

CptS 122 Assignment #4

This assignment is a continuation of lab 7. In this homework, you will create a program that will add simple effects to a PPM image.

Prerequisites

Before beginning this assignment, you will need some sort of imaging software that can open PPM-based images. I recommend [Ifranview](#) for Windows as it's free and lightweight.

PPM Image Format

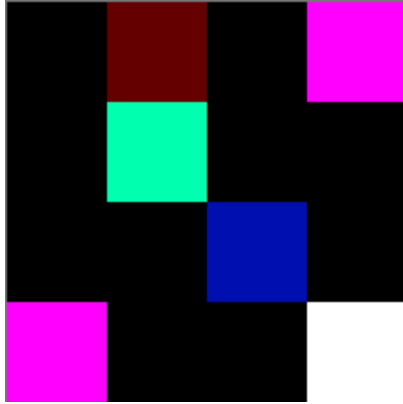
The PPM image format is encoded in human-readable ASCII text. It might be helpful to read over the [formal PPM specification document](#). A PPM document has two pieces: the header and the body. The header is always the top three uncommented lines in the file:

```
P3
4 4
255
```

The first line specifies the type of image that is contained within the file. We will always use the "P3" specification. The second line specifies the number of columns and rows present in the image. In this example, we have a 4x4 image. The final number indicates the maximum value for each red, green, and blue element in the picture. Having a max value of 255 is quite common and is the value that we will always use. Below the header is the body of the PPM file. Each pixel has a red, green, and blue value. For example, the content of our 4x4 image might be:

```
0 0 0      100 0 0      0 0 0      255 0 255
0 0 0      0 255 175    0 0 0      0 0 0
0 0 0      0 0 0      0 15 175    0 0 0
255 0 255  0 0 0      0 0 0      255 255 255
```

With these values, the pixel in the first column and row has a RGB value of (0,0,0) and the last column in the first row has an RGB value of (255,0,255). A blown-up image containing these values would look like:



Task

Your task is to write a program that will perform the following basic image manipulations on a user-specified image. Below is a reference value and the images that result from the image processing.

Original Image

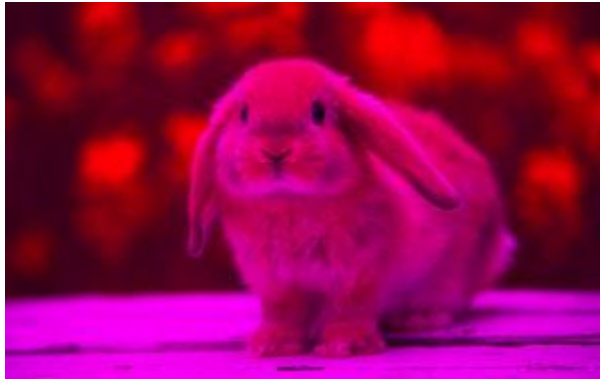


Remove Red



Will remove all red (i.e. set red to 0) pixels in the image

Remove Green



Will remove all green pixels in the image

Remove Blue



Will remove all blue pixels in the image

Negate Red



Will negate all red pixels in an image. To negate a color, simply subtract the current pixel's red color value from 255.

Negate Green



Will negate all green pixels in an image.

Negate Blue



Will negate all blue pixels in an image.

Add Random Noise



Will add random noise to the image by randomly selecting a value ranging from -10 to 10 and adding that value to the pixel's current value. If the addition causes the pixel value to be less than 0, then set to 0. Likewise, if the addition causes the pixel's value to be greater than 255, just set the pixel's value to 255. For example, we randomly generate the number -7. Next, we add -7 to each of the current pixel's red, green, and blue values.

High Contrast



Examine each color in a pixel. If the color is greater than half of 255, then max the color out (i.e. set to 255). Otherwise, set the value to 0. For example, performing a high contrast operation on a pixel whose RGB values are 170, 50, 100 would result in an RGB value of 255, 0, 0.

Grayscale



Converts the image to grayscale by setting the value of each color in a given pixel to the average of all three values. For example, if a pixel's RGB was 177, 15, 39, the resulting RGB value would be 77, 77, 77.

BONUS - Flip Horizontal



Flips the image along the horizontal axis.

BONUS - Flip Vertical



Flips the image along the vertical axis.

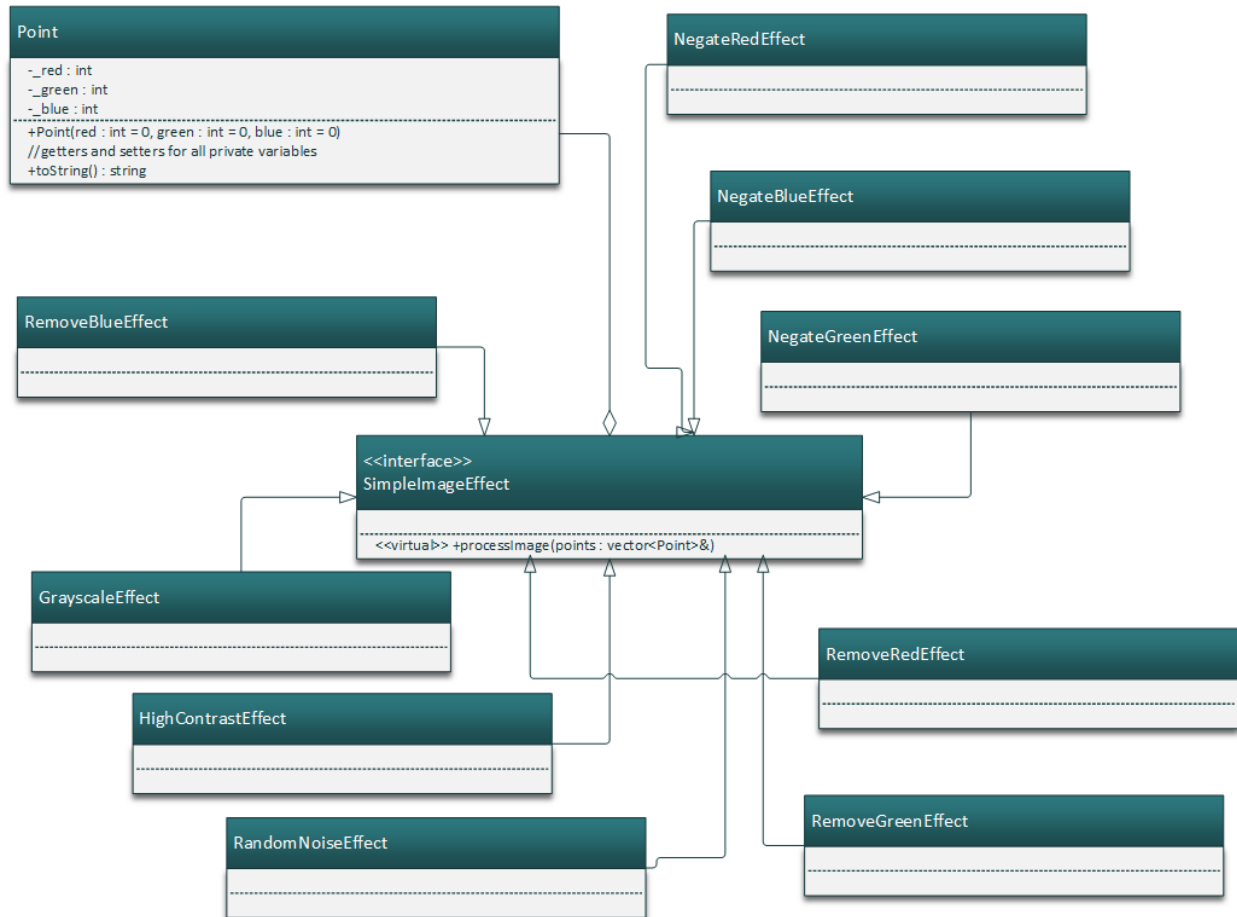
Chaining Effects Together

Note that manipulations can be chained together. Below is the image generated from performing a negate red, negate green, and negate blue operation.



UML Diagram

In order to complete this assignment, you must develop the following classes:

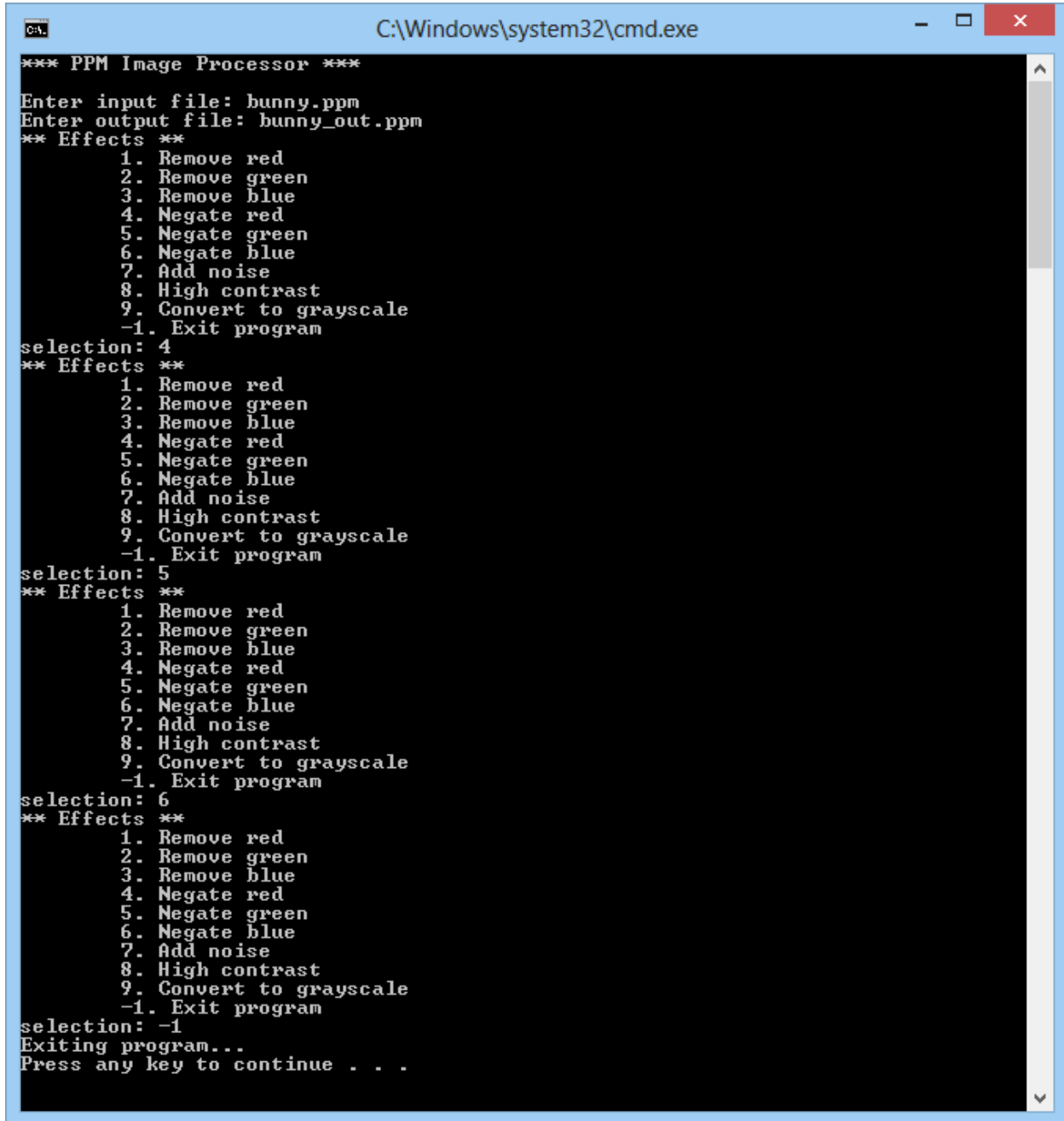


Note that it might help to zoom in on the UML diagram. Basically, we have our `Point` class that we developed in Lab 7. In the center, we have the `SimpleImageEffect` interface that defines a single pure virtual function called `processImage`. Each effect described in the previous section derives itself from `SimpleImageEffect`, implementing the `processImage` method in such a way to fulfill its role (e.g. `NegateRedEffect` will negate all red pixels in the image). Note that while a factory-type class isn't defined, it may be useful for you to write one.

Also note the white diamond going from `SimpleImageEffect` to `Point`. This is called "aggregation" and is another form of a "has-a" relationship. The main difference between aggregation and composition (black diamond) has to deal with the lifespan of an object. A black diamond means that the object goes out of scope at the same time as the owner class. A white diamond means that the object stays alive even when the owner class goes out of scope. If you think about the lifespan of a `Point`, it is easy to see that a given `Point` will exist for a longer period than the `SimpleImageEffect`. This makes sense as the `Point` still needs to be written to the output file!

Sample Output

Below is the sample output from my program:



```
*** PPM Image Processor ***
Enter input file: bunny.ppm
Enter output file: bunny_out.ppm
** Effects **
  1. Remove red
  2. Remove green
  3. Remove blue
  4. Negate red
  5. Negate green
  6. Negate blue
  7. Add noise
  8. High contrast
  9. Convert to grayscale
 -1. Exit program
selection: 4
** Effects **
  1. Remove red
  2. Remove green
  3. Remove blue
  4. Negate red
  5. Negate green
  6. Negate blue
  7. Add noise
  8. High contrast
  9. Convert to grayscale
 -1. Exit program
selection: 5
** Effects **
  1. Remove red
  2. Remove green
  3. Remove blue
  4. Negate red
  5. Negate green
  6. Negate blue
  7. Add noise
  8. High contrast
  9. Convert to grayscale
 -1. Exit program
selection: 6
** Effects **
  1. Remove red
  2. Remove green
  3. Remove blue
  4. Negate red
  5. Negate green
  6. Negate blue
  7. Add noise
  8. High contrast
  9. Convert to grayscale
 -1. Exit program
selection: -1
Exiting program...
Press any key to continue . . .
```

Note that the program allows for unlimited effects to be added to a given image. Also note that some of these effects might take a while for your computer to process!

Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your script to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. A basic list of rules can be found on OSBLE.

Deliverables

You must upload your program store through OSBIDE no later than midnight on Friday, March 14, 2014. For more information on how to submit an assignment, [read this page](#).

Grading Criteria

Your assignment will be judged by the following criteria:

Error Free Compile (weight: 5%)

- [0] Your program contains compiler errors.
- [1] Your program compiles without issue.

Error Free Runtime (weight: 10%)

- [0] Your program throws a runtime exception.
- [1] Your program does not encounter any runtime exceptions.

Pointer Cleanup (weight: 5%)

- [0] Your program does not delete any dynamically-created pointers
- [1] Your program remembers to delete some, but not all dynamically-created pointers
- [2] Your program deletes all dynamically-created pointers

Style (weight: 5%)

- [0] Your program does not contain good style. Variables are poorly named, your program doesn't contain a header, there is little or no documentation, your document isn't properly formatted (indentation, whitespace, etc.), and is generally hard to read.
- [1] Your program contains okay style. Some of the items listed in the [0] section are accounted for
- [2] Your program contains great style. Almost everything in listed in the [0] section is accounted for.

User Interface (weight: 10%)

- [0] Your program does not even attempt to follow the UI guidelines
- [1] Your program's UI has major inconsistencies when compared to the sample output
- [2] Your program's UI has minor inconsistencies when compared to the sample output
- [3] Your program completely matches the specified user interface guidelines

Class Correctness (weight: 15%)

- [0] Your program does not contain nor use any of the classes specified in the UML diagram
- [1] Your program contains but does not use the classes specified in the UML diagram
- [2] Your program contains and partially uses the classes specified in the UML diagram
- [3] Your program contains and uses the classes specified in the UML diagram.

Required Functionality (weight: 5% / ea)

- Remove red
- Remove green
- Remove blue
- Negate red
- Negate green
- Negate blue
- Add random noise
- High contrast
- Grayscale

Compound Operations (5%)

- [0] Your program does not allow for compound image manipulations (i.e. cannot combine multiple image manipulations in a single program run)
- [1] Your program allows for compound image manipulations, but does not perform them correctly (i.e. things don't output as they should)
- [2] Your program allows for compound image manipulations and performs them correctly.

EXTRA CREDIT (5%)

- [0] You do not attempt the extra credit
- [1] You add either the ability to flip the image horizontally or vertically.
- [2] You add the ability to flip the image both horizontally and vertically.