Arizona State University SER334:

Operating Systems and System Programming

Image Processing

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

1 Background

In this assignment you will write a program that applies a filter to an image. The image will be loaded from a local BMP or PPM file, specified by the user, and then be transformed using a color shift also specified by the user. The resulting file will be saved back to a BMP or PPM file that can be viewed on the host 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 and PPM file formats 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 or PPM file, applying a color shift to its pixel data, and saving the modified image back into a BMP or PPM file. 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 – chosen as it is the default for Windows 10 BMP format (that means compression method is BI_RGB), which have a 14-byte bmp header, a 40-byte dib header, and a variable length pixel array. This is shown in Figure 2. PPM files will also be in 24-bit. For testing, please only use the PPM and BMP files attached to the assignment.

As a base requirement, your program must compile and run under Xubuntu (or another variant of Ubuntu) 18.04. Sample output for this program is shown in Figure 1.

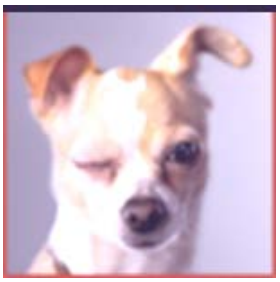
- Specific Requirements:
  - BMP Headers IO: Create structures for the headers of a BMP file (BMP header struct and DIB header struct) and functions which read and write them. [4 Points]
  - PPM Headers IO: Create structures for the headers of a PPM file (PPM header struct) and functions which read and write them. [4 Points]
  - Pixel IO: Create a structure which represents a single pixel (24-bit pixel struct) and the functions for reading and writing all pixels in both PPM and BMP files. [4 Point]
  - 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 option list. The output file option is specified by “-o” followed by the output file name. Validate that the input file exists and that both files are of the correct type (either bmp or ppm). [4 Points]
    - 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).
  - Output file format: The user optionally can set the output file type to either PPM or BMP by using the “-t” option. The only valid options for type should be “PPM” or “BMP”. If no format is chosen, then the default is “BMP”.

- RGB Color Shift input: Accept options for red, green, or blue color shift. These are specified by“-r” “-g” or “-b” followed by an integer. Validate that these are integers. Like the “-o” option described above, these can come in any order in the option list and are optional for the user to enter. [4 Points]
  - Note: “-o” “-r” “-g” and “-b” “-t” are all options. All for of these can come in any order (-r -g -o -b, for example) and none of them are required.
- Copy images: Correctly copy images from a BMP file to a new BMP file and from a PPM file to a new PPM file. [4 Points]
- Convert images: Correctly copy images from a BMP file to a new PPM file and from a PPM file to a new BMP file. [4 Points]
- Color Shift calculation: 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 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).
- Modular Programming: Use the provided header files correctly and create corresponding C files, along with a main file. You do not have to create any more header files or C files than this but you may if it helps you. Only modify the headers files where it says TODOs or to include other relevant header files.[4 Points]

Figure 1 shows an example of how the finished program should function IF IT is UNCLAMPED.

Figure 1: Example of running the program with example output without clamping.

This is the result if you clamp it correctly.



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 2, a graphical overview of a BMP file's layout is shown. The layout is literally the meaning of the bits/bytes in the file

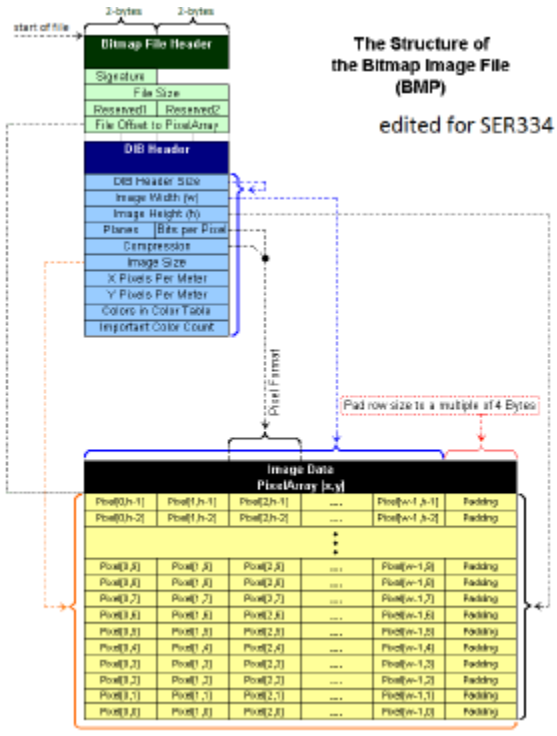


Figure 2: BMP file format structure for 24-bit files without compression. Image modified from https://en.wikipedia.org/wiki/File:BMPfileFormat.png.

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 a 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 HxD.

3.2 The PPM File Format

The PPM file format is a much simpler format. Xubuntu supports opening PPM by default, but to open it in Windows or MacOS you may use GIMP: https://www.gimp.org/. For reference, please visit the official specification here: http://netpbm.sourceforge.net/doc/ppm.html. The header of a PPM file consists of the signature “P6”, follow by white space, the width of the file (in ASCII characters, not binary), white space, and the height of the image (in ASCII characters, not binary). After the header, there is a single white space character followed by the pixel array. The pixels are 3 bytes each, with each color being represented by a single byte. The columns are stored left to right and the rows top to bottom. **Do not use the Plain PPM format that is described in the spec, use the regular one.** From top to bottom, each PPM image text file consists of the following:

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6". Whitespace (blanks, TABs, CRs, LFs).
2. A width, formatted as ASCII characters in decimal. Whitespace.

3. A height, again in ASCII decimal. Whitespace.
4. The maximum color value (*maxval*), again in ASCII decimal. Must be less than 65536 and more than zero. A single whitespace character (usually a newline).
5. A raster of *height* rows, in order from top to bottom. Each row consists of *width* pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the *maxval* is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

Plan to review examples of PPM on PPM Wikipedia page (https://en.wikipedia.org/wiki/Netpbm_format#PPM_example) to get a feel for the contents of each of these regions. Some PPM header examples are available on http://paulbourke.net/dataformats/ppm/. You should include all necessary information of the header in the PPM struct.

3.3 Loading the BMP header

This is example of code to load the BMP file header (the first 14 bits). Figure 3 shows the output.

Figure 3: 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 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 order of the calls to fread define 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 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.

3.4 Creating file headers

When copying from BMP to BMP or PPM to PPM, you could easily copy the original file's header into the new one. But when copying from BMP to PPM, or PPM to BMP, 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
- PPM Header:
  - Magic Number: “P6”
  - Maximum color value: “255”

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. **Note: Since PPM files are in ASCII (not binary), you should write and read them differently. fprint and fscanf would be a good choice to write and read PPM files.**

NOTE: Highly recommended that you read and utilize the programming aid posted along with this assignment to help you complete this assignment.

5 Validation

So to help you verify your work is proceeding here are some values for the sample filenames to help you thru your development.

```
COMMAND LINE ARGUMENTS
 Input: ../ttt.bmp,
 Output: out.ppm,
 RGB: 0 0 0


INPUT HEADER
signature: BM
size: 69366
reserved1: 0
reserved2: 0
offset_pixel_array: 54
DIB header size:40
Image width: 152
Image height: 152
Planes: 1
Bits per pixel: 24
Compression: 0
Image size: 69312
X resolution: 3780
Y resolution: 3780
Colors in color table: 0
Important color count: 0

OUTPUT HEADER
Magic number: P6
Height: 152
Width: 152
Maxval: 255
```

```
COMMAND LINE ARGUMENTS
 Input: ../nehoymenoy.ppm,
 Output: out.bmp,
 RGB: 0 0 0


INPUT HEADER
Magic number: P6
Height: 152
Width: 152
Maxval: 255

OUTPUT HEADER
signature: BM
size: 69366
reserved1: 0
reserved2: 0
offset_pixel_array: 0
DIB header size:40
Image width: 152
Image height: 152
Planes: 1
Bits per pixel: 24
Compression: 0
Image size: 69312
X resolution: 3780
Y resolution: 3780
Colors in color table: 0
Important color count: 0
```

Both courtesy of : Trent Engelman

6 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.

REMEMBER: You must turn in **all files - both *.c and *.h** needed to compile, build, and run your code. Per syllabus, all files must be there and compile out of the box to avoid the 20% non-compile out-of-box penalty.

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

Source Code: Please name your main class as "LastNameImageProcessor.c" (e.g. "EdgarImageProcessor.c").

• If your program fails to compile out-of-the-box, there will be a mandatory deduction 20% from the assignment points.

• If you do not follow the file submission standards (e.g., the submission contains project files, lacks a proper header), there will be a mandatory deduction of 10% from the assignment points.

• If compiling or running your program differs from stated on the assignment and it is not specified as a requirement, please include a readme file with steps to execute the program. If you do not, there will be a mandatory 10% deduction from the assignment points.

Example readme

Usage:
Open CLI and navigate to this directory.

```
gcc ReynoldsImageProcessor.c PixelProcessor.c BmpProcessor.c PpmProcessor.c -o ImageProcessor
./ImageProcessor ttt.bmp -o ttt.ppm
./ImageProcessor ttt.bmp -r 89 -o ttt.ppm
```