia.org/wiki/BMP_file_format. In Figure 1, a graphical overview of a BMP file's layout is shown. The layout is literally the meaning of the bits/bytes in the file starting at 0 and going to the end of the file. The first (green) region shows the BMP header information in 14 bytes: 2 for the signature, 4 for the file size, 2 for reserved1, 2 for reserved2, and 4 for file offset. *Details on each of these (such as their data format, and contents) can be found on the Wikipedia page.* For example, the area labeled "Signature" should contain the characters BM to confirm that the file is in BMP format. The second (blue region) shows the DIP header information in 40 bytes: 4 for header size, 4 for width, 4 for height, and so on. The last region (yellow) forms a 2D array of pixels. It stores columns from left to right, but rows are inverted to be bottom to top. Note that each row of pixels in this section is padded to be a multiple of four bytes. For example, if a line contains two pixels (24-bits or 3-bytes each), then an additional 2-bytes of blank data will be appended.

Review the following struct called BMP_Header which holds the bmp header information. (You should also consider creating structs to hold the dib header and pixel data.) Notice that the entries in BMP_Header correspond to the pieces of data listed in the file format. In general, a chunk of 8-bits should be represented as a char, a chunk of 16-bits as a short, and 32-bits as an int. (Optionally: consider using unsigned types.)

```
struct BMP_Header {
        char signature[2];          //ID field
        int size;                   //Size of the BMP file
        short reserved1;            //Application specific
        short reserved2;            //Application specific
        int offset_pixel_array;     //Offset where the pixel array can be found
};
```

Plan to review "Example 1" on the Wikipedia page to get a feel for the contents of each of these regions. A recreated version of that image is provided as the attached "test2.bmp" file. A good exercise would be to view the file in a hex editor like Bless.

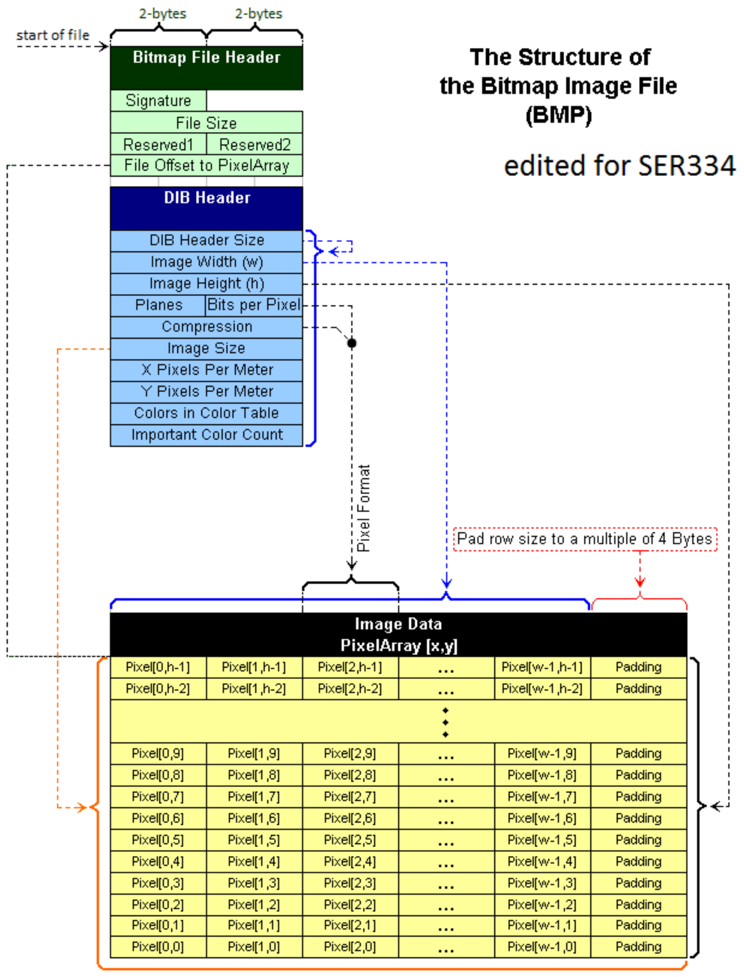
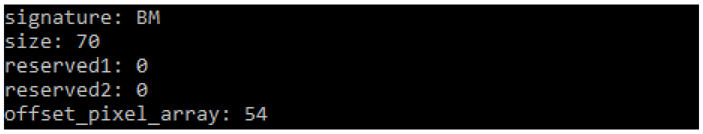

2-bytes
2-bytes
start of file
Bitmap File Header
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
The Structure of the Bitmap Image File (BMP)
edited for SER334
Pixel Format
Pad row size to a multiple of 4 Bytes
Image Data PixelArray [x,y]
Pixel[0,h-1] Pixel[1,h-1] Pixel[2,h-1] ... Pixel[w-1,h-1] Padding
Pixel[0,h-2] Pixel[1,h-2] Pixel[2,h-2] ... Pixel[w-1,h-2] Padding
Pixel[0,9] Pixel[1,9] Pixel[2,9] ... Pixel[w-1,9] Padding
Pixel[0,8] Pixel[1,8] Pixel[2,8] ... Pixel[w-1,8] Padding
Pixel[0,7] Pixel[1,7] Pixel[2,7] ... Pixel[w-1,7] Padding
Pixel[0,6] Pixel[1,6] Pixel[2,6] ... Pixel[w-1,6] Padding
Pixel[0,5] Pixel[1,5] Pixel[2,5] ... Pixel[w-1,5] Padding
Pixel[0,4] Pixel[1,4] Pixel[2,4] ... Pixel[w-1,4] Padding
Pixel[0,3] Pixel[1,3] Pixel[2,3] ... Pixel[w-1,3] Padding
Pixel[0,2] Pixel[1,2] Pixel[2,2] ... Pixel[w-1,2] Padding
Pixel[0,1] Pixel[1,1] Pixel[2,1] ... Pixel[w-1,1] Padding
Pixel[0,0] Pixel[1,0] Pixel[2,0] ... Pixel[w-1,0] Padding

Figure 1: BMP file format structure for 24-bit files without compression. Image modified from https://en.wikipedia.org/wiki/File:BMPfileFormat.png.

3.2 Loading the BMP header

This is example of code to load the BMP file header (the first 14 bits). Figure 2 shows the output.



```
signature: BM
size: 70
reserved1: 0
reserved2: 0
offset_pixel_array: 54
```

Figure 2: Output of base file on test2.bmp.

```
//sample code to read first 14 bytes of BMP file format
FILE* file = fopen("test2.bmp", "rb");
struct BMP_Header header;

//read bitmap file header (14 bytes)
fread(&header.signature, sizeof(char)*2, 1, file);
fread(&header.size, sizeof(int), 1, file);
fread(&header.reserved1, sizeof(short), 1, file);
```

```
    fread(&header.reserved2, sizeof(short), 1, file);
    fread(&header.offset_pixel_array, sizeof(int), 1, file);

    printf("signature: %c%c\n", header.signature[0], header.signature[1]);
    printf("size: %d\n", header.size);
    printf("reserved1: %d\n", header.reserved1);
    printf("reserved2: %d\n", header.reserved2);
    printf("offset_pixel_array: %d\n", header.offset_pixel_array);

    fclose(file);
```

The key functions here are:

- `FILE *fopen(const char *filename, const char *mode)`

  This creates a new file stream. fopen is used with the "rb" mode to indicate we are "r"eading a file in "b"inary mode. To read a file, use the mode "wb" instead of "rb".

- `size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream)`

  For each call to fread, we give it first a pointer to an element of a struct containing the BMP header, then the number of bytes to read, the number of times to read (typically 1), and the file stream to use. Note that the order of the calls to fread defines the order in which we read data so the order must match the file layout. (There is also a function called fwrite which works in exactly the opposite manner. The first parameter won't be a pointer though.)

One function not used here but which may be useful is fseek:

- `int fseek ( FILE * stream, long int offset, int origin )`

  The purpose of fseek is to move the "reading head" of the FILE object by some number of bytes. It can used to skip a number of bytes. The first parameter to fseek is the file pointer, followed by a number of bytes to move, and an origin. For origin, SEEK_CUR is a relative repositioning while SEEK_SET is global repositioning.

The basic idea to support loading a BMP file will be to parse it by byte-byte (using fread) into a set of structs. Later, you can use fwrite to write out the contents of those structs to a file stream to save the filtered image.

3.3 Making BMP file headers

When creating a new BMP, especially one that has been resized, you will need to fill in the header of the new file yourself. For example, you will need to fill in the compression type, the number of color planes, etc for BMP files. The following defines default values you should use when creating an image. All values that are not defined here, you should calculate based on the input image.

- BMP Header:
  - Signature: "BM"
  - Reserved 1: 0
  - Reserved 2: 0
- DIB Header:
  - Planes: 1
  - Compression: 0
  - Horizontal resolution: 3780
  - Vertical resolution: 3780
  - Color number: 0
  - Important color number: 0

4 Include Files

To complete this assignment, you may find the following include files and functions useful:

- *stdio.h*: Defines standard IO functions.
- *stdlib.h*: Defines memory allocation functions.
- *string.h*: Defines string manipulation functions.
  - char* strcat(char* dest, const char* src): The strcat() function appends the string pointed to by src to the end of the string pointed to by dest.
  - char* strcpy(char* dest, const char* src): The strcpy() function copies the string pointed to, by src to dest.
  - size_t strlen(const char* str): The strlen() function computes the length of the string str up to, but not including the terminating null character.
  - char* strtok_r(char* str, const char* delim, char** saveptr): The strtok_r function parses a string into a sequence of tokens.
- *unistd.h*: Defines POSIX operating system API functions.
  - int getopt(int argc, char *const argv[], const char *optstring): The getopt() function is a builtin function in C and is used to parse command line arguments.

Your solution may not include all of these include files or functions, they are only suggestions. If you want to include any other files, please check with the instructor or TA before doing so.

5 Submission

The submission for this assignment has one part: a source code submission. The files should be zipped in a file named "LastNameImageProcessor.zip" (e.g. "EdgarImageProcessor.zip") attached to the homework submission link on Canvas.

Writeup: For this assignment, no write up is required.

Source Code: Please name your main file as "LastNameImageProcessor.c" (e.g. "EdgarImageProcessor.c"). Your ZIP should also include: BMPHandler.c, BMPHandler.h, Image.c, and Image.h.