Homework 2 – Image Conversion

Due: October 3rd (11:59pm)

Overview:

In this homework assignment you will implement a basic image conversion program. The goal of the homework is to build familiarity with low-level details of image file formats, build low-level C++ programming skills, and gain experience with creating and using command-line programs.

Getting Started:

You should use the skeleton code on the course webpage as a starting point for the assignment. There are several files that are provided for you to setup the framework and read/write image files, but you should only change `image.cpp`.

The files that need to be in your project are:

- `main.cpp` – Parses the command line arguments, calls the appropriate image functions
- `stb_image.h` – Reads images files (.bmp, .jpg/.jpeg, .png, .tga)
- `stb_image_write.h` – Write images files (.bmp, .jpg/.jpeg, .png, .tga)
- `pixel.[h/cpp]` – Pixel processing
- `image.[h/cpp]` – Image Processing **(The only file you need to edit!)**

A small number of starter images are included with the code, but you can find more in your personal collection or online (https://www.google.com/imghp). The images you use must be one of the supported formats (.bmp, .jpg/.jpeg, .png, .tga, .ppm).

Completing this homework will directly prepare you to finish Project 2.

How the Program Works:

The UI in this assignment was kept as simple as possible, so you can concentrate on the image processing. The program runs on the command line and performs operations in the order that they appear in the arguments. For example, to increase the brightness of the image `in.bmp` by 10%, and save the result in the image `out.bmp`, you would type:

```
image -input in.bmp -brightness 1.1 -output out.bmp
```

Notice the `input` parameter must appear first. Remember, everything happens in the order specified. First the input, then the brightness change, then the writing to the specified output file.

To see the complete list of commands options, type:

```
image –help
```

If you specify more than one option, the options are processed in the order that they are encountered. It is also possible to specify `-output` multiple times, to save out intermediate results:

```
image -input in.bmp -output converted.bmp -crop 10 10 50 50
-output cropped.jpg -brightness 1.1 -output crop_bright.png
```

What You Have To Do:

File formats – Supporting Pixel Depth in PPMs

The provided code uses the popular STB_Image library to read and write images in the *.bmp*, *.jpg/.jpeg*, *.png*, and *.tga* file formats. Additionally, inside `image.cpp` are functions for reading and writing files in the *.ppm* format. These functions both have limitations that you need to fix as part of this homework:

1. The function `write_ppm()` currently ignores the `bits` parameter. This parameter says how many bits to use for each channel of each pixel. Update this function so that the file writes out with the correct image depth precision. This should set the maximum value in the ppm file to *2^bits*, and round each number to be relative to this new maximum.

2. The function `read_ppm()` currently ignores the maximum value parameter of the .ppm file. Update this function so that the file converts whatever value is read to be proportional to 255.

Internally, the `Image()` class represents pixels with 8 bits (values 0-255). You can assume the `bits` parameter in `write_ppm()` will only ever be 1-8. Likewise, the maximum value in a .ppm file will only be the following values: 1, 3, 7 15, 31, 63, 127, or 255. The pixel depth the *.ppm* file should be written to is set by the `-ppm_depth` command to the image program.

Image Quantization

The pixel depth discussed above only affects the values written to or read from files. Separately, we can also quantize the represented value to a certain number of bits. This will not actually affect the file size or the maximum pixel value written, it just quantizes the internal representation. In implementing this feature, you may find the `Pixel::PixelQuant()` function to be helpful.

Image Cropping

You must also implement the ability for the image to extract a sub-image specified by a top-left corner along with a width and a heigh. Most of the infrastructure is already set-up for you, but you need to implement the actual cropping function, `Image::Crop()`.

Extract Channel
Finally, you need to support an operation which leaves the specified channel intact and sets the other channels to 0. Again, most of the infrastructure is already set-up for you, but you need to implement the actually channel extraction function, `Image::ExtractChannel()`.

Brightness
Changing the brightness has already been implemented for you. Inspecting the function `Image:: Brighten()`, may be helpful for understanding how the `Image` and `Pixel` classes work, and help provide ideas about how to best implement pixel quantization, image cropping, and channel extraction.

Example Results
The results in the ExampleResults folder were achieved as follows:
./a.out -input goldy.ppm -quantize 2 -output goldy_quantize_2.png
./a.out -input goldy.ppm -quantize 2 -output goldy_quantize_2.ppm
./a.out -input goldy.ppm -ppm_depth 2 -output goldy_depth_2.ppm
./a.out -input goldy.ppm -ppm_depth 8 -output goldy_depth_8.ppm
./a.out -input goldy.ppm -crop 250 50 100 150 -output goldy_cropped.ppm
./a.out -input goldy.ppm -crop 250 50 100 150 -extractChannel 1 -output goldy_cropped_green.jpeg
./a.out -input goldy.ppm -brightness 1.4 -crop 250 50 100 150 -extractChannel 0 -output goldy_cropped_bright_red.jpg

Submission Details:
You will turn in only the file *image.cpp*. Additionally, your code must be able compile using the following command on the lab machines:

```
g++ -fsanitize=address -std=c++14 main.cpp pixel.cpp
image.cpp -o image
```

There are a few example files provided for comparison. You should also run your own tests.

Discussion questions:
Not graded, but you should know the answers!

1. What is the difference between setting the pixel export depth (`-ppm_depth`) and quantizing a value (`-quantize`)?

2. How much space would you expect to save going from a file with a depth of 8 bits per channel to one with 2 bits per channel? What savings do you actually see? Why?