

EECS 10: Assignment 5

Prepared by: Che-Wei Chang, Prof. Rainer Doemer

September 2, 2012

Due Monday September 9, 2013 11:00pm. Note: this is a one-week assignment.
--

1 Digital Image Processing [80 points + 20 bonus points]

In this assignment you will learn some basic digital image processing (DIP) techniques by developing an image manipulation program called *PhotoLab*. Using the *PhotoLab*, the user can load an image from a file, apply a set of DIP operations to the image, and save the processed image in a file.

1.1 Introduction

A digital image is essentially a two-dimensional matrix, which can be represented in C by a two-dimensional array of pixels. A pixel is the smallest unit of an image. The color of each pixel is composed of three primary colors, red, green, and blue; each color is represented by an intensity value between 0 and 255. In this assignment, you will work on images with a fixed size, 640×410 , and type, Portable Pixel Map (PPM).

The structure of a PPM file consists of two parts, a header and image data. In the header, the first line specifies the type of the image, P6; the next line shows the width and height of the image; the last line is the maximum intensity value. After the header follows the image data, arranged as RGBRGBRGB..., pixel by pixel in binary representation.

Here is an example of a PPM image file:

```
P6
640 410
255
RGBRGBRGB...
```

1.2 Initial Setup

Before you start working on the assignment, do the following:

```
mkdir hw5
cd hw5
cp ~eecs10/hw5/PhotoLab.c ./
cp ~eecs10/hw5/UCI_Firetrucks.ppm ./
```

NOTE: Please execute the above setup commands only **ONCE** before you start working on the assignment! Do not execute them after you start the implementation, otherwise your code will be overwritten!

The file *PhotoLab.c* is the template file where you get started. It provides the functions for image file reading and saving, test automation as well as the DIP function prototypes and some variables (do not change those function prototypes or variable definitions). You are free to add more variables and functions to the program.

The files *UCI_Firetrucks.ppm* is the PPM images that we will use to test the DIP operations. Once a DIP operation is done, you can save the modified image. You will be prompted for a name of the image. The saved image *name.ppm* will be automatically converted to a JPEG image and sent to the folder *public_html* in your home directory. You are then able to see the image in a web browser at: <http://newport.eecs.uci.edu/~youruserid>, if required names are used (i.e. 'bw', 'negative', 'hflip', 'vflip', 'blur', 'noise', 'zoomin', 'vmirror' for each corresponding function). If you save images by other names, use the link <http://newport.eecs.uci.edu/~youruserid/imagenname.jpg> to access the photo.

Note that whatever you put in the *public_html* directory will be publicly accessible; make sure you don't put files there that you don't want to share, i.e. do not put your source code into that directory.

1.3 Program Specification

In this assignment, your program should be able to read and save image files. To let you concentrate on DIP operations, the functions for file reading and saving are provided. These functions are able to catch many file reading and saving errors, and show corresponding error messages.

Your program is a menu driven program. The user should be able to select DIP operations from a menu as the one shown below:

```
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```

Note: options '9: Zoom in' and '10: Mirror an image vertically' are bonus questions (10 pts each). If you decide to skip these two options, you still need to implement the option '11: Test all functions'.

1.3.1 Load a PPM Image

This option prompts the user for the name of an image file. You don't have to implement a file reading function; just use the provided one, *ReadImage*. Once option 1 is selected, the following should be shown:

```
Please input the file name to load: UCI_Firetrucks
```

After a name, for example *UCI_Firetrucks*, is entered, the *PhotoLab* will load the file *UCI_Firetrucks.ppm*. Note that, in this assignment please always enter file names without the extension when you load or save a file (i.e. enter 'UCI_Firetrucks', instead of 'UCI_Firetrucks.ppm'). If it is read correctly, the following is shown:

```
please make your choice: 1
Please input the file name to load: UCI_Firetrucks
UCI_Firetrucks.ppm was read successfully!
```

```
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```

Then, you can select other options. If there is a reading error, for example the file name is entered incorrectly or the file does not exist, the following message is shown:

```
please make your choice: 1
Please input the file name to load: UCI_Firetrucks.ppm
Cannot open file "UCI_Firetrucks.ppm" for reading!
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```

In this case, try option 1 again with the correct filename.

1.3.2 Save a PPM Image

This option prompts the user for the name of the target image file. You don't have to implement a file saving function; just use the provided one, *SaveImage*. Once option 2 is selected, the following is shown:

```
please make your choice: 2
Please input the file name to save: bw
bw.ppm was saved successfully.
bw.jpg was stored for viewing.
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
```

- 7: Blur an image
- 8: Add noise to an image
- 9: Zoom in (Bonus)
- 10: Mirror an image vertically (Bonus)
- 11: Test all functions
- 12: Exit

please make your choice:

The saved image will be automatically converted to a JPEG image and sent to the folder *public.html*. You then are able to see the image at: <http://newport.eecs.uci.edu/~youruserid> (For off campus, the link is: <http://newport.eecs.uci.edu/~youruserid/imagename.jpg>)

1.3.3 Change a Color Image to Black and White



(a) Color image



(b) Black and white image

Figure 1: A color image and its black and white counterpart.

A black and white image is the one that the intensity values are the same for all color channels, red, green, and blue, at each pixel. To change a color image to grey, assign a new intensity, which is given by $(R + G + B)/3$, to all the color channels at a pixel. The R, G, B are the old intensity values for the red, the green, and the blue channels at the pixel. You need to define and implement the following function to do the job.

```
/* change color image to black and white */
void BlackNWhite(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);
```

Figure 1 shows an example of this operation. Your program's output for this option should be like:

```
Please make your choice: 3
"Black & White" operation is done!
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
```

```

10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:

```

Save the image with name 'bw' after this step.

1.3.4 Make a negative of an image

A negative image is an image in which all the intensity values have been inverted. To achieve this, each intensity value at a pixel is subtracted from the maximum value, 255, and the result is assigned to the pixel as a new intensity. You need to define and implement a function to do the job. You need to define and implement the following function to do this DIP.

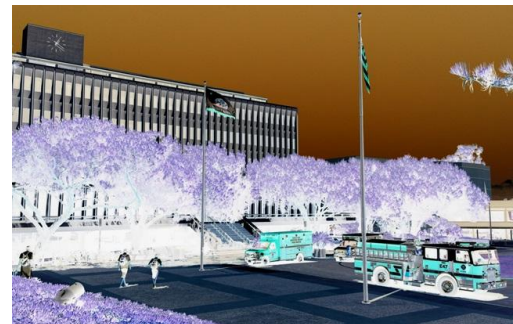
```

/* reverse image color */
void Negative(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
             unsigned char B[WIDTH][HEIGHT]);

```



(a) Original image



(b) Negative image

Figure 2: An image and its negative counterpart.

Figure 2 shows an example of this operation. Your program's output for this option should be like:

```

Please make your choice: 4
"Negative" operation is done!
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:

```

Save the image with name 'negative' after this step.

1.3.5 Flip Image Horizontally



(a) Original image



(b) Horizontally flipped image

Figure 3: An image and its horizontally flipped counterpart.

To flip an image horizontally, the intensity values in horizontal direction should be reversed. The following shows an example.

	1	2	3	4	5		5	4	3	2	1
before horizontal flip:	0	1	2	3	4	after horizontal flip:	4	3	2	1	0
	3	4	5	6	7		7	6	5	4	3

You need to define and implement the following function to do this DIP.

```
/* flip image horizontally */  
void HFlip(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],  
unsigned char B[WIDTH][HEIGHT]);
```

Figure 3 shows an example of this operation. Your program's output for this option should be like:

```
Please make your choice: 5  
"HFlip" operation is done!  
-----  
1: Load a PPM image  
2: Save an image in PPM and JPEG format  
3: Change a color image to black and white  
4: Make a negative of an image  
5: Flip an image horizontally  
6: Flip an image vertically  
7: Blur an image  
8: Add noise to an image  
9: Zoom in (Bonus)  
10: Mirror an image vertically (Bonus)  
11: Test all functions  
12: Exit  
please make your choice:
```

Save the image with name 'hflip' after this step.

1.3.6 Flip Image Vertically

To flip an image vertically, the intensity values in vertical direction should be reversed. The following shows an example.

```
           1 2 3 4 5           3 4 5 6 7
before vertical flip: 0 1 2 3 4   after vertical flip: 0 1 2 3 4
                   3 4 5 6 7                   1 2 3 4 5
```

You need to define and implement the following function to do this DIP.

```
/* flip image vertically */
void VFlip(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);
```

Figure 4 shows an example of this operation. Your program's output for this option should be like:



(a) Original image



(b) Vertically flipped image

Figure 4: An image and its vertically flipped counterpart.

Please make your choice: 6

"VFlip" operation is done!

```
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```

Save the image with name 'vflip' after this step.



(a) Original Image



(b) Image with blur

Figure 5: An aged image and its blur counterpart.

1.3.7 Blur

The blurring works this way: the intensity value at each pixel is mapped to a new value, which is the average of itself and its 8 neighbours. The following shows an example:

```
X X X X X
X 4 1 2 X
X 2 0 1 X
X 4 1 3 X
X X X X X
```

To blur the image, the intensity of the center pixel with the value of 0 is changed to $(4 + 1 + 2 + 2 + 0 + 1 + 4 + 1 + 3)/9 = 2$. Repeat this for every pixel, and for every color channel (red, green, and blue) of the image. You need to define and implement a function to do this DIP.

Note that special care has to be taken for pixels located at the image boundaries. For ease of implementation, you may choose to ignore the pixels at the border of the image where no neighbor pixels exist.

The blurred image should look like the figure shown in Figure 5(b):

Please enter your choice: 7

"Blur" operation is done!

```
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```


1.3.8 Add Noise to Image

In this operation, you add noise to an image. The noise added is a special kind, called salt-and-pepper noise, which means the noise is either black or white. You need to define and implement a function to do the job. If the percentage of noise is n , then the number of noise added to the image is given by $n * WIDTH * HEIGHT / 100$, where $WIDTH$ and $HEIGHT$ are the image size. You need the knowledge of random number generator from the previous assignments. Figure 7 shows an example of this operation with n set to 20%.

Once user chooses this option, your program's output should be like:

```
Plelase enter your choice: 8
"Blur" operation is done!
```

```
-----
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to black and white
4: Make a negative of an image
5: Flip an image horizontally
6: Flip an image vertically
7: Blur an image
8: Add noise to an image
9: Zoom in (Bonus)
10: Mirror an image vertically (Bonus)
11: Test all functions
12: Exit
please make your choice:
```



(a) Original image



(b) Noise setting: $n=20$

Figure 6: An image and its noise (salt-and-pepper) corrupted counterpart.

Implementation hint: calculate first the number of noise pixels depending on the noise percentage; then, use the random number generator to determine the position (x and y coordinate) of each noise pixel. Create a white or a black pixel by setting the intensity value at each color channel to its maximum (255) or minimum (0) respectively. Note: the number of white pixels should be approximately equal to the number of black pixels.



(a) Original image (source:Freedigitalphotos.net)



(b) 2x Zoom-In image

Figure 7: An image and its 2x Zoom-In counterpart

1.3.9 Zoom-In

In this section, you need to implement a simple zoom-in 2x function, that enlarge the right-bottom part of the image by a factor of two.

To zoom in, the intensity values in the center of the picture are distributed to the whole picture. The following shows an example:

	1	0	5	10		7	7	12	12
	2	1	6	11		7	7	12	12
before zoom-in:	3	2	7	12	after zoom-in:	8	8	13	13
	4	3	8	13		8	8	13	13

You need to define and implement a function to do this DIP. Figure 7 shows an example of this operation.

Please enter your choice: 7

"Zoom in 2x" operation is done!

- 1: Load a PPM image
- 2: Save the image in PPM and JPEG format
- 3: Make a negative of an image
- 4: Flip the image horizontally
- 5: Flip the image vertically
- 6: Rotate the image 180 degrees
- 7: Zoom in
- 8: Image Overlay
- 9: Age the image
- 10: Blur the image (10 extra points)
- 11: Detect image edges (10 extra points)
- 12: Exit

please enter your choice:

1.3.10 Mirror Image Vertically (bonus points: 10 pts)

To mirror an image vertically, the intensity values in horizontal direction at the bottom should be reversed and copied to the top. The following shows an example.

	1	2	3	4	5		1	2	3	4	5
before horizontal mirror:	4	3	2	1	0	after horizontal mirror:	4	3	2	1	0
	3	4	5	6	7		1	2	3	4	5

You need to define and implement the following function to do this DIP.

```
/* mirror image vertically */  
void VMirror(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],  
unsigned char B[WIDTH][HEIGHT]);
```

Figure 8 shows an example of this operation. Your program's output for this option should be like:

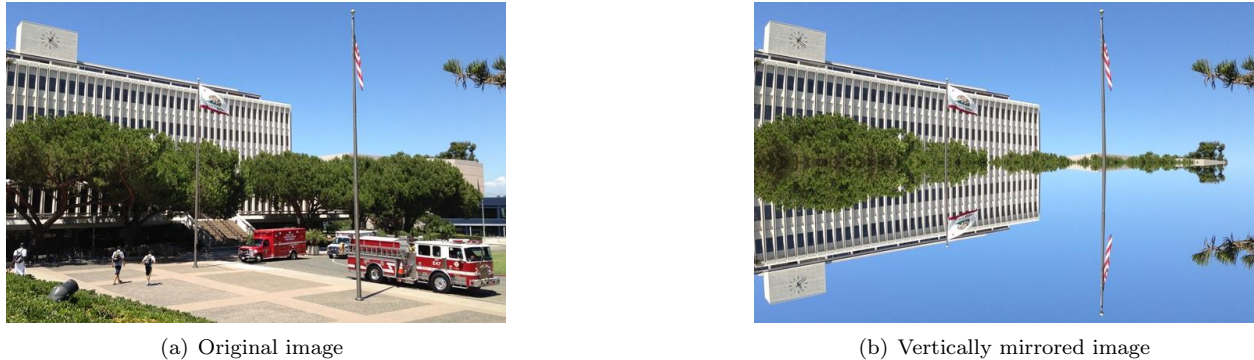


Figure 8: An image and its vertically mirrored counterpart.

Please make your choice: 10
"VMirror" operation is done!

```
-----  
1: Load a PPM image  
2: Save an image in PPM and JPEG format  
3: Change a color image to black and white  
4: Make a negative of an image  
5: Flip an image horizontally  
6: Flip an image vertically  
7: Blur an image  
8: Add noise to an image  
9: Zoom in (Bonus)  
10: Mirror an image vertically (Bonus)  
11: Test all functions  
12: Exit  
please make your choice:
```

Save the image with name 'vmirror' after this step.

1.3.11 Test all functions

Finally, you are going to write a function to test all previous functions. In this function, you are going to call DIP functions one by one and to observe the results. The function is for the designer to quickly test the program, so you should supply all necessary parameters when testing. The function should look like:

```
void AutoTest(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],  
unsigned char B[WIDTH][HEIGHT])
```

```

{
char fname[SLEN] = "UCI_Firetrucks";
char sname[SLEN];

ReadImage(fname, R, G, B);
BlackNWhite(R, G, B);
strcpy(sname, "bw");
SaveImage(sname, R, G, B);
printf("Black & White tested!\n\n");

ReadImage(fname, R, G, B);
Negative(R, G, B);
strcpy(sname, "bw");
SaveImage(sname, R, G, B);
printf("Black & White tested!\n\n");

...
ReadImage(fname, R, G, B);
AddNoise(20, R, G, B);
strcpy(sname, "bw");
SaveImage(sname, R, G, B);
printf("Black & White tested!\n\n");

...
...
ReadImage(fname, R, G, B);
VMirror(R, G, B);
strcpy(sname, "vmirror");
SaveImage(sname, R, G, B);
printf("VMirror tested!\n\n");

...
}

```

Once user chooses this option, your program's output should be like:

```

please make your choice: 11
UCI_Firetrucks.ppm was read successfully!
bw.ppm was saved successfully.
bw.jpg was stored for viewing.
Black & White tested!

```

```

UCI_Firetrucks.ppm was read successfully!
negative.ppm was saved successfully.
negative.jpg was stored for viewing.
Negative tested!

```

```

...
...

```

1.4 Implementation

1.4.1 Function Prototypes

For this assignment, you need the following functions (those function prototypes (declarations) are already provided in *PhotoLab.c*. Please do not change them):

```
/* print a menu */
void PrintMenu();

/* read image from a file */
int ReadImage(char fname[SLEN], unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* save a processed image */
int SaveImage(char fname[SLEN], unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* change color image to black & white */
void BlackNWhite(unsigned char R[WIDTH][HEIGHT],
                 unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* reverse image color */
void Negative(unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* flip image horizontally */
void HFlip(unsigned char R[WIDTH][HEIGHT],
           unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* flip image vertically */
void VFlip(unsigned char R[WIDTH][HEIGHT],
           unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* blur the image */
void Blur(unsigned char R[WIDTH][HEIGHT],
          unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* add noise */
void AddNoise(int n, unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* zoom in */
void Zoomin(unsigned char R[WIDTH][HEIGHT],
            unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* mirror image vertically */
void VMirror(unsigned char R[WIDTH][HEIGHT],
             unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);

/* Test all functions */
void AutoTest(unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);
```

You may want to define other functions as needed.

NOTE: The *ReadImage()*, *WriteImage()*, and *Aging()* functions are already defined in *PhotoLab.c*. The *AutoTest()* function is partially defined (You need to complete the definition of *AutoTest()*). The *AutoTest()* function is called in the *main()* function without the menu. This is just an example to show how to call the *AutoTest()* function. You need to add the code for the menu of this program and use proper function calls for different DIP operations.

1.4.2 Global constants

You also need the following global constants (they are also declared in *PhotoLab.c*, please don't change their names):

```
#define WIDTH 640 /* Image width */
#define HEIGHT 410 /* image height */
#define SLEN 80 /* maximum length of file names */
```

1.4.3 Pass in arrays by reference

In the main function, three two-dimensional arrays are defined. They are used to save the RGB information for the current image:

```
int main()
{
unsigned char R[WIDTH][HEIGHT]; /* for image data */
unsigned char G[WIDTH][HEIGHT];
unsigned char B[WIDTH][HEIGHT];
}
```

When any of the DIP operations is called in the main function, those three arrays: *R[WIDTH][HEIGHT]*, *G[WIDTH][HEIGHT]*, *B[WIDTH][HEIGHT]* are the parameters passed into the DIP functions. Since arrays are passed by reference, any changes to *R[][]*, *G[][]*, *B[][]* in the DIP functions will be applied to those variables in the main function. In this way, the current image can be updated by DIP functions without defining global variables.

In your DIP function implementation, there are two ways to save the target image information in *R[][]*, *G[][]*, *B[][]*. Both options work and you should decide which option is better based on the specific DIP manipulation function at hand.

Option 1: using local variables You can define local variables to save the target image information. For example:

```
void DIP_function_name()
{
unsigned char RT[WIDTH][HEIGHT]; /* for target image data */
unsigned char GT[WIDTH][HEIGHT];
unsigned char BT[WIDTH][HEIGHT];
}
```

Then, at the end of each DIP function implementation, you should copy the data in *RT[][]*, *GT[][]*, *BT[][]* over to *R[][]*, *G[][]*, *B[][]*.

Option 2: in place manipulation Sometimes you do not have to create new local array variables to save the target image information. Instead, you can just manipulate on $R[][]$, $G[][]$, $B[][]$ directly. For example, in the implementation of `Negative()` function, you can assign the result of 255 minus each pixel value directly back to this pixel entry.

2 Script File

To demonstrate that your program works correctly, perform the following steps and submit the log as your script file:

1. Start the script by typing the command: *script*
2. Compile and run your program
3. Choose 'Test all functions' (The file names must be 'bw', 'negative', 'hflip', 'hmirror', 'blur', 'noise', 'zoomin', 'vmirror' for the corresponding function)
4. Exit the program
5. Stop the script by typing the command: *exit*
6. Rename the script file to *PhotoLab.script*

NOTE: make sure use exactly the same names as shown in the above steps when saving modified images! The script file is important, and will be checked in grading; you must follow the above steps to create the script file.

3 Submission

Use the standard submission procedure to submit the following files:

- *PhotoLab.c* (with your code filled in!)
- *PhotoLab.script*

Please leave the images generated by your program in your *public_html* directory. Don't delete them as we may consider them when grading! You don't have to submit any images.