EECS 10: Assignment 7

Prepared by: Xu Han, Prof. Rainer Doemer

November 12, 2010

Due on Monday 11/29/2010 12:00pm. Note: this is a two-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, 480×640, 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
480 640
255
RGBRGBRGB...
```

1.2 Initial Setup

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

```
cd ~
mkdir -p hw7
cd hw7
cp ~eecs10/hw7/PhotoLab.c .
cp ~eecs10/hw7/swan.ppm .
cp ~eecs10/hw7/anteater.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 *swan.ppm* and *anteater.ppm* are 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', 'flip', 'mirror', 'noise', 'color', 'overlay', 'blur', 'border' for each corresponding function). If you save images by other names, use the link *http://newport.eecs.uci.edu/∼youruserid/imagename.jpg* to access the photo.

Note that whatever you put in the *public_html* directory will be publicly accessable; 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 (like the previous assignment). 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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

Note: options '10: Blur an image' and '11: Add border to an image' are bonus questions (10 pts each). If you decide to skip these two options, you still need to implement the option '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: swan
```

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

```
Please make your choice: 1
Please input the file name to load: swan
swan.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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  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:

```
Cannot open file "swan.ppm.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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  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*)

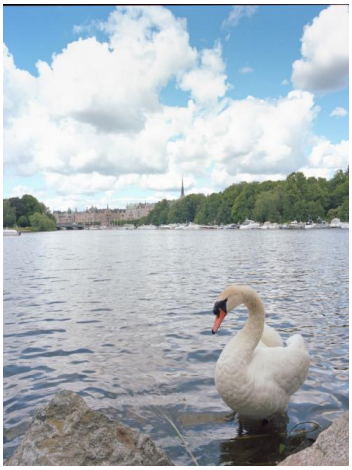
1.3.3 Change a Color Image to Black and White

A 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
```



(a) Color image

(b) Black and white image

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

```
 6:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

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

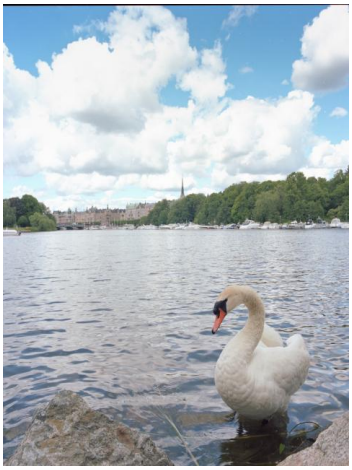
1.3.4 Make a negative of an image

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.

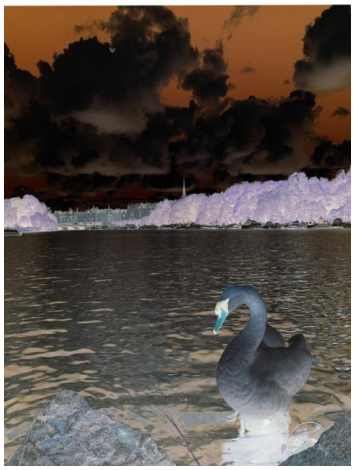
```
/* reverse image color */
void Negative(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT]);
```

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

```
Please make your choice: 4
```



(a) Original image



(b) Negative image

Figure 2: An image and its negative counterpart.

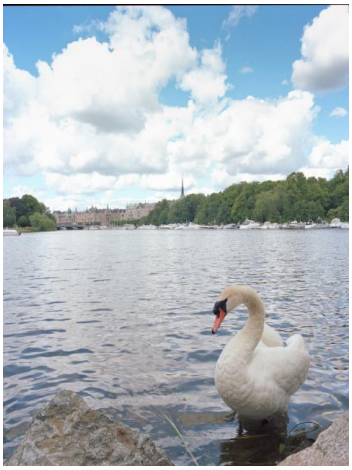
```
"Negative" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Make a negative of an image
 5:  Flip an image horizontally
 6:  Mirror an image horizontally
 7:  Add noise to an image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

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

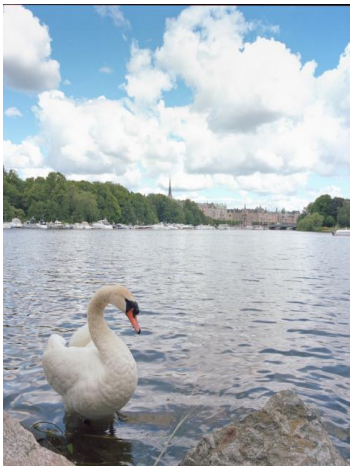
1.3.5 Flip Image Horizontally

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
```



(a) Original image



(b) Horizontally flipped image

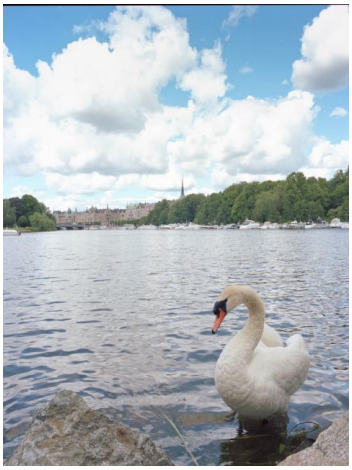
Figure 3: An image and its horizontally flipped counterpart.

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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```



(a) Original image

(b) Horizontally mirrored image

Figure 4: An image and its horizontally mirrored counterpart.

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

1.3.6 Mirror Image Horizontally

To mirror an image horizontally, the intensity values in horizontal direction on the right side should be reversed and copied to the left side. The following shows an example.

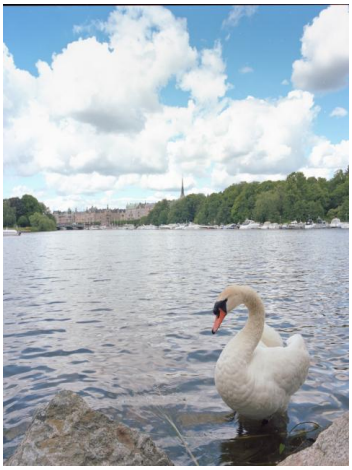
```
                          1 2 3 4 5                                5 4 3 4 5
  before horizontal mirror:3 4 2 1 0      after horizontal mirror:0 1 2 1 0
                          3 4 5 6 7                                7 6 5 6 7
```

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

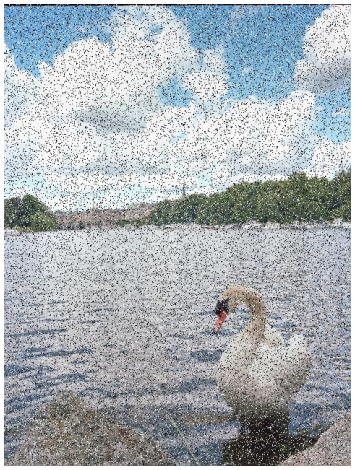
```
/* mirror image horizontally */
void HMirror(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);
```

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

```
Please make your choice: 6
"HMirror" 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
```



(a) Original image



(b) Noise setting: n=20

Figure 5: An image and its noise (salt-and-pepper) corrupted counterpart.

```
 6:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

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

1.3.7 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 5 shows an example of this operation with n set to 20%.

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

```
/* add salt-and-pepper noise to image */
void AddNoise(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
```

```
     unsigned char B[WIDTH][HEIGHT], int percentage);
```

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

```
Please make your choice: 7
Enter the percentage of noise: 20
"AddNoise" 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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

Implementation hint: Firstly, you need to calculate the number of noise pixels depending on the noise percentage. Secondly, use the random number generator to determine the position (x and y coordinate) of each noise pixel; note that x and y are integers and they should be in the range of [0, WIDTH-1] and [0, HEIGHT-1] respectively. Then you 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 that the number of white pixels should be approximately equal to the number of black pixels, so you should create white and black noise pixels alternatively.

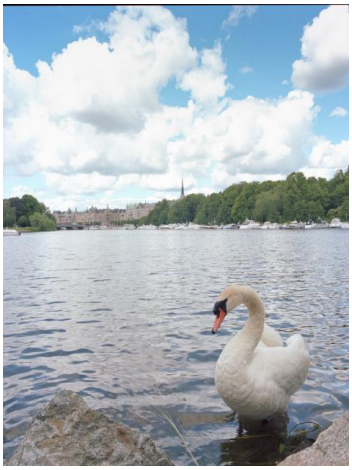
Save the image with name 'noise' after this step.

1.3.8 Color Correct

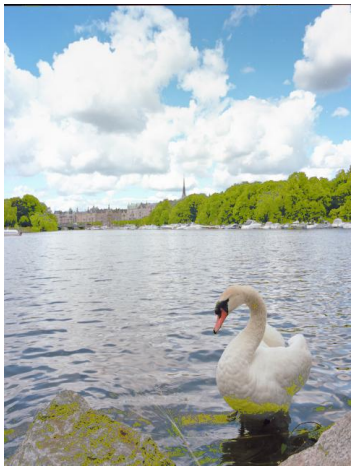
This operation will modify a kind of color in the image, for example, changing all the white to pink. The user will input a target color in form of R, G, B values and three offset values for each component. Your program must firstly search through the picture and locate similar colors, i.e. find colors in the range of (R-range, R+range), (G-range, G+range), (B-range, B+range). We set the range to a contant of 30 in this assignment (e.g. int range = 30;). Once you locate the color in question, use the offset values to correct the color by adding the offset to the original color. Make sure your resulting R/G/B is still in the range of [0, 255]. Figure 6 shows an example of this operation with target color (90, 110, 80) and offset (40, 40, -60).

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

```
/* correct a color in the image */
void ColorCorrect(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT],int R_target, int G_target, int B_target,
     int R_offset, int G_offset, int B_offset);
```



(a) Original image



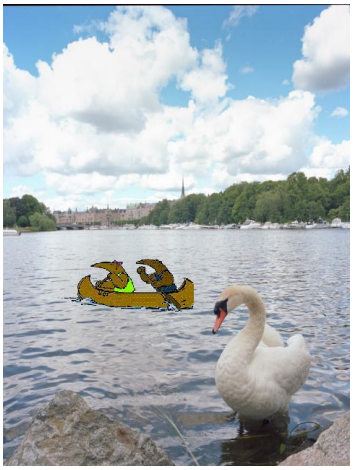
(b) Target color (90, 110, 80) offset (40, 40, -60).

Figure 6: An image and its color corrected counterpart.

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

```
Please make your choice:8
Please enter target color in form of R, G, B:
 R: 90
 G: 110
 B: 80
Please enter offset value to target R, G, B:
 Offset to R: 40
 Offset to G: 40
 Offset to B: -60
"ColorCorrect" 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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
```

(a) A second image



(b) Overlay two image at position (80, 350)

Figure 7: A image and the overlayed image.

```
Please make your choice:
```

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

1.3.9 Image Overlay

This function overlies the current image with a second image. In our program, we will put an image of two anteaters on the original image.

To start the implementation, you need to promt the user to enter the file name of the second image first, and then you read that image in the beginning of the overlay function (by using ReadImageW()). In this assignment, the second image is 'anteater.ppm'(184 by 92 pixels). Since the image is much smaller than the original one, the user also need to enter the position of overlay with coordinates (x, y).

Take a look at the second image at Figure 7. Note that it has a white background and the blue water part which are inconsistent with our original image. To achieve the overlay effect, we will treat the background and the blue part in the second image as transparent color. That is, each of the non-background/non-blue pixels in anteater.ppm will be overlayed to a position in the original image, whereas background/blue pixels will stay as in the original image. Whether or not a pixel in anteater.ppm is a background/blue pixel can be decided by the RGB values of this pixel. More specifically, if a pixel has RGB value of (255, 255 ,255) which represents white backgroup, or (0, 152, 253) which represents the blue water part, this pixel should not be put onto original image.

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

```
/* Load an image and Overlay it on the current image */
void Overlay(char fname[SLEN], unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT], int x_offset, int y_offset);
```

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

```
Please make your choice: 9
Please input the file name for the second image: anteater
Please input x coordinate of the overlay image: 80
Please input y coordinate of the overlay image: 350
anteater.ppm was read successfully!
"Image Overlay" 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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

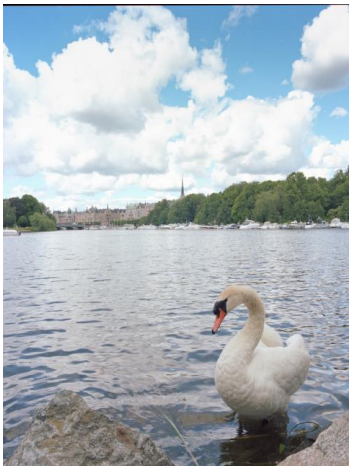
The effect can be seen in Figure 7 when position is chosen as (80, 350):
Save the image with name 'overlay' after this step.

1.3.10 Blur (bonus points: 10pts)

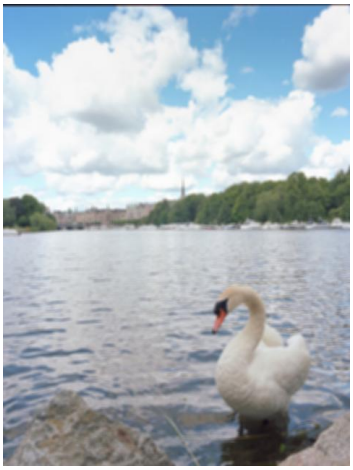
The blurring works this way: the intensity value at each pixel is mapped to a new value, which is the average of itself and its 24 neighbours. The following shows an example:

```
x x x x x x x
x 9 6 3 5 1 x
x 7 4 1 2 3 x
x 1 2 0 1 8 x
x 5 4 1 3 7 x
x 8 6 5 4 2 x
x x x x x x x
```

To blur the image, the intensity of the center pixel with the value of 0 is changed to $(9+6+3+5+1+7+4+1+2+3+1+2+0+1+8+5+4+1+3+7+8+6+5+4+2)/25 = 3$. Repeat this for every pixel, and for every color channel (red, green, and blue) of the image. You need to define and implement the following function to do this DIP.



(a) Original image



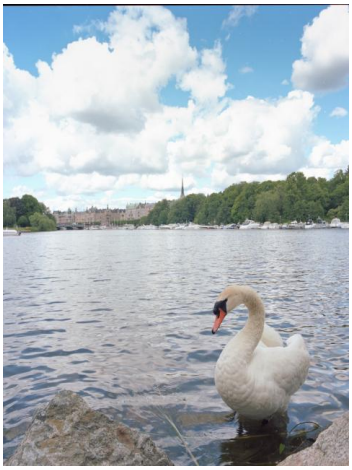
(b) Blured image

Figure 8: An image and its blurred counterpart.

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

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 neighbour pixels exist.
After the two process, the blurred image should look like the figure shown in Figure 8(b):

```
Please make your choice: 10
"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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```



(a) Original image

(b) Image with borders, border color = black, width = 10 pixels

Figure 9: An image and its counterpart when borders are added.

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

1.3.11 Add borders to an image (bonus points: 10pts)

This operation will add borders to the current image. The border color and width (in pixels) of the borders are parameters given by users. Figure 9 shows an example of adding borders to an image.

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

```
/* add a border to the image */
void AddBorder(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT], int r, int g, int b, int bwidth);
```

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

```
Please make your choice: 11
Enter the R value of the border color(0 to 255): 0
Enter the G value of the border color(0 to 255): 0
Enter the B value of the border color(0 to 255): 0
Enter the width of the border: 10
"Add Border" 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:  Mirror an image horizontally
 7:  Add noise to an image
 8:  Perform color correction
 9:  Image Overlay
10:  Blur an image
11:  Add border to an image
12:  Test all functions
13:  Exit
Please make your choice:
```

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

1.3.12 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] = "swan";
 char fname2[SLEN] = "anteater";
 char sname[SLEN];

 ReadImage(fname, R, G, B);
 BlackNWhite(R, G, B);
 strcpy(sname, "bw");/*string copy function to prepare the file name to be saved*/
 SaveImage(sname, R, G, B);
 printf("Black & White tested!\n\n");

...
...
 ReadImage(fname, R, G, B);
 Overlay(fname2, R, G, B, 80, 350);
 strcpy(sname, "overlay");
 SaveImage(sname, R, G, B);
 printf("Overlay tested!\n\n");

...
}
```

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

```
Please make your choice: 12
```

```
swan.ppm was read successfully!
bw.ppm was saved successfully.
bw.jpg was stored for viewing.
Black & White tested!

swan.ppm was read successfully!
"Negative" operation is done!
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 to define the following functions (those function prototypes are already provided in *PhotoLab.c*. Please do not change them):

```
/*** function declarations ***/
/* 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]);
/* read Overlay image from a file */
int ReadImageW(char fname[SLEN], unsigned char R[WIDTH2][HEIGHT2],
    unsigned char G[WIDTH2][HEIGHT2], unsigned char B[WIDTH2][HEIGHT2]);

/* 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]);
```

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

/* add salt-and-pepper noise to image */
void AddNoise(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT], int percentage);

/* correct a color in the image */
void ColorCorrect(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT],int R_target, int G_target, int B_target,
     int R_offset, int G_offset, int B_offset);

/* Load an image and Overlay it on the current image */
void Overlay(char fname[SLEN], unsigned char R[WIDTH][HEIGHT],
     unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT],
     int x_offset, int y_offset);

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

/* add a border to the image */
void AddBorder(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
     unsigned char B[WIDTH][HEIGHT], int r, int g, int b, int bwidth);

/* 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.

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  480 /* Image width */
#define HEIGHT 640 /* image height */
#define SLEN    80 /* maximum length of file names */
#define WIDTH2 184 /* Overlay Image width */
#define HEIGHT2 92 /* Overlay Image height */
```

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', 'flip', 'mirror', 'noise','color', 'overlay', 'blur','border' 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.