CIS 330: Project #3G
Assigned: May 25th, 2018
"Due" May 30th, 2018
BUT: this is not submitted. Will be graded as 3H is graded.
Worth 7% of your grade

Please read this entire prompt!

Add 5 new filters:

1) Mirror
2) Rotate
3) Subtract
4) Grayscale
5) blue

Add 1 new source:

1) Constant color

Add 1 new sink:

1) Checksum

Also: make the two image inputs in Sink be const pointers.

The specifics of the interfaces for filters, source, and sink are specified in the main3G.C file. All of these new filters should work within our existing data flow system (Update/Execute/logging/exceptions/etc).

Place the headers for all of these new modules in "filter.h". (Note that some of them aren't filters … that's OK.)

== Filter 1: Mirror ==

Mirror performs a horizontal (left <-> right) reflection on the image. That is, the left-most column of the input will be the right-most column of the output, the second-to-left column of the input will be the second-to-right column of the output, etc.

== Filter 2: Rotate ==

Crop takes a range of pixels in width and a range of pixels in height and extracts the Rotate performs a 90 degree clockwise rotation on the image. That is, the top row of the input will be the right-most column of the output, the second-to-top row of the input will be the second-to-right column of the output, etc.

If the input has width W and height H, the output will have width H and height W.

== Filter 3: Subtract ==

Subtract finds the difference of two images: input1 - input2

The input images should have exactly the same dimensions. The difference is taken over each color channel. No color channel of the output can be negative, so each subtraction should bottom out at 0. As an example, consider the red color channel of the output at pixel (i,j):

```
if input1(i,j).r > input2(i,j).r  then
    output(i,j).r = input1(i,j).r - input2(i,j).r
else
    0
```

Be careful doing arithmetic with unsigned chars! (See the note for CheckSum.)

== Filter 4: Grayscale ==

Grayscale removes the color from an image. This means that every pixel in the output image has the same "grayscale" value in each of its color channels. Of course, different pixels will have different grayscale values. The exact calculation is:

```
output(i,j) = input(i,j).r / 5  +  input(i,j).g / 2  +  input(i,j).b / 4
```

IMPORTANT: Be sure that you're using integer division for this ... your checksum will not match if you use floating point division.

== Filter 5: Blur ==

Blur will "blur" an image. This means that a pixel in the output image is a sort of average of its neighbors from the input image. As an example, suppose we have a 5 by 5 image. Here is the exact calculation for the output pixel at (2,3):

```
output(2,3) = input(1,2) / 8  +  input(2,2) / 8  +  input(3,2) / 8
            + input(1,3) / 8  +  input(3,3) / 8
            + input(1,4) / 8  +  input(2,4) / 8  +  input(3,4) / 8
```

This calculation should be performed for *each color channel*.

Note that pixels on the edge of the image have a different number of neighbors. Rather than deal with this using a different formula, the pixels on the edge should just be copied from the input to the output. The above formula should only be applied to pixels on the "inside" of the image.

IMPORTANT: Be sure that you're using integer division for this … your checksum will not match if you use floating point division. Because of this, dividing each value by 8 and adding them up is different than adding them up then dividing by 8. Implement the calculation as it's given above!

== Source 1: ConstantColor ==

This takes a color and a size. and produces an image of that size with that color

== Sink 1: CheckSum ==

This sink sums up the total value of the red channel, the blue channel and the green channel and outputs it a file. The name of the file is specified as a command line argument.

VERY IMPORTANT: the sum is to be taken modulo 256. So, for each channel (red, green, blue), it will output a number between 0 and 255.

NOTE: if you sum into an unsigned char, then it automatically does summing modulo 256.

Also: the correct output for 3G should be exactly:

"CHECKSUM: 7, 2, 125\n"
7 would be the sum of the red values (% 256)
2 is the sum of the green values (%256)
125 is the sum of the blue values (%256)

== const pointers ==

Your Sink has two pointers two images. Assume they are called input1 and input2. Their declaration in Sink.h likely looks like:

```
Image *input1;
Image *input2;
```

You should change them to be const:

```
const Image *input1;
const Image *input2;
```

Adding the const is the easy part … the issue is dealing with any compilation issues.

== What to turn in ==

Nothing. This project will be graded as 3H is graded.