Advanced Technical Assessment

Objective

The task is to implement an application that is able to perform various image processing operations on an input image.

The image processing part of the application should be written without built-in or 3rd party image processing libraries. To clarify, this restriction does not apply for other aspects (e.g. loading an image file).

We are looking for the ability to write code that is easily extendable using good OOP design. Please try to maximize code reuse, and anticipate future extensions. Yes, premature generalization of code is bad in most cases, but ignore that for this assessment. :)

Additional notes:

- Feel free to use the internet to appropriate resources. Please send us the list of resources that you used for this assessment along with the final code submission.
- Feel free to change the code that is already submitted/committed for subsequent sections. But all functionalities should work in the final submission.

Deliverables

This task is to be completed using any of the following languages: C++, Objective-C, Java, Python.

Please submit your code according to one of the following two options:

1. **If you like to use git:** Tag (git tag) the application when each of the parts are finished, and send us the zipped repository at the end.
2. Otherwise, after completing each part, please zip up the implementation at that point and send us the code. There should be 5 distinct zip archives sent.

Background

Input files are images in 2D grayscale 8-bit arrays (pixel values of 0 is black, 255 is white). These image operations will use the pixel values from the original image to create the new image. Parts 1 and 2 involve applying two different **linear image filters** on the input image, with the **mask** provided below. The masks are applied by performing **convolution** on the image with the mask. Note that depending on the convolution, when a **convolution** is performed on an

image with a 3x3 mask, the resulting image may not be the same size. For this assessment, we will ignore the edge pixels in the image so this is not an issue.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Blur mask for part 1

-1/8	-1/8	-1/8
-1/8	2	-1/8
-1/8	-1/8	-1/8

Sharpen mask for part 2

See the following link for more information on image manipulation:
http://www.tutorialspoint.com/dip/index.htm

Assessment

Part 1 - Blur

Make it so that the application is able to blur the image (using the mask from above). The output image should look more blurry than the source image.
Example: input image: `whale_source.png`, output: `whale_blur.png`

Part 2 - Sharpen

Extend the application so that it is able to sharpen the image (using the mask from above) as well. The output image should look sharper than the source image.
Example: input image: `whale_source.png`, output: `whale_sharpen.png`

Part 3 - Thresholding

Extend the application so that the application is able to perform a threshold operation. Threshold operation is defined as setting pixels with any value greater than or equal to the threshold value to the maximum value (255), and setting the value of every other pixel to the minimum value (0). The inputs for this operation are the source image and an threshold value.
Example: input image: `whale_source.png`, threshold: `128`,
output: `whale_whale_threshold_128.png`

Part 4 - Salt and Pepper Noise Removal

Extend the application so that the application is able to remove a certain type of noise from the provided image. This noise is manifested on the image in the form of random 0-value and 255-value pixels.

Example: input image: `whale_salt_and_pepper.png`, output: similar to `whale_source.png`

Part 5 - Additive Noise Removal

Extend the application that can remove a horizontal sine noise from the provided image. The noise is responsible for the wave pattern, as seen on the sample image below. The noise is additive, and can be different for each image. In order to assist with the noise removal, a calibration image is provided. The calibration image is a solid grayscale image (every pixel value set to 128) that has been exposed to the exact same noise the input file is affected by.

Example: input image: `whale_sine.png`, calibration image: `calibration.png`, output: similar to `whale_source.png`

Part 6 - Testing

Write some automated tests to test your algorithm for part 5. This could be a unit test suite, a separate executable that returns pass or fail, or a special mode built into the application.