Evaluation Test Instructions

This test was made to leverage your skills on daily tasks expected to be performed by members of our imaging team. This document consists of a series of challenges, each one regarding its own level of complexity. The candidate will be evaluated according to the problems that he or she is able to successfully or partially accomplish, so, the candidate is encouraged to try its best to fulfill all the challenges, but it can be understandable if it does only a part of them. The problems are not ordered according to their complexity. It is important to present the results from all your tries of each challenge, even if it is not complete.

Attached together with this document, you have received the base code (main.cpp) that you will work on.

You should not alter any files besides main.cpp, but you may create as many files as you need. Have in mind that you are allowed to use standard libraries from C++11 or newer, and the methods from **ImageHelper** class only.

WARNING: Do not use any other function from any other external library that you didn't implement by yourself, otherwise, you will be disqualified!

The test consists of six tasks in which you must create filters to be applied on the given input sample image Lena.bmp. Your program must create an output with the filtered image as a result. To check your results, output examples of what we expect are given.

We remind you that the usage of preexisting libraries is not allowed, you must create your own methods and functions to obtain the required results using **Object Oriented Programming**.

The code was tested on Dev-C++ 5.11. If you want to use another IDE, check the CImg documentation to make it work properly.

Estimated effort : **4 hours**
Due date : **48 hours after task assignment**

Important note:

The variable `unsigned char *image` holds the reference to the input image that you will use during this test. The values are ordered first along the width, height and channel, starting from the upper-left pixel to the bottom-right pixel of the image, with a classical scanline run. Therefore, a RGB color image will be stored in memory as:

R1	R2	R3	R4	...	G1	G2	G3	G4	...	B1	B2	B3	B4	...

(scanline run),

and not as

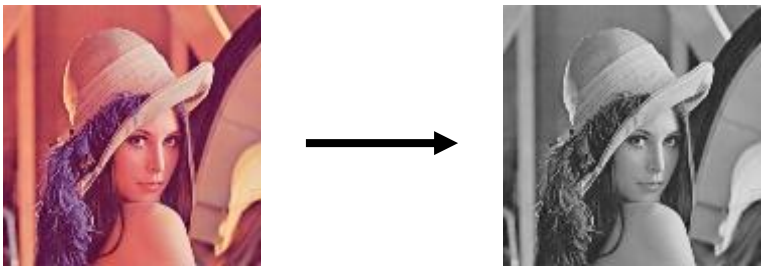
R1	G1	B1	R2	G2	B2	R3	G3	B3	R4	G4	B4	R5	G5	...

(interleaved channels),

Where *Rn* is the value stored to represent the intensity of Red pixel *n*. The same rule can be extended to Green and Blue pixels.

Create an ImageEditor class that will implement the following filters:

1) Convert the image from RGB to Grayscale;

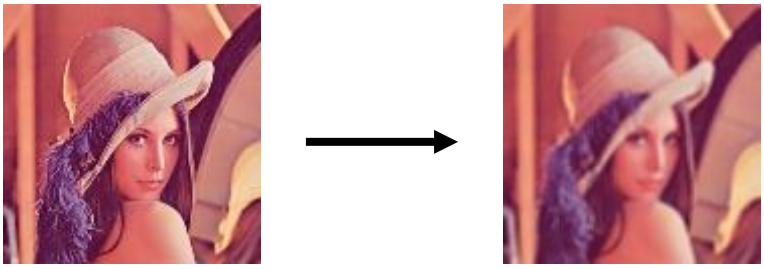


2) Apply a blur effect to the image;

The blur effect can be seen as smoothing out an image. This can be done in many ways. The method you are required to use is a simple one: by taking each pixel and averaging with its neighbors. Using a 3x3 kernel size, the conversion can be obtained using the following:

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Average



Note: the output image should have the same width and height as the input image.

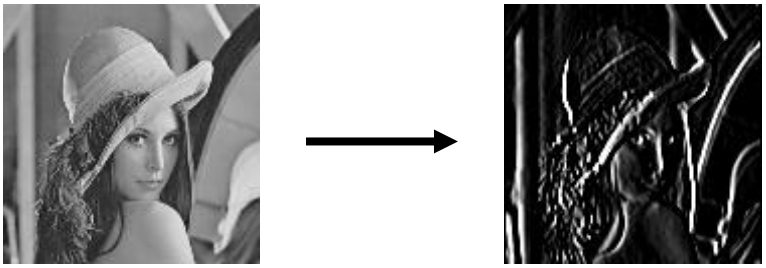
3) Apply a horizontal sobel filter to the image

The sobel filter is usually used as an edge detector. You should use the grayscale image from task 1 and apply a horizontal sobel filter with the following kernel:

-1	0	1
-2	0	2
-1	0	1

Horizontal

The expected result should be similar to the following example:



Note: the output image should have the same width and height as the input image.

4) Obtain a edge mask using Sobel filters

Implement a vertical sobel filter and, using both horizontal and vertical sobel filters, obtain a mask containing only the edges of the original image. Tip: Use thresholding with a value around 200 in pixel intensity to get defined edges.

5) Obtain a cartoonized image

Use the edge mask and the original image in order to obtain a 'cartoonized' version of the image. A cartoonized image, in this case, is an image with strengthened black edges, like the one in the example below.

Note: the output image should have the same width and height as the input image.
Use your ImageEditor in main.cpp to apply these filters and save each output using the helper.saveImage method available (remember to set channel parameter properly to save your RGB or grayscale outputs).

You will be graded based on:

- Task completion
- Code organization
- Memory management
- OOP architecture

Upon completion, send back your folder with altered main.cpp along with any files you deem relevant to the interviewers' email. Do not send the .exe file.