Assignment P5

Due Date: April 14

Purpose

In this assignment, you will create a class that stores an image. You will get more practice with arrays including dynamic memory allocation. The project also entails writing separate header, implementation, and application (main) files.

Problem

There are many image file formats. Most of these, such as JPEG or PNG, compress the image and store the data in a way that is not easy to process. However, the PPM, or Portable Pixel Map, format is uncompressed and can be stored in ASCII form, which means plain ol' human text. In the project, you will look at some PPM images and find that they are difficult to decipher. Your program will read a PPM image file, process it ("image processing"), and recreate the original image.

Input

The input is file in PPM format that represents an image. Such a file looks as follows:

```
P3
100 120
255
 59  99 164   61  99 162   62 100 162   66 101 162   64 100 160
 66 102 162   63 101 161   65 101 161   66 101 164   68 101 154
     etc.
```

- The first line is the "magic number;" "P3" identifies the file as one in the PPM format.
- The second line shows the width w (columns) followed by the height h (rows) **in that order!**
- The third line shows the maximum color value. While this line will be in the file, we will ignore that value.
- What follows is h rows of $3w$ columns, where each set of three values represents one pixel in the image. This is because color is represented as an integer 0..255 in each of red (R), green (G), and blue (B), which comprises the aptly named RGB system. In each color, 0 represents the absence of that color and 255 represents the brightest value for that color. Thus, in the file above, the first pixel is 59, 99, and 164. Red is 59/255 so a rather dark red, green is 99/255, so a darkish green, and blue is 164/255 which is a medium blue. The combination of these colors becomes the color of that pixel. More of this will be discussed in class.

Your program should prompt for the name of such a PPM file and store it in an array defined within a class (see below). The program should then prompt for three floating point values which represent the multipliers for each color. This is how you will transform the file from one that looks like nothing to one that clearly shows an image (not necessarily beautiful).

Output

The program should produce an output file called "output.ppm." This file will be the processed file and should look reasonably realistic. To create this file, each of RGB of each pixel in the original file must be multiplied by the respective input multiplier.

Specifics

- You must use a class. To make this easier, use the class definition shown below:

```
class image {
   struct ppm {
      int red;
      int green;
      int blue;
   };
   ppm **pixels;
   int rows;
   int cols;
public:
   image();
   void readFile();
   void fixColors();
   void writeFile();
};
```

  This class should not be altered, **except** that comments are allowed, and **additional** methods can be added, if desired. This class should be in a separate header file, which can also contain include directives and namespaces (which are not part of the class).

- As shown above, the 2D array that stores the image must be dynamically allocated. An example of how this is done is as follows:

```
pixels = new ppm*[rows];
for (i = 0; i < rows; i++) {
   pixels[i] = new ppm[cols];
}
```

  Once the array has been allocated, you can then use the [ ] as you normally would; that is, you don't have to do pointer arithmetic.

- The entire `main()` function should look as follows; no alterations are allowed (except for comments or if you add functionality).

```
int main() {
   image picture;

   picture.readFile();
   picture.fixColors ();
   picture.writeFile();

   return 0;
}
```

  This should be in a separate application program file. Using the file `main` that CLion creates for you is acceptable.

- All of your methods should be no longer than 25 lines. Remember that you may add additional helper methods if needed.

- Remember to properly comment all of your code, including the functions and all parameters.

- Whenever using files, you should test that the file pointer is actually valid before continuing to process!

Notes

Be sure that you are reading the input file correctly. You can test this by making a tiny sample PPM file, reading it, and simply displaying it with `cout` statements as you are reading it. *Segmentation faults or other crashes are common in this kind of program!*

Study the given class carefully before trying to write your code. Understanding exactly how things are stored is critical to reading the file and doing the image processing.

You will have at least three files to submit (header, implementation, and main application). Zip these together using the usual naming conventions with `.zip` as the file extension. For example, my project would be `mgousieP5.zip`.

To say of a picture, as is often said in its praise,
that is shows great and earnest labor,
is to say that it is incomplete and unfit for view.
– James McNeill Whistler