

Computer Science 215
Tools and Techniques for Software Development
Fall 2001
Project 5 – Image Manipulator

Due: Tuesday, 11/27/2001 9:00 a.m.

For this assignment, you will write a program that provides some simple image manipulation utilities.

In computer graphics, images are composed of colored dots known as pixels. Each pixel value can be specified as a combination of three colors <R, G, B> where R is red, G is green, and B is blue. Each of these elements is an integer ranging from 0 to some maximum color value (in ppm files, the maximum color value is often 255). For example, the color red is represented by <255, 0, 0>, white is <255, 255, 255>, and black is <0, 0, 0>.

While there are many graphics file formats (e.g., jpeg, gif, tiff, targa, etc.), we will focus on ppm since it is the simplest graphics file format. A ppm file has a particular format that consists of a header section with general information about the image, and a data section with the actual RGB values for each pixel. The header information is always written as ASCII text, but the data section can be ASCII or binary.

The ppm file format is as follows:

```
Pn                - P3 = ASCII pixel data, P6 = binary pixel data
# comment        - optional comment line(s)
www hhhh         - width, hhhh
# comment        - optional comment line(s)
nnn              - maximum color value

RRR GGG BBB      - ASCII pixel data (for each pixel)
rr gg bb         - binary pixel data (for each pixel)
```

Note that the first line of the file indicates the type of pixel data present in the data section. Comments can be multiple lines as long as each line begins with #. The pixel data format will be either of the last two lines above, depending on the file type.

From the command line, the user will invoke the program in the following manner:

```
im <input file name>
```

where <input file name> refers to the ppm file containing the image. If the input file cannot be found, the program should print an error message and exit. Note that when reading in a ppm file, your program needs to determine if the file is ASCII or binary before reading the pixel data.

After reading the input image, the program should print a message about the input file and present a user menu as follows:

```
ASCII file "in.ppm" successfully read
```

```
Please select one of the following options:
```

- 1 Lighten image by 10%
- 2 Darken image by 10%
- 3 Flip image
- 4 Rotate image 90 degrees CCW
- 5 Write ASCII ppm file
- 6 Write binary ppm file
- 7 Quit

```
Option:
```

Of course, if the input file is a binary file, the message should begin "binary file ...". Numerical error checking for the menu item is required (i.e., make sure the value is in range). Information for each of the options follows.

1 **Lighten image by 10%** – When this option is selected, each R, G, and B value should be increased by 10%.

2 **Darken image by 10%** – When this option is selected, each R, G, and B value should be decreased by 10%.

3 **Flip image** – This option should flip the image along the vertical center line.

4 **Rotate image 90 degrees CCW** – This option rotates the image 90 degrees counterclockwise.

5 **Write ASCII ppm file** – When this option is selected, the user should be prompted for an output file name and print a message when complete:

```
...
```

```
Option: 5
```

```
Please enter output file name: out.ppm
```

```
ASCII file "out.ppm" successfully written
```

where `out.ppm` is the file name entered by the user. If for some reason the file cannot be written, the program should print out an error message and return the user to the menu.

6 **Write binary ppm file** – This option works similarly to option 7, but with a binary output file.

7 **Quit** – The program should terminate when this option is selected.

All of the ppm functions should be in a file called `ppm.c`, while the rest of the functions will reside in `im.c`. Create header files for each and include where necessary.

For this program, you **must** use **structs** and **typedefs** for the following types:

COLOR_T – a struct containing three unsigned characters for **R**, **G**, and **B**
IMAGE_T – a struct containing four fields: an integer **width**, an integer **height**, an integer **max_color**, and a double pointer to **COLOR_T** called **pixels**

The **pixels** field in **IMAGE_T** must be allocated memory during program execution. It is a double pointer since pixel data is usually specified by a two-dimensional array. After memory allocation, you should be able to index the individual elements using *x* and *y* coordinates.

You **must** define functions with the **exact** prototypes listed below (these will *help* you):

```
void lighten_image (IMAGE_T *im);
void darken_image (IMAGE_T *im);
void flip_image (IMAGE_T *im);
void rotate_image (IMAGE_T *im);
void read_ppm_file (IMAGE_T *im, char *filename);
void write_ascii_ppm_file (IMAGE_T *im, char *filename);
void write_binary_ppm_file (IMAGE_T *im, char *filename);
```

lighten_image (IMAGE_T *im) runs through all the pixel values and increases their values by 10%. If a value exceeds the **max_color** value, it should simply be set to **max_color**.

darken_image (IMAGE_T *im) runs through all the pixel values and decreases their values by 10%.

flip_image (IMAGE_T *im) reverses the image from right to left. You may allocate new space for the flipped image (and free the old space) if you like.

rotate_image (IMAGE_T *im) rotates the image counterclockwise by 90 degrees. Note that the dimensions of the image may change; therefore, you should allocate new space of the appropriate size for the rotated image and free the old space.

read_ppm_file (IMAGE_T *im, char *filename) opens the file name **filename** for reading. After reading the header information, it reads the pixel data in either ASCII or binary format. All read values are stored in **im** for later use. You may want to create two additional functions, **read_ascii_ppm (IMAGE_T *im, FILE *fp)** and **read_binary_ppm (IMAGE_T *im, FILE *fp)**, for reading the pixel data.

write_ascii_ppm_file (IMAGE_T *im, char *filename) opens the file name **filename** for writing and writes out the header and ASCII pixel data. In the first set of comments, include your name on one line, and the month and year on the next. For simplicity, write out 8 RGB values per line in the output file.

`write_binary_ppm_file (IMAGE_T *im, char *filename)` opens the file name `filename` for writing and writes out the header in ASCII format and the pixel data in binary format. In the first set of comments, include your name on one line, and the month and year on the next.

Compile the program as follows:

```
cc im.c ppm.c -o im
```

A sample run, along with sample images, are available off the class web page. Use `xv` to verify your output results.

Extra Credit

[10 points] Write a program called `ppmdiff` that takes two files as command line arguments and determines if their pixel values are exactly the same. Your program should be able to handle both ASCII ppm files and binary ppm files of any arbitrary size. Comparisons between two ASCII files, two binary files, or one ASCII and one binary file should be possible.

Your code should be well-designed, commented, and submitted with your other files.